Contents

1 Introduction 2
1.1 Abstract . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.2 Purpose . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.3 Scope of Specification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.4 Normative References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.5 Version/Change History . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.6 Deprecation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.7 Requirements Language . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.8 Typographical Conventions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.8.1 Naming Conventions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.8.2 Vendor Naming Conventions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
1.9 Glossary and Acronyms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
1.10 Acknowledgements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

2 Design Overview 7
2.1 Software Landscape . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
2.2 Design Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
2.2.1 Hardware Optimizations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.2.2 Hardware Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.3 Assumptions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.3.1 Portability . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.3.2 Opaqueness . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.4 Object-Oriented Behaviors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.5 OpenVX Framework Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.6 OpenVX Data Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
2.7 Error Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.8 Graphs Concepts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.8.1 Linking Nodes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.8.2 Virtual Data Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.8.3 Node Parameters . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
2.8.4 Graph Parameters . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
2.8.5 Execution Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.8.6 Asynchronous Mode . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.8.7 Node Execution Independence . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.8.8 Verification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.9 Callbacks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.10 User Kernels . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.10.1 Parameter Validation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
2.10.2 User Kernels Naming Conventions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
2.11 Immediate Mode Functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.12 Targets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.13 Base Vision Functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.13.1 Inputs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.13.2 Outputs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
## 3 Module Documentation

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Vision Functions</td>
<td>31</td>
</tr>
<tr>
<td>3.1.1 Detailed Description</td>
<td>31</td>
</tr>
<tr>
<td>3.2 Absolute Difference</td>
<td>35</td>
</tr>
<tr>
<td>3.2.1 Detailed Description</td>
<td>35</td>
</tr>
<tr>
<td>3.2.2 Function Documentation</td>
<td>35</td>
</tr>
<tr>
<td>vxAbsDiffNode(vx_graph graph, vx_image in1, vx_image in2, vx_image out)</td>
<td>35</td>
</tr>
<tr>
<td>vxuAbsDiff(vx_context context, vx_image in1, vx_image in2, vx_image out)</td>
<td>35</td>
</tr>
<tr>
<td>3.3 Accumulate</td>
<td>37</td>
</tr>
<tr>
<td>3.3.1 Detailed Description</td>
<td>37</td>
</tr>
<tr>
<td>3.3.2 Function Documentation</td>
<td>37</td>
</tr>
<tr>
<td>vxAccumulateImageNode(vx_graph graph, vx_image input, vx_image accum)</td>
<td>37</td>
</tr>
<tr>
<td>vxuAccumulateImage(vx_context context, vx_image input, vx_image accum)</td>
<td>37</td>
</tr>
<tr>
<td>3.4 Accumulate Squared</td>
<td>39</td>
</tr>
<tr>
<td>3.4.1 Detailed Description</td>
<td>39</td>
</tr>
<tr>
<td>3.4.2 Function Documentation</td>
<td>39</td>
</tr>
<tr>
<td>vxAccumulateSquareImageNode(vx_graph graph, vx_image input, vx_scalar shift, vx_image accum)</td>
<td>39</td>
</tr>
<tr>
<td>vxuAccumulateSquareImage(vx_context context, vx_image input, vx_scalar shift, vx_image accum)</td>
<td>39</td>
</tr>
<tr>
<td>3.5 Data Object Copy</td>
<td>41</td>
</tr>
<tr>
<td>3.5.1 Detailed Description</td>
<td>41</td>
</tr>
<tr>
<td>3.5.2 Function Documentation</td>
<td>41</td>
</tr>
<tr>
<td>vxCopyNode(vx_graph graph, vx_reference input, vx_reference output)</td>
<td>41</td>
</tr>
<tr>
<td>vxuCopy(vx_context context, vx_reference input, vx_reference output)</td>
<td>41</td>
</tr>
<tr>
<td>3.6 Accumulate Weighted</td>
<td>43</td>
</tr>
<tr>
<td>3.6.1 Detailed Description</td>
<td>43</td>
</tr>
<tr>
<td>3.6.2 Function Documentation</td>
<td>43</td>
</tr>
<tr>
<td>vxAccumulateWeightedImageNode(vx_graph graph, vx_image input, vx_scalar alpha, vx_image accum)</td>
<td>43</td>
</tr>
<tr>
<td>vxuAccumulateWeightedImage(vx_context context, vx_image input, vx_scalar alpha, vx_image accum)</td>
<td>43</td>
</tr>
<tr>
<td>3.7 Control Flow</td>
<td>45</td>
</tr>
<tr>
<td>3.7.1 Detailed Description</td>
<td>45</td>
</tr>
<tr>
<td>3.7.2 Function Documentation</td>
<td>46</td>
</tr>
<tr>
<td>vxScalarOperationNode(vx_graph graph, vx_enum scalar_operation, vx_scalar a, vx_scalar b, vx_scalar output)</td>
<td>46</td>
</tr>
<tr>
<td>vxSelectNode(vx_graph graph, vx_scalar condition, vx_reference true_value, vx_reference false_value, vx_reference output)</td>
<td>46</td>
</tr>
</tbody>
</table>
3.8 Arithmetic Addition

3.8.1 Detailed Description

3.8.2 Function Documentation
vxAddNode(vx_graph graph, vx_image in1, vx_image in2, vx_enum policy, vx_image out) 48
vxuAdd(vx_context context, vx_image in1, vx_image in2, vx_enum policy, vx_image out) 48

3.9 Arithmetic Subtraction

3.9.1 Detailed Description

3.9.2 Function Documentation
vxSubtractNode(vx_graph graph, vx_image in1, vx_image in2, vx_enum policy, vx_image out) 50
vxuSubtract(vx_context context, vx_image in1, vx_image in2, vx_enum policy, vx_image out) 50

3.10 Bitwise AND

3.10.1 Detailed Description

3.10.2 Function Documentation
vxAndNode(vx_graph graph, vx_image in1, vx_image in2, vx_image out) 52
vxuAnd(vx_context context, vx_image in1, vx_image in2, vx_image out) 52

3.11 Bitwise EXCLUSIVE OR

3.11.1 Detailed Description

3.11.2 Function Documentation
vxXorNode(vx_graph graph, vx_image in1, vx_image in2, vx_image out) 54
vxuXor(vx_context context, vx_image in1, vx_image in2, vx_image out) 54

3.12 Bitwise INCLUSIVE OR

3.12.1 Detailed Description

3.12.2 Function Documentation
vxOrNode(vx_graph graph, vx_image in1, vx_image in2, vx_image out) 56
vxuOr(vx_context context, vx_image in1, vx_image in2, vx_image out) 56

3.13 Bitwise NOT

3.13.1 Detailed Description

3.13.2 Function Documentation
vxNotNode(vx_graph graph, vx_image input, vx_image output) 58
vxuNot(vx_context context, vx_image input, vx_image output) 58

3.14 Box Filter

3.14.1 Detailed Description

3.14.2 Function Documentation
vxBox3x3Node(vx_graph graph, vx_image input, vx_image output) 60
vxuBox3x3(vx_context context, vx_image input, vx_image output) 60

3.15 Non-Maxima Suppression

3.15.1 Detailed Description

3.15.2 Function Documentation
vxNonMaxSuppressionNode(vx_graph graph, vx_image input, vx_image mask, vx_int32 win_size, vx_enum norm_type, vx_image output) 62
vxuNonMaxSuppression(vx_context context, vx_image input, vx_image mask, vx_int32 win_size, vx_image output) 62

3.16 Canny Edge Detector

3.16.1 Detailed Description

3.16.2 Enumeration Type Documentation
vx_norm_type_e

3.16.3 Function Documentation
vxCannyEdgeDetectorNode(vx_graph graph, vx_image input, vx_threshold hyst, vx_int32 gradient_size, vx_enum norm_type, vx_image output) 65
vxuCannyEdgeDetector(vx_context context, vx_image input, vx_threshold hyst, vx_int32 gradient_size, vx_enum norm_type, vx_image output) 65

3.17 Channel Combine

3.17.1 Detailed Description

3.17.2 Function Documentation
vxChannelCombineNode(vx_graph graph, vx_image plane0, vx_image plane1, vx_image plane2, vx_image plane3, vx_image output) 67
vxuChannelCombine(vx_context context, vx_image plane0, vx_image plane1, vx_image plane2, vx_image plane3, vx_image output) 67
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.18</td>
<td>Channel Extract</td>
</tr>
<tr>
<td>3.18.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.18.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>vxChannelExtractNode(vx_graph graph, vx_image input, vx_enum channel, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>vxuChannelExtract(vx_context context, vx_image input, vx_enum channel, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>3.19</td>
<td>Color Convert</td>
</tr>
<tr>
<td>3.19.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.19.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>vxColorConvertNode(vx_graph graph, vx_image input, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>vxuColorConvert(vx_context context, vx_image input, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>3.20</td>
<td>Convert Bit depth</td>
</tr>
<tr>
<td>3.20.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.20.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>vxConvertDepthNode(vx_graph graph, vx_image input, vx_image output, vx_enum policy, vx_int32 shift)</td>
<td></td>
</tr>
<tr>
<td>vxuConvertDepth(vx_context context, vx_image input, vx_image output, vx_enum policy, vx_int32 shift)</td>
<td></td>
</tr>
<tr>
<td>3.21</td>
<td>Custom Convolution</td>
</tr>
<tr>
<td>3.21.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.21.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>vxConvolveNode(vx_graph graph, vx_image input, vx_convolution conv, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>vxuConvolve(vx_context context, vx_image input, vx_convolution conv, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>3.22</td>
<td>Dilate Image</td>
</tr>
<tr>
<td>3.22.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.22.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>vxDilate3x3Node(vx_graph graph, vx_image input, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>vxuDilate3x3(vx_context context, vx_image input, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>3.23</td>
<td>Equalize Histogram</td>
</tr>
<tr>
<td>3.23.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.23.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>vxEqualizeHistNode(vx_graph graph, vx_image input, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>vxuEqualizeHist(vx_context context, vx_image input, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>3.24</td>
<td>Erode Image</td>
</tr>
<tr>
<td>3.24.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.24.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>vxErode3x3Node(vx_graph graph, vx_image input, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>vxuErode3x3(vx_context context, vx_image input, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>3.25</td>
<td>Fast Corners</td>
</tr>
<tr>
<td>3.25.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.25.2</td>
<td>Segment Test Detector</td>
</tr>
<tr>
<td>3.25.3</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>vxFastCornersNode(vx_graph graph, vx_image input, vx_scalar strength_thresh, vx_bool nonmax_suppression, vx_array corners, vx_scalar num_corners)</td>
<td></td>
</tr>
<tr>
<td>vxuFastCorners(vx_context context, vx_image input, vx_scalar strength_thresh, vx_bool nonmax_suppression, vx_array corners, vx_scalar num_corners)</td>
<td></td>
</tr>
<tr>
<td>3.26</td>
<td>Gaussian Filter</td>
</tr>
<tr>
<td>3.26.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.26.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>vxGaussian3x3Node(vx_graph graph, vx_image input, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>vxuGaussian3x3(vx_context context, vx_image input, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>3.27</td>
<td>Non Linear Filter</td>
</tr>
<tr>
<td>3.27.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.27.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>vxNonLinearFilterNode(vx_graph graph, vx_enum function, vx_image input, vx_matrix mask, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>vxuNonLinearFilter(vx_context context, vx_enum function, vx_image input, vx_matrix mask, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>3.28</td>
<td>Harris Corners</td>
</tr>
<tr>
<td>3.28.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.28.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>vxHarrisCornersNode(vx_graph graph, vx_image input, vx_scalar k1, vx_scalar k2, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>vxuHarrisCorners(vx_context context, vx_image input, vx_scalar k1, vx_scalar k2, vx_image output)</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>3.28.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.28.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td></td>
<td>vxHarrisCornersNode(vx_graph graph, vx_image input, vx_scalar strength_thresh, vx_scalar min_distance, vx_scalar sensitivity, vx_int32 gradient_size, vx_int32 block_size, vx_array corners, vx_scalar num_corners)</td>
</tr>
<tr>
<td></td>
<td>vxuHarrisCorners(vx_context context, vx_image input, vx_scalar strength_thresh, vx_scalar min_distance, vx_scalar sensitivity, vx_int32 gradient_size, vx_int32 block_size, vx_array corners, vx_scalar num_corners)</td>
</tr>
<tr>
<td>3.29</td>
<td>Histogram</td>
</tr>
<tr>
<td>3.29.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.29.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td></td>
<td>vxHistogramNode(vx_graph graph, vx_image input, vx_distribution distribution)</td>
</tr>
<tr>
<td></td>
<td>vxuHistogram(vx_context context, vx_image input, vx_distribution distribution)</td>
</tr>
<tr>
<td>3.30</td>
<td>Gaussian Image Pyramid</td>
</tr>
<tr>
<td>3.30.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.30.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td></td>
<td>vxGaussianPyramidNode(vx_graph graph, vx_image input, vx_pyramid gaussian)</td>
</tr>
<tr>
<td></td>
<td>vxuGaussianPyramid(vx_context context, vx_image input, vx_pyramid gaussian)</td>
</tr>
<tr>
<td>3.31</td>
<td>Laplacian Image Pyramid</td>
</tr>
<tr>
<td>3.31.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.31.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td></td>
<td>vxLaplacianPyramidNode(vx_graph graph, vx_image input, vx_pyramid laplacian, vx_image output)</td>
</tr>
<tr>
<td></td>
<td>vxuLaplacianPyramid(vx_context context, vx_image input, vx_pyramid laplacian, vx_image output)</td>
</tr>
<tr>
<td>3.32</td>
<td>Reconstruction from a Laplacian Image Pyramid</td>
</tr>
<tr>
<td>3.32.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.32.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td></td>
<td>vxLaplacianReconstructNode(vx_graph graph, vx_pyramid laplacian, vx_image input, vx_image output)</td>
</tr>
<tr>
<td></td>
<td>vxuLaplacianReconstruct(vx_context context, vx_pyramid laplacian, vx_image input, vx_image output)</td>
</tr>
<tr>
<td>3.33</td>
<td>Integral Image</td>
</tr>
<tr>
<td>3.33.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.33.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td></td>
<td>vxIntegralImageNode(vx_graph graph, vx_image input, vx_image output)</td>
</tr>
<tr>
<td></td>
<td>vxuIntegralImage(vx_context context, vx_image input, vx_image output)</td>
</tr>
<tr>
<td>3.34</td>
<td>Magnitude</td>
</tr>
<tr>
<td>3.34.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.34.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td></td>
<td>vxMagnitudeNode(vx_graph graph, vx_image grad_x, vx_image grad_y, vx_image mag)</td>
</tr>
<tr>
<td></td>
<td>vxuMagnitude(vx_context context, vx_image grad_x, vx_image grad_y, vx_image mag)</td>
</tr>
<tr>
<td>3.35</td>
<td>Mean and Standard Deviation</td>
</tr>
<tr>
<td>3.35.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.35.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td></td>
<td>vxMeanStdDevNode(vx_graph graph, vx_image input, vx_scalar mean, vx_scalar stddev)</td>
</tr>
<tr>
<td></td>
<td>vxuMeanStdDev(vx_context context, vx_image input, vx_float32 mean, vx_float32 stddev)</td>
</tr>
<tr>
<td>3.36</td>
<td>Median Filter</td>
</tr>
<tr>
<td>3.36.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.36.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td></td>
<td>vxMedian3x3Node(vx_graph graph, vx_image input, vx_image output)</td>
</tr>
<tr>
<td></td>
<td>vxuMedian3x3(vx_context context, vx_image input, vx_image output)</td>
</tr>
<tr>
<td>3.37</td>
<td>Min, Max Location</td>
</tr>
<tr>
<td>3.37.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>3.37.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td></td>
<td>vxMinMaxLocNode(vx_graph graph, vx_image input, vx_scalar minVal, vx_scalar maxVal, vx_array minLoc, vx_array maxLoc, vx_scalar minCount, vx_scalar maxCount)</td>
</tr>
<tr>
<td>TableLookup</td>
<td>131</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td>Optical Flow Pyramid (LK)</td>
<td>116</td>
</tr>
<tr>
<td>Max</td>
<td>114</td>
</tr>
<tr>
<td>Sobel 3x3</td>
<td>129</td>
</tr>
<tr>
<td>Scale Image</td>
<td>128</td>
</tr>
<tr>
<td>Remap</td>
<td>123</td>
</tr>
<tr>
<td>Phase</td>
<td>119</td>
</tr>
<tr>
<td>Pixel-wise Multiplication</td>
<td>121</td>
</tr>
<tr>
<td>Min</td>
<td>112</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>112</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>114</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>117</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>119</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>119</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>121</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>121</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>123</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>123</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>125</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>127</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>127</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>128</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>129</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>129</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>129</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>131</td>
</tr>
<tr>
<td>Function Documentation</td>
<td>131</td>
</tr>
</tbody>
</table>

Code Examples:

- `vxuSobel3x3(vx_context context, vx_image input, vx_image output_x, vx_image output_y)`
- `vxSobel3x3Node(vx_graph graph, vx_image input, vx_image output_x, vx_image output_y)`
- `vxuHalfScaleGaussian(vx_context context, vx_image input, vx_image output, vx_int32 kernel_size)`
- `vxHalfScaleGaussianNode(vx_graph graph, vx_image input, vx_image output, vx_int32 kernel_size)`
- `vxuMinMaxLoc(vx_context context, vx_image input, vx_scalar minVal, vx_scalar maxVal, vx_array minLoc, vx_array maxLoc, vx_scalar minCount, vx_scalar maxCount)`
- `vxuOpticalFlowPyrLK(vx_context context, vx_pyramid old_images, vx_pyramid new_images, vx_array old_points, vx_array new_points_estimates, vx_enum termination, vx_scalar epsilon, vx_scalar num_iterations, vx_scalar use_initial_estimate, vx_size window_dimension)`
- `vxuOpticalFlowPyrLKNode(vx_graph graph, vx_pyramid old_images, vx_pyramid new_images, vx_array old_points, vx_array new_points_estimates, vx_enum termination, vx_scalar epsilon, vx_scalar num_iterations, vx_scalar use_initial_estimate, vx_size window_dimension)`
- `vxMin(vx_context context, vx_image input, vx_image output, vx_scalar minVal, vx_scalar maxVal, vx_array minLoc, vx_array maxLoc, vx_scalar minCount, vx_scalar maxCount)`
- `vxuMin(vx_context context, vx_image input, vx_image output, vx_scalar minVal, vx_scalar maxVal, vx_array minLoc, vx_array maxLoc, vx_scalar minCount, vx_scalar maxCount)`
- `vxMinNode(vx_graph graph, vx_image input, vx_image output)`
- `vxuMin(vx_context context, vx_image input, vx_image output)`
- `vxMax(vx_context context, vx_image input, vx_image output)`
- `vxuMax(vx_context context, vx_image input, vx_image output)`
- `vxMaxNode(vx_graph graph, vx_image input, vx_image output)`
- `vxuMax(vx_context context, vx_image input, vx_image output)`
- `vxuHalfScaleGaussian(vx_context context, vx_image input, vx_image output, vx_int32 kernel_size)`
- `vxHalfScaleGaussianNode(vx_graph graph, vx_image input, vx_image output, vx_int32 kernel_size)`
CONTENTS

3.46.2 Function Documentation ................................................................. 131
vxTableLookupNode(vx_graph graph, vx_image input, vx_lut lut, vx_image output) 131
vxuTableLookup(vx_context context, vx_image input, vx_lut lut, vx_image output) 131

3.47 Thresholding .................................................................................... 133
3.47.1 Detailed Description ....................................................................... 133
vxThresholdNode(vx_graph graph, vx_image input, vx_threshold thresh, vx_image output) 133
vxuThreshold(vx_context context, vx_image input, vx_threshold thresh, vx_image output) 134

3.48 Warp Affine ....................................................................................... 135
3.48.1 Detailed Description ....................................................................... 135
vxWarpAffineNode(vx_graph graph, vx_image input, vx_matrix matrix, vx_enum type, vx__image output) 135
vxuWarpAffine(vx_context context, vx_image input, vx_matrix matrix, vx_enum type, vx__image output) 136

3.49 Warp Perspective ............................................................................... 137
3.49.1 Detailed Description ....................................................................... 137
vxWarpPerspectiveNode(vx_graph graph, vx_image input, vx_matrix matrix, vx_enum type, vx__image output) 137
vxuWarpPerspective(vx_context context, vx_image input, vx_matrix matrix, vx_enum type, vx__image output) 138

3.50 Bilateral Filter ................................................................................... 139
3.50.1 Detailed Description ....................................................................... 139
vxBilateralFilterNode(vx_graph graph, vx_tensor src, vx_int32 diameter, vx_float32 sigma--
Space, vx_float32 sigmaValues, vx_tensor dst) ........................................... 139
vxuBilateralFilter(vx_context context, vx_tensor src, vx_int32 diameter, vx_float32 sigma--
Space, vx_float32 sigmaValues, vx_tensor dst) ........................................... 140

3.51 MatchTemplate .................................................................................. 141
3.51.1 Detailed Description ....................................................................... 141
vx_comp_metric_e .................................................................................... 141
vxMatchTemplateNode(vx_graph graph, vx_image src, vx_image templateImage, vx_enum
matchingMethod, vx_image output) .......................................................... 142
vxuMatchTemplate(vx_context context, vx_image src, vx_image templateImage, vx_enum
matchingMethod, vx_image output) .......................................................... 142

3.52 LBP ................................................................................................ 143
3.52.1 Detailed Description ....................................................................... 143
vx_lbp_format_e ....................................................................................... 144
vxLBPNode(vx_graph graph, vx_image in, vx_enum format, vx_int8 kernel_size, vx_image
out) .......................................................................................................... 145
vxuLBP(vx_context context, vx_image in, vx_enum format, vx_int8 kernel_size, vx_image out) 145

3.53 HOG ................................................................................................ 146
3.53.1 Detailed Description ....................................................................... 146
struct vx_hog_t ......................................................................................... 146
vxHOGCellsNode(vx_graph graph, vx_image input, vx_int32 cell_width, vx_int32 cell__
height, vx_int32 num_bins, vx_tensor magnitudes, vx_tensor bins) ............. 147
vxHOGFeaturesNode(vx_graph graph, vx_image input, vx_tensor magnitudes, vx_tensor
bins, const vx_hog_t *params, vx_size hog_param_size, vx_tensor features) .... 148
vxuHOGCells(vx_context context, vx_image input, vx_int32 cell_width, vx_int32 cell__height,
vx_int32 num_bins, vx_tensor magnitudes, vx_tensor bins) ....................... 149
3.54 HoughLinesP ................................................................. 152
  3.54.1 Detailed Description ............................................... 152
  3.54.2 Data Structure Documentation ..................................... 152
  3.54.3 Function Documentation ........................................... 153
    vxHoughLinesPNode(vx_graph graph, vx_image input, const vx_hough_lines_p_t ...
    vxuHoughLinesP(vx_context context, vx_image input, const vx_hough_lines_p_t ...

3.55 Tensor Multiply .......................................................... 155
  3.55.1 Detailed Description ............................................... 155
  3.55.2 Function Documentation ........................................... 155
    vxTensorMultiplyNode(vx_graph graph, vx_tensor input1, vx_tensor input2, vx_scalar ...
    vxuTensorMultiply(vx_context context, vx_tensor input1, vx_tensor input2, vx_scalar ...

3.56 Tensor Add ............................................................... 157
  3.56.1 Detailed Description ............................................... 157
  3.56.2 Function Documentation ........................................... 157
    vxTensorAddNode(vx_graph graph, vx_tensor input1, vx_tensor input2, vx_enum policy, ...
    vxuTensorAdd(vx_context context, vx_tensor input1, vx_tensor input2, vx_enum policy, ...

3.57 Tensor Subtract ........................................................ 159
  3.57.1 Detailed Description ............................................... 159
  3.57.2 Function Documentation ........................................... 159
    vxTensorSubtractNode(vx_graph graph, vx_tensor input1, vx_tensor input2, vx_enum policy, ...
    vxuTensorSubtract(vx_context context, vx_tensor input1, vx_tensor input2, vx_enum policy, ...

3.58 Tensor TableLookUp .................................................... 161
  3.58.1 Detailed Description ............................................... 161
  3.58.2 Function Documentation ........................................... 161
    vxTensorTableLookupNode(vx_graph graph, vx_tensor input1, vx_lut lut, vx_tensor output) ...
    vxuTensorTableLookup(vx_context context, vx_tensor input1, vx_lut lut, vx_tensor output) ...

3.59 Tensor Transpose ...................................................... 163
  3.59.1 Detailed Description ............................................... 163
  3.59.2 Function Documentation ........................................... 163
    vxTensorTransposeNode(vx_graph graph, vx_tensor input, vx_tensor output, vx_size dimension1, ...
    vxuTensorTranspose(vx_context context, vx_tensor input, vx_tensor output, vx_size dimension1, ...

3.60 Tensor Convert Bit-Depth .............................................. 165
  3.60.1 Detailed Description ............................................... 165
  3.60.2 Function Documentation ........................................... 165
    vxTensorConvertDepthNode(vx_graph graph, vx_tensor input, vx_enum policy, vx_scalar norm, ...
    vxuTensorConvertDepth(vx_context context, vx_tensor input, vx_enum policy, vx_scalar norm, ...

3.61 Tensor Matrix Multiply .............................................. 167
  3.61.1 Detailed Description ............................................... 167
  3.61.2 Data Structure Documentation ..................................... 167
    struct vx_tensor_matrix_multiply_params_t
    struct vx_tensor_matrix_multiply_node ...
  3.61.3 Function Documentation ........................................... 168
vxTensorMatrixMultiplyNode(vx_graph graph, vx_tensor input1, vx_tensor input2, vx_tensor
input3, const vx_tensor_matrix_multiply_params_t *matrix_multiply_params,
vx_tensor output) .......................................................... 168
vxuTensorMatrixMultiply(vx_context context, vx_tensor input1, vx_tensor input2, vx_tensor
input3, const vx_tensor_matrix_multiply_params_t *matrix_multiply_params,
vx_tensor output) .......................................................... 168
3.62 Basic Features .......................................................... 169
3.62.1 Detailed Description ................................................. 169
3.62.2 Data Structure Documentation ....................................... 175
struct vx_coordinates2d_t .................................................. 175
struct vx_coordinates2df_t .................................................. 175
struct vx_coordinates3d_t .................................................. 176
struct vx_keypoint_t ....................................................... 176
struct vx_line2d_t ........................................................... 176
struct vx_rectangle_t ...................................................... 176
3.62.3 Macro Definition Documentation .................................... 177
VX_TYPE_MASK ............................................................... 177
VX_DF_IMAGE ............................................................... 177
VX_ENUM_BASE ............................................................... 177
3.62.4 Typedef Documentation ............................................... 177
vx_enum ................................................................. 177
vx_bool ................................................................. 177
vx_status .............................................................. 178
3.62.5 Enumeration Type Documentation ................................... 178
vx_bool_e ................................................................. 178
vx_type_e ................................................................. 178
vx_status_e ............................................................... 180
vx_enum_e ................................................................. 181
vx_convert_policy_e ...................................................... 182
vx_df_image_e ............................................................. 182
vx_target_e ............................................................... 182
vx_channel_e .............................................................. 183
vx_interpolation_type_e .................................................. 183
vx_non_linear_filter_e ................................................... 184
vx_pattern_e ............................................................. 184
vx_vendor_id_e ........................................................... 184
3.62.6 Function Documentation .............................................. 185
vxGetStatus(vx_reference reference) ..................................... 185
3.63 Objects .................................................................. 186
3.63.1 Detailed Description .................................................. 186
3.64 Object: Reference ......................................................... 187
3.64.1 Detailed Description .................................................. 187
3.64.2 Macro Definition Documentation .................................... 187
VX_MAX_REFERENCE_NAME ................................................ 187
3.64.3 Typedef Documentation ............................................... 188
vx_reference ............................................................... 188
3.64.4 Enumeration Type Documentation ................................... 188
vx_reference_attribute_e .................................................. 188
3.64.5 Function Documentation .............................................. 188
vxQueryReference(vx_reference ref, vx_enum attribute, void *ptr, vx_size size) ............ 188
vxReleaseReference(vx_reference *ref_ptr) .................................. 188
vxRetainReference(vx_reference ref) ....................................... 189
vxSetReferenceName(vx_reference ref, const vx_char *name) ......................... 189
3.65 Object: Context ............................................................. 191
3.65.1 Detailed Description .................................................. 191
3.65.2 Typedef Documentation ............................................... 192
vx_context ............................................................... 192
3.65.3 Enumeration Type Documentation ................................... 192
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
</table>
| 3.72 | Object: LUT
| 3.72.1 | Detailed Description
| 3.72.2 | Enumeration Type Documentation
| 3.72.3 | Function Documentation
| 3.73 | Object: Matrix
| 3.73.1 | Detailed Description
| 3.73.2 | Enumeration Type Documentation
| 3.73.3 | Function Documentation
| 3.74 | Object: Pyramid
| 3.74.1 | Detailed Description
| 3.74.2 | Enumeration Type Documentation
| 3.74.3 | Function Documentation
| 3.75 | Object: Remap
| 3.75.1 | Detailed Description
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.75.2</td>
<td>Enumeration Type Documentation</td>
<td>259</td>
</tr>
<tr>
<td></td>
<td>vx_remap_attribute_e</td>
<td>259</td>
</tr>
<tr>
<td>3.75.3</td>
<td>Function Documentation</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td>vxCreateRemap(vx_context context, vx_uint32 src_width, vx_uint32 dst_width, vx_uint32 dst_height)</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td>vxCreateVirtualRemap(vx_graph graph, vx_uint32 src_width, vx_uint32 src_height, vx_void_ptr src, vx_size src_size)</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td>vxReleaseRemap(vx_remap remap)</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td>vxMapRemapPatch(vx_remap remap, const vx_rectangle_t *rect, vx_map_id map_id, vx_void_ptr remap_table, vx_size table_size)</td>
<td>261</td>
</tr>
<tr>
<td></td>
<td>vxUnmapRemapPatch(vx_remap remap, vx_map_id map_id)</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td>vxCopyRemapPatch(vx_remap remap, const vx_rectangle_t *rect, vx_size user_stride_y, vx_size user_stride_x, vx_size user_stride_z, const vx_pixel_value_t *true_value_ptr, const vx_pixel_value_t *false_value_ptr, const vx_pixel_value_t *lower_value_ptr, const vx_pixel_value_t *upper_value_ptr, vx_pixel_value_t *true_value, vx_pixel_value_t *false_value, vx_pixel_value_t *lower_value, vx_pixel_value_t *upper_value)</td>
<td>263</td>
</tr>
<tr>
<td></td>
<td>vxQueryRemap(vx_remap remap, void *ptr, vx_size size)</td>
<td>264</td>
</tr>
<tr>
<td>3.76</td>
<td>Object: Scalar</td>
<td>265</td>
</tr>
<tr>
<td>3.76.1</td>
<td>Detailed Description</td>
<td>265</td>
</tr>
<tr>
<td>3.76.2</td>
<td>Typedef Documentation</td>
<td>266</td>
</tr>
<tr>
<td></td>
<td>vx_scalar</td>
<td>266</td>
</tr>
<tr>
<td>3.76.3</td>
<td>Enumeration Type Documentation</td>
<td>266</td>
</tr>
<tr>
<td></td>
<td>vx_scalar_attribute_e</td>
<td>266</td>
</tr>
<tr>
<td></td>
<td>vx_scalar_operation_e</td>
<td>266</td>
</tr>
<tr>
<td>3.76.4</td>
<td>Function Documentation</td>
<td>267</td>
</tr>
<tr>
<td></td>
<td>vxCreateScalar(vx_context context, vx_enum data_type, const void *ptr)</td>
<td>267</td>
</tr>
<tr>
<td></td>
<td>vxCreateScalarWithSize(vx_context context, vx_enum data_type, const void *ptr, vx_size size)</td>
<td>267</td>
</tr>
<tr>
<td></td>
<td>vxCreateVirtualScalar(vx_graph graph, vx_enum data_type)</td>
<td>267</td>
</tr>
<tr>
<td></td>
<td>vxReleaseScalar(vx_scalar +scalar)</td>
<td>268</td>
</tr>
<tr>
<td></td>
<td>vxQueryScalar(vx_scalar scalar, vx_enum attribute, void *ptr, vx_size size)</td>
<td>268</td>
</tr>
<tr>
<td></td>
<td>vxCopyScalar(vx_scalar scalar, void *user_ptr, vx_enum user_mem_type, vx_size user_size)</td>
<td>269</td>
</tr>
<tr>
<td></td>
<td>vxCopyScalarWithSize(vx_scalar scalar, vx_size user_size, void *user_ptr, vx_enum user_mem_type, vx_size user_size)</td>
<td>269</td>
</tr>
<tr>
<td>3.77</td>
<td>Object: Threshold</td>
<td>271</td>
</tr>
<tr>
<td>3.77.1</td>
<td>Detailed Description</td>
<td>271</td>
</tr>
<tr>
<td>3.77.2</td>
<td>Enumeration Type Documentation</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td>vx_threshold_type_e</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td>vx_threshold_attribute_e</td>
<td>272</td>
</tr>
<tr>
<td>3.77.3</td>
<td>Function Documentation</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td>vxCreateThresholdForImage(vx_context context, vx_enum thresh_type, vx_df_image input_format, vx_df_image output_format)</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td>vxCreateVirtualThresholdForImage(vx_graph graph, vx_enum thresh_type, vx_df_image input_format, vx_df_image output_format)</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td>vxCopyThresholdValue(vx_threshold thresh, vx_pixel_value_t *value_ptr, vx_enum usage, vx_enum user_mem_type)</td>
<td>273</td>
</tr>
<tr>
<td></td>
<td>vxCopyThresholdRange(vx_threshold thresh, vx_pixel_value_t *lower_value_ptr, vx_pixel_value_t *upper_value_ptr, vx_enum usage, vx_enum user_mem_type)</td>
<td>274</td>
</tr>
<tr>
<td></td>
<td>vxCopyThresholdOutput(vx_threshold thresh, vx_pixel_value_t *true_value_ptr, vx_pixel_value_t *false_value_ptr, vx_pixel_value_t *lower_value_ptr, vx_pixel_value_t *upper_value_ptr, vx_enum usage, vx_enum user_mem_type)</td>
<td>275</td>
</tr>
<tr>
<td></td>
<td>vxReleaseThreshold(vx_threshold +thresh)</td>
<td>276</td>
</tr>
<tr>
<td></td>
<td>vxSetThresholdAttribute(vx_threshold thresh, vx_enum attribute, const void *ptr, vx_size size)</td>
<td>276</td>
</tr>
<tr>
<td></td>
<td>vxQueryThreshold(vx_threshold thresh, vx_enum attribute, void *ptr, vx_size size)</td>
<td>277</td>
</tr>
<tr>
<td>3.78</td>
<td>Object: ObjectArray</td>
<td>278</td>
</tr>
<tr>
<td>3.78.1</td>
<td>Detailed Description</td>
<td>278</td>
</tr>
<tr>
<td>3.78.2</td>
<td>Enumeration Type Documentation</td>
<td>278</td>
</tr>
<tr>
<td></td>
<td>vx_object_array_attribute_e</td>
<td>278</td>
</tr>
<tr>
<td>3.78.3</td>
<td>Function Documentation</td>
<td>279</td>
</tr>
<tr>
<td></td>
<td>vxCreateObjectArray(vx_context context, vx_reference exemplar, vx_size count)</td>
<td>279</td>
</tr>
</tbody>
</table>
3.93.3 Function Documentation
vxDirective(vx_reference reference, vx_enum directive) ........................................... 324
3.94 Framework: User Kernels ................................................................. 326
3.94.1 Detailed Description ........................................................................ 326
3.94.2 Typedef Documentation
vx_meta_format ................................................................. 330
vx_publish_kernels_f ................................................................. 330
vx_unpublish_kernels_f ................................................................. 330
vx_kernel_f ........................................................................ 330
vx_kernel_initialize_f ................................................................. 331
vx_kernel_deinitialize_f ................................................................. 331
vx_kernel_validate_f ................................................................. 331
vx_kernel_image_valid_rectangle_f ................................................................. 332
3.94.3 Enumeration Type Documentation
vx_meta_valid_rect_attribute_e ................................................................. 332
3.94.4 Function Documentation
vxAllocateUserKernelId(vx_context context, vx_enum *pKernelEnumId) ......................... 332
vxAllocateUserKernelLibraryId(vx_context context, vx_enum *pLibraryId) ......................... 333
vxLoadKernels(vx_context context, const vx_char *module) ........................................... 333
vxUnloadKernels(vx_context context, const vx_char *module) .......................................... 334
vxAddUserKernel(vx_context context, const vx_char name[VX_MAX_KERNEL_NAME],
vx_enum enumeration, vx_kernel_f func_ptr, vx_uint32 numParams, vx_kernel_validate_f validate,
vx_kernel_initialize_f init, vx_kernel_deinitialize_f deinit) ........................................ 335
vxFinalizeKernel(vx_kernel kernel) ........................................................................ 335
vxAddParameterToKernel(vx_kernel kernel, vx_uint32 index, vx_enum dir, vx_enum data_type,
vx_enum state) ........................................................................ 335
vxRemoveKernel(vx_kernel kernel) ........................................................................ 336
vxSetKernelAttribute(vx_kernel kernel, vx_enum attribute, const void *ptr, vx_size size) . 336
vxSetMetaFormatAttribute(vx_meta_format meta, vx_enum attribute, const void *ptr, vx_size size) ........................................................................ 337
vxSetMetaFormatFromReference(vx_meta_format meta, vx_reference exemplar) .................. 338
3.95 Framework: Graph Parameters ........................................................... 339
3.95.1 Detailed Description ........................................................................ 339
3.95.2 Function Documentation
vxAddParameterToGraph(vx_graph graph, vx_parameter parameter) ................................. 340
vxSetGraphParameterByIndex(vx_graph graph, vx_uint32 index, vx_reference value) ......... 341
vxGetGraphParameterByIndex(vx_graph graph, vx_uint32 index) ........................................ 341
Chapter 1

Introduction

1.1 Abstract

OpenVX is a low-level programming framework domain to enable software developers to efficiently access computer vision hardware acceleration with both functional and performance portability. OpenVX has been designed to support modern hardware architectures, such as mobile and embedded SoCs as well as desktop systems. Many of these systems are parallel and heterogeneous: containing multiple processor types including multi-core CPUs, DSP subsystems, GPUs, dedicated vision computing fabrics as well as hardwired functionality. Additionally, vision system memory hierarchies can often be complex, distributed, and not fully coherent. OpenVX is designed to maximize functional and performance portability across these diverse hardware platforms, providing a computer vision framework that efficiently addresses current and future hardware architectures with minimal impact on applications.

OpenVX contains:

- a library of predefined and customizable vision functions,
- a graph-based execution model to combine function enabling both task and data-independent execution, and;
- a set of memory objects that abstract the physical memory.

OpenVX defines a C Application Programming Interface (API) for building, verifying, and coordinating graph execution, as well as for accessing memory objects. The graph abstraction enables OpenVX implementers to optimize the execution of the graph for the underlying acceleration architecture.

OpenVX also defines the vxu utility library, which exposes each OpenVX predefined function as a directly callable C function, without the need for first creating a graph. Applications built using the vxu library do not benefit from the optimizations enabled by graphs; however, the vxu library can be useful as the simplest way to use OpenVX and as first step in porting existing vision applications.

As the computer vision domain is still rapidly evolving, OpenVX provides an extensibility mechanism to enable developer-defined functions to be added to the application graph.

1.2 Purpose

The purpose of this document is to detail the Application Programming Interface (API) for OpenVX.

1.3 Scope of Specification

The document contains the definition of the OpenVX API. The conformance tests that are used to determine whether an implementation is consistent to this specification are defined separately.

1.4 Normative References

The section “Module Documentation” forms the normative part of the specification. Each API definition provided in that chapter has certain preconditions and post conditions specified that are normative. If these normative conditions are not met, the behavior of the function is undefined.
1.5 Version/Change History

- OpenVX 1.0 Provisional - November, 2013
- OpenVX 1.0 Provisional V2 - June, 2014
- OpenVX 1.0 - September 2014
- OpenVX 1.0.1 - April 2015
- OpenVX 1.1 - May 2016
- OpenVX 1.2 - May 2017

1.6 Deprecation

Certain items that are deprecated through the evolution of this specification document are removed from it. However, to provide a backward compatibility for such items for a certain time period these items are made available via a compatibility header file available with the release of this specification document (vx_compatibility.h). The items listed in this compatibility header file are temporary only and are removed permanently when the backward compatibility is no longer supported for those items.

1.7 Requirements Language

In this specification, the words *shall* or *must* express a requirement that is binding, *should* expresses design goals or recommended actions, and *may* expresses an allowed behavior.

1.8 Typographical Conventions

The following typographical conventions are used in this specification.

- **Bold** words indicate warnings or strongly communicated concepts that are intended to draw attention to the text.
- **Monospace** words signify an API element (i.e., class, function, structure) or a filename.
- **Italics** denote an emphasis on a particular concept, an abstraction of a concept, or signify an argument, parameter, or member.

- Throughout this specification, code examples given to highlight a particular issue use the format as shown below:
  ```c
  /* Example Code Section */
  int main(int argc, char *argv[])
  {
    return 0;
  }
  ```

- Some “mscgen” message diagrams are included in this specification. The graphical conventions for this tool can be found on its website.

  See also [http://www.mcternan.me.uk/mscgen/](http://www.mcternan.me.uk/mscgen/)

1.8.1 Naming Conventions

The following naming conventions are used in this specification.

- Opaque objects and atomics are named as *vx_object*, e.g., *vx_image* or *vx_uint8*, with an underscore separating the object name from the “vx” prefix.

- Defined Structures are named as *vx_struct_t*, e.g., *vx_imagepatch_addressing_t*, with underscores separating the structure from the “vx” prefix and a “t” to denote that it is a structure.
• Defined Enumerations are named as `vx_enum_e`, e.g., `vx_type_e`, with underscores separating the enumeration from the “vx” prefix and an “e” to denote that it is an enumerated value.

• Application Programming Interfaces are named `vxSomeFunction()` using camel case, starting with lowercase, and no underscores, e.g., `vxCreateContext()`.

• Vision functions also have a naming convention that follows a lower-case, inverse dotted hierarchy similar to Java Packages, e.g.,

  "org.khronos.openvx.color_convert".

This minimizes the possibility of name collisions and promotes sorting and readability when querying the namespace of available vision functions. Each vision function should have a unique dotted name of the style: `tld.vendor.library.function`. The hierarchy of such vision function namespaces is undefined outside the subdomain “org.khronos”, but they do follow existing international standards. For OpenVX-specified vision functions, the “function” section of the unique name does not use camel case and uses underscores to separate words.

1.8.2 Vendor Naming Conventions

The following naming conventions are to be used for vendor specific extensions.

• Opaque objects and atomics are named as `vx_object_vendor`, e.g., `vx_ref_array_acme`, with an underscore separating the vendor name from the object name.

• Defined Structures are named as `vx_struct_vendor_t`, e.g., `vx_mdview_acme_t`, with an underscore separating the vendor from the structure name and a “t” to denote that it is a structure.

• Defined Enumerations are named as `vx_enum_vendor_e`, e.g., `vx_convolution_name_acme_e`, with an underscores separating the vendor from the enumeration name and an “e” to denote that it is an enumerated value.

• Defined Enumeration values are named as `VX_ENUMVALUE_VENDOR`, e.g., `VX_PARAM_STRUCT_ATTRIBUTE_SIZE_ACME` using only capital letters staring with the “VX” prefix, and underscores separating the words.

• Application Programming Interfaces are named `vxSomeFunctionVendor()` using camel case, starting with lowercase, and no underscores, e.g., `vxCreateRefArrayAcme()`.

1.9 Glossary and Acronyms

• Atomic: The specification mentions atomics, which means a C primitive data type. Usages that have additional wording, such as atomic operations do not carry this meaning.

• API: Application Programming Interface that specifies how a software component interacts with another.

• Framework: A generic software abstraction in which users can override behaviors to produce application-specific functionality.

• Engine: A purpose-specific software abstraction that is tunable by users.

• Run-time: The execution phase of a program.

• Kernel: OpenVX uses the term kernel to mean an abstract computer vision function, not an Operating System kernel. Kernel may also refer to a set of convolution coefficients in some computer vision literature (e.g., the Sobel “kernel”). OpenVX does not use this meaning. OpenCL uses kernel (specifically `cl_kernel`) to qualify a function written in “CL” which the OpenCL may invoke directly. This is close to the meaning OpenVX uses; however, OpenVX does not define a language.
1.10 Acknowledgements

This specification would not be possible without the contributions from this partial list of the following individuals from the Khronos Working Group and the companies that they represented at the time:

- Erik Rainey - Amazon
- Radhakrishna Giduthuri - AMD
- Mikael Bourges-Sevenier - Aptina Imaging Corporation
- Dave Schreiner - ARM Limited
- Renato Grottesi - ARM Limited
- Hans-Peter Nilsson - Axis Communications
- Amit Shoham - BDTi
- Frank Brill - Cadence Design Systems
- Thierry Lepley - Cadence Design Systems
- Shorin Kyo - Huawei
- Paul Buxton - Imagination Technologies
- Steve Ramm - Imagination Technologies
- Ben Ashbaugh - Intel
- Mostafa Hagog - Intel
- Andrey Kamaev - Intel
- Yaniv klein - Intel
- Andy Kuzma - Intel
- Tomer Schwartz - Intel
- Alexander Alekhin - Itseez
- Roman Donchenko - Itseez
- Victor Erukhimov - Itseez
- Vadim Pisarevsky - Itseez
- Vlad Vinogradov - Itseez
- Cormac Brick - Movidius Ltd
- Anshu Arya - MulticoreWare
- Shervin Emami - NVIDIA
- Kari Pulli - NVIDIA
- Neil Trevett - NVIDIA
- Daniel Laroche - NXP Semiconductors
- Susheel Gautam - QUALCOMM
- Doug Knisely - QUALCOMM
- Tao Zhang - QUALCOMM
- Yuki Kobayashi - Renesas Electronics
• Andrew Garrard - Samsung Electronics
• Erez Natan - Samsung Electronics
• Tomer Yanir - Samsung Electronics
• Chang-Hyo Yu - Samsung Electronics
• Olivier Pothier - STMicroelectronics International NV
• Chris Tseng - Texas Instruments, Inc.
• Jesse Villareal - Texas Instruments, Inc.
• Jiechao Nie - Verisilicon.Inc.
• Shehrzad Qureshi - Verisilicon.Inc.
• Xin Wang - Verisilicon.Inc.
• Stephen Neuendorffer - Xilinx, Inc.
Chapter 2

Design Overview

2.1 Software Landscape

OpenVX is intended to be used either directly by applications or as the acceleration layer for higher-level vision frameworks, engines or platform APIs.

![Diagram showing OpenVX usage overview]

Figure 2.1: OpenVX Usage Overview

2.2 Design Objectives

OpenVX is designed as a framework of standardized computer vision functions able to run on a wide variety of platforms and potentially to be accelerated by a vendor’s implementation on that platform. OpenVX can improve the
performance and efficiency of vision applications by providing an abstraction for commonly-used vision functions and an abstraction for aggregations of functions (a “graph”), thereby providing the implementer the opportunity to minimize the run-time overhead.

The functions in OpenVX are intended to cover common functionality required by many vision applications.

2.2.1 Hardware Optimizations

This specification makes no statements as to which acceleration methodology or techniques may be used in its implementation. Vendors may choose any number of implementation methods such as parallelism and/or specialized hardware offload techniques.

This specification also makes no statement or requirements on a “level of performance” as this may vary significantly across platforms and use cases.

2.2.2 Hardware Limitations

The OpenVX focuses on vision functions that can be significantly accelerated by diverse hardware. Future versions of this specification may adopt additional vision functions into the core standard when hardware acceleration for those functions becomes practical.

2.3 Assumptions

2.3.1 Portability

OpenVX has been designed to maximize functional and performance portability wherever possible, while recognizing that the API is intended to be used on a wide diversity of devices with specific constraints and properties. Tradeoffs are made for portability where possible: for example, portable Graphs constructed using this API should work on any OpenVX implementation and return similar results within the precision bounds defined by the OpenVX conformance tests.

2.3.2 Opaqueness

OpenVX is intended to address a very broad range of devices and platforms, from deeply embedded systems to desktop machines and distributed computing architectures. The OpenVX API addresses this range of possible implementations without forcing hardware-specific requirements onto any particular implementation via the use of opaque objects for most program data.

All data, except client-facing structures, are opaque and hidden behind a reference that may be as thin or thick as an implementation needs. Each implementation provides the standardized interfaces for accessing data that takes care of specialized hardware, platform, or allocation requirements. Memory that is imported or shared from other APIs is not subsumed by OpenVX and is still maintained and accessible by the originator.

OpenVX does not dictate any requirements on memory allocation methods or the layout of opaque memory objects and it does not dictate byte packing or alignment for structures on architectures.

2.4 Object-Oriented Behaviors

OpenVX objects are both strongly typed at compile-time for safety critical applications and are strongly typed at run-time for dynamic applications. Each object has its typedef’d type and its associated enumerated value in the `vx_type_e` list. Any object may be down-cast to a `vx_reference` safely to be used in functions that require this, specifically `vxQueryReference`, which can be used to get the `vx_type_e` value using an `vx_enum`.

2.5 OpenVX Framework Objects

This specification defines the following OpenVX framework objects.

- **Object: Context** - The OpenVX context is the object domain for all OpenVX objects. All data objects live in the context as well as all framework objects. The OpenVX context keeps reference counts on all objects and must do garbage collection during its deconstruction to free lost references. While multiple clients may connect to
the OpenVX context, all data are private in that the references that refer to data objects are given only to the creating party. The results of calling an OpenVX function on data objects created in different contexts are undefined.

- **Object: Kernel** - A Kernel in OpenVX is the abstract representation of a computer vision function, such as a “Sobel Gradient” or “Lucas Kanade Feature Tracking”. A vision function may implement many similar or identical features from other functions, but it is still considered a single, unique kernel as long as it is named by the same string and enumeration and conforms to the results specified by OpenVX. Kernels are similar to function signatures in this regard.

- **Object: Parameter** - An abstract input, output, or bidirectional data object passed to a computer vision function. This object contains the signature of that parameter's usage from the kernel description. This information includes:
  - **Signature Index** - The numbered index of the parameter in the signature.
  - **Object Type** - e.g. VX_TYPE_IMAGE, or VX_TYPE_ARRAY, or some other object type from vx_type_e.
  - **Usage Model** - e.g. VX_INPUT, VX_OUTPUT, or VX_BIDIRECTIONAL.
  - **Presence State** - e.g. VX_PARAMETER_STATE_REQUIRED, or VX_PARAMETER_STATE_OPTIONAL.

- **Object: Node** - A node is an instance of a kernel that will be paired with a specific set of references (the parameters). Nodes are created from and associated with a single graph only. When a vx_parameter is extracted from a Node, an additional attribute can be accessed:
  - **Reference** - The vx_reference assigned to this parameter index from the Node creation function (e.g., vxSobel3x3Node).

- **Object: Graph** - A set of nodes connected in a directed (only goes one-way) acyclic (does not loop back) fashion. A Graph may have sets of Nodes that are unconnected to other sets of Nodes within the same Graph. See Graph Formalisms.

### 2.6 OpenVX Data Objects

Data objects are object that are processed by graphs in nodes.

- **Object: Array** - An opaque array object that could be an array of primitive data types or an array of structures.

- **Object: Convolution** - An opaque object that contains $M \times N$ matrix of vx_int16 values. Also contains a scaling factor for normalization. Used specifically with vxuConvolve and vxConvolveNode.

- **Object: Delay** - An opaque object that contains a manually controlled, temporally-delayed list of objects.

- **Object: Distribution** - An opaque object that contains a frequency distribution (e.g., a histogram).

- **Object: Image** - An opaque image object that may be some format in vx_df_image_e.

- **Object: LUT** - An opaque lookup table object used with vxTableLookupNode and vxuTableLookup.

- **Object: Matrix** - An opaque object that contains $M \times N$ matrix of some scalar values.

- **Object: Pyramid** - An opaque object that contains multiple levels of scaled vx_image objects.

- **Object: Remap** - An opaque object that contains the map of source points to destination points used to transform images.

- **Object: Scalar** - An opaque object that contains a single primitive data type.

- **Object: Threshold** - An opaque object that contains the thresholding configuration.

- **Object: ObjectArray** - An opaque array object that could be an array of any data-object (not data-type) of OpenVX except Delay and ObjectArray objects.

- **Object: Tensor** - An opaque multidimensional data object. Used in functions like vxHOGFeaturesNode, vxHOGCellsNode and the Neural Networks extension.
2.7 Error Objects

Error objects are specialized objects that may be returned from other object creator functions when serious platform issue occur (i.e., out of memory or out of handles). These can be checked at the time of creation of these objects, but checking also may be put-off until usage in other APIs or verification time, in which case, the implementation must return appropriate errors to indicate that an invalid object type was used.

```c
vx_object obj = vxCreateObject(context, ...);
vx_status status = vxGetStatus((vx_reference)obj);
if (status == VX_SUCCESS) {
    // object is good
}
```

2.8 Graphs Concepts

The **graph** is the central computation concept of OpenVX. The purpose of using graphs to express the Computer Vision problem is to allow for the possibility of any implementation to maximize its optimization potential because all the operations of the graph and its dependencies are known ahead of time, before the graph is processed.

Graphs are composed of one or more nodes that are added to the graph through node creation functions. Graphs in OpenVX must be created ahead of processing time and verified by the implementation, after which they can be processed as many times as needed.

2.8.1 Linking Nodes

Graph Nodes are linked together via data dependencies with no explicitly-stated ordering. The same reference may be linked to other nodes. Linking has a limitation, however, in that only one node in a graph may output to any specific data object reference. That is, only a single writer of an object may exist in a given graph. This prevents indeterminate ordering from data dependencies. All writers in a graph shall produce output data before any reader of that data accesses it.

2.8.2 Virtual Data Objects

Graphs in OpenVX depend on data objects to link together nodes. When clients of OpenVX know that they do not need access to these intermediate data objects, they may be created as virtual. Virtual data objects can be used in the same manner as non-virtual data objects to link nodes of a graph together; however, virtual data objects are different in the following respects.

- Inaccessible - No calls to an Map/Unmap or Copy APIs shall succeed given a reference to an object created through a virtual create function from a Graph external perspective. Calls to Map/Unmap or Copy APIs from within client-defined node that belongs to the same graph as the virtual object will succeed as they are Graph internal.

- Scoped - Virtual data objects are scoped within the Graph in which they are created; they cannot be shared outside their scope. The live range of the data content of a virtual data object is limited to a single graph execution. In other word, data content of a virtual object is undefined before graph execution and no data of a virtual object should be expected to be preserved across successive graph executions by the application.

- Intermediates - Virtual data objects should be used only for intermediate operations within Graphs, because they are fundamentally inaccessible to clients of the API.

- Dimensionless or Formatless - Virtual data objects may have dimensions and formats partially or fully undefined at creation time. For instance, a virtual image can be created with undefined or partially defined dimensions (0x0, Nx0 or 0xN where N is not null) and/or without defined format (VX_DF_IMAGE_VIRT). The undefined property of the virtual object at creation time is undefined with regard to the graph and mutable at graph verification time; it will be automatically adjusted at each graph verification, deduced from the node that outputs the virtual object. Dimensions and format properties that are well defined at virtual object creation time are immutable and can't be adjusted automatically at graph verification time.

- Attributes - Even if a given Virtual data object does not have its dimensionality or format completely defined, these attributes may still be queried. If queried before the object participates in a graph verification, the attribute value returned is what the user provided (e.g., "0" for the dimension). If queried after graph verification (or re-verification), the attribute value returned will be the value determined by the graph verification rules.
• The Dimensionless or Formatless aspect of virtual data is a commodity that allows creating graphs generic with regard to dimensions or format, but there are restrictions:

  1. Nodes may require the dimensions and/or the format to be defined for a virtual output object when it can't be deduced from its other parameters. For example, a Scale node requires well defined dimensions for the output image, while ColorConvert and ChannelCombine nodes require a well defined format for the output image.
  2. An image created from ROI must always be well defined (vx_rectangle_t parameter) and can't be created from a dimensionless virtual image.
  3. A ROI of a formatless virtual image shouldn't be a node output.
  4. A tensor created from View must always be well defined and can't be created from a dimensionless virtual tensor.
  5. A view of a formatless virtual tensor shouldn't be a node output.
  6. Levels of a dimensionless or formatless virtual pyramid shouldn't be a node output.

• Inheritance - A sub-object inherits from the virtual property of its parent. A sub-object also inherits from the Dimensionless or Formatless property of its parent with restrictions:

  1. it is adjusted automatically at graph verification when the parent properties are adjusted (the parent is the output of a node)
  2. it can't be adjusted at graph verification when the sub-object is itself the output of a node.

• Optimizations - Virtual data objects do not have to be created during Graph validation and execution and therefore may be of zero size.

These restrictions enable vendors the ability to optimize some aspects of the data object or its usage. Some vendors may not allocate such objects, some may create intermediate sub-objects of the object, and some may allocate the object on remote, inaccessible memories. OpenVX does not proscribe which optimization the vendor does, merely that it may happen.

2.8.3 Node Parameters

Parameters to node creation functions are defined as either atomic types, such as vx_int32, vx_enum, or as objects, such as vx_scalar, vx_image. The atomic variables of the Node creation functions shall be converted by the framework into vx_scalar references for use by the Nodes. A node parameter of type vx_scalar can be changed during the graph execution; whereas, a node parameter of an atomic type (vx_int32 etc.) require at least a graph revalidation if changed. All node parameter objects may be modified by retrieving the reference to the vx_parameter via vxGetParameterByIndex, and then passing that to vxQueryParameter to retrieve the reference to the object.

```c
vx_parameter param = vxGetParameterByIndex(node, p);
vx_reference ref;
vxQueryParameter(param, VX_PARAMETER_REF, &ref, sizeof(ref));
```

If the type of the parameter is unknown, it may be retrieved with the same function.

```c
vx_enum type;
vxQueryParameter(param, VX_PARAMETER_TYPE, &type, sizeof(type));
```

/* cast the ref to the correct vx_<type>. Atomics are now vx_scalar */

2.8.4 Graph Parameters

Parameters may exist on Graphs, as well. These parameters are defined by the author of the Graph and each Graph parameter is defined as a specific parameter from a Node within the Graph using vxAddParameterToGraph. Graph parameters communicate to the implementation that there are specific Node parameters that may be modified by the client between Graph executions. Additionally, they are parameters that the client may set without the reference to the Node but with the reference to the Graph using vxSetGraphParameterByIndex. This allows for the Graph authors to construct Graph Factories. How these factories work falls outside the scope of this document.

See also

Framework: Graph Parameters
2.8.5 Execution Model

Graphs must execute in both:

- **Synchronous blocking mode** (in that vxProcessGraph will block until the graph has completed), and in
- **Asynchronous single-issue-per-reference mode** (via vxScheduleGraph and vxWaitGraph).

Asynchronous Mode

In asynchronous mode, Graphs must be single-issue-per-reference. This means that given a constructed graph reference \( G \), it may be scheduled multiple times but only executes sequentially with respect to itself. Multiple graphs references given to the asynchronous graph interface do not have a defined behavior and may execute in parallel or in series based on the behavior or the vendor's implementation.

2.8.6 Graph Formalisms

To use graphs several rules must be put in place to allow deterministic execution of Graphs. The behavior of a processGraph( \( G \) ) call is determined by the structure of the Processing Graph \( G \). The Processing Graph is a bipartite graph consisting of a set of Nodes \( N_1 \ldots N_n \) and a set of data objects \( d_1 \ldots d_i \). Each edge \(( N_y, D_z )\) in the graph represents a data object \( D_z \) that is written by Node \( N_y \) and each edge \(( D_y, N_i )\) represents a data object \( D_y \) that is read by Node \( N_i \). Each edge \( e \) has a name Name( \( e \) ), which gives the parameter name of the node that references the corresponding data object. Each Node Parameter also has a type Type( node, name ) in \{INPUT, OUTPUT, INOUT\}. Some data objects are Virtual, and some data objects are Delay. Delay data objects are just collections of data objects with indexing (like an image list) and known linking points in a graph. A node may be classified as a head node, which has no backward dependency. Alternatively, a node may be a dependent node, which has a backward dependency to the head node. In addition, the Processing Graph has several restrictions:

1. **Output typing** - Every output edge \(( N_i, D_z )\) requires Type( \( N_i \), Name( \( N_i, D_z ) \) ) in \{OUTPUT, INOUT\}
2. **Input typing** - Every input edge \(( N_x, D_y )\) requires Type( \( N_x \), Name( \( D_y, N_x ) \) ) in \{INPUT\} or \{INOUT\}
3. **Single Writer** - Every data object is the target of at most one output edge.
4. **Broken Cycles** - Every cycle in \( G \) must contain at least input edge \(( D_x, N_z )\) where \( D_x \) is Delay.
5. **Virtual images must have a source** - If \( D_y \) is Virtual, then there is at least one output edge that writes \( D_y \) ( \( N_i, D_y )\)
6. **Bidirectional data objects shall not be virtual** - If Type( \( N_y, Name( N_i, D_y ) \) ) is INOUT implies \( D_y \) is non-Virtual.
7. **Delay data objects shall not be virtual** - If \( D_y \) is Delay then it shall not be Virtual.
8. **A uniform image cannot be output or bidirectional.**

The execution of each node in a graph consists of an atomic operation (sometimes referred to as firing) that consumes data representing each input data object, processes it, and produces data representing each output data object. A node may execute when all of its input edges are marked present. Before the graph executes, the following initial marking is used:

- All input edges \(( D_x, N_z )\) from non-Virtual objects \( D_x \) are marked (parameters must be set).
- All input edges \(( D_x, N_i )\) with an output edge \(( N_z, D_x )\) are unmarked.
- All input edges \(( D_y, N_z )\) where \( D_y \) is a Delay data object are marked.

Processing a node results in unmarking all the corresponding input edges and marking all its output edges; marking an output edge \(( N_i, D_z )\) where \( D_z \) is not a Delay results in marking all of the input edges \(( D_y, N_i )\). Following these rules, it is possible to statically schedule the nodes in a graph as follows: Construct a precedence graph \( P \), including all the nodes \( N_1 \ldots N_n \), and an edge \(( N_i, N_j )\) for every pair of edges \(( N_i, D_y )\) and \(( D_y, N_j )\) where \( D_y \) is not a Delay. Then unconditionally fire each node according to any topological sort of \( P \).

The following assertions should be verified:

- \( P \) is a Directed Acyclic Graph (DAG), implied by 4 and the way it is constructed.
• Every data object has a value when it is executed, implied by 5, 6, 7, and the marking.

• Execution is deterministic if the nodes are deterministic, implied by 3, 4, and the marking.

• Every node completes its execution exactly once.

The execution model described here just acts as a formalism. For example, independent processing is allowed across multiple depended and depending nodes and edges, provided that the result is invariant with the execution model described here.

Contained & Overlapping Data Objects

There are cases in which two different data objects referenced by an output parameter of node \( N_1 \) and input parameter of node \( N_2 \) in a graph induce a dependency between these two nodes: For example, a pyramid and its level images, image and the sub-images created from it by `vxCreateImageFromROI` or `vxCreateImageFromChannel`, or overlapping sub-images of the same image. Following figure show examples of this dependency. To simplify subsequent definitions and requirements a limitation is imposed that if a sub-image \( I' \) has been created from image \( I \) and sub-image \( I'' \) has been created from \( I' \), then \( I'' \) is still considered a sub-image of \( I \) and not of \( I' \). In these cases it is expected that although the two nodes reference two different data objects, any change to one data object might be reflected in the other one. Therefore it implies that \( N_1 \) comes before \( N_2 \) in the graph's topological order. To ensure that, following definitions are introduced.

![Pyramid Example](image-url)

![Image Example](image-url)

1. Containment Set - \( C(d) \), the set of recursively contained data objects of \( d \), named Containment Set, is defined as follows:
   - \( C_0(d)=\{d\} \)
   - \( C_1(d) \) is the set of all data objects that are directly contained by \( d \).
(a) If \( d \) is an image, all images created from an ROI or channel of \( d \) are directly contained by \( d \).
(b) If \( d \) is a pyramid, all pyramid levels of \( d \) are directly contained by \( d \).
(c) If \( d \) is an object array, all elements of \( d \) are directly contained by \( d \).
(d) If \( d \) is a delay object, all slots of \( d \) are directly contained by \( d \).

- For \( i > 1 \), \( C_i(d) \) is the set of all data objects that are contained by \( d \) at the \( i \)th order
  \[
  C_i(d) = \bigcup_{d' \in C_{i-1}(d)} C_1(d')
  \]

- \( C(d) \) is the set that contains \( d \) itself, the data objects contained by \( d \), the data objects that are contained by the data objects contained by \( d \) and so on. Formally:
  \[
  C(d) = \bigcup_{i=0}^{\infty} C_i(d)
  \]

2. \( I(d) \) is a predicate that equals true if and only if \( d \) is an image.

3. **Overlapping Relationship** - The overlapping relation \( R_{ov} \) is a relation defined for images, such that if \( i_1 \) and \( i_2 \) in \( C(i) \), \( i \) being an image, then \( i_1 R_{ov} i_2 \) is true if and only if \( i_1 \) and \( i_2 \) overlap, i.e., there exists a point \((x,y)\) of \( i \) that is contained in both \( i_1 \) and \( i_2 \). Note that this relation is reflexive and symmetric, but not transitive: \( i_1 \) overlaps \( i_2 \) and \( i_2 \) overlaps \( i_3 \) does not necessarily imply that \( i_1 \) overlaps \( i_3 \), as illustrated in the following figure:

![Figure 2.4: Overlap Example](image)

4. **Dependency Relationship** - The dependency relationship \( N_1 \rightarrow N_2 \) is a relation defined for nodes. \( N_1 \rightarrow N_2 \) means that \( N_2 \) depends on \( N_1 \) and then implies that \( N_2 \) must be executed after the completion of \( N_1 \).

5. \( N_1 \rightarrow N_2 \) if \( N_1 \) writes to a data object \( d_1 \) and \( N_2 \) reads from a data object \( d_2 \) and:

\[
 d_1 \in C(d_2) \text{ or } d_2 \in C(d_1) \text{ or } (I(d_1) \text{ and } I(d_2) \text{ and } d_1 R_{ov} d_2)
\]

If data object \( D_{y} \) of an output edge \((N_{x}, D_{y})\) overlaps with a data object \( D_{z} \) then the result is implementation defined.

### 2.8.7 Node Execution Independence

In the following example a client computes the gradient magnitude and gradient phase from a blurred input image. The \texttt{vxMagnitudeNode} and \texttt{vxPhaseNode} are independently computed, in that each does not depend on the output of the other. OpenVX does not mandate that they are run simultaneously or in parallel, but it could be implemented this way by the OpenVX vendor.
The code to construct such a graph can be seen below.

```c
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 2.5: A simple graph with some independent nodes.
2.8.8 Verification

Graphs within OpenVX must go through a rigorous validation process before execution to satisfy the design concept of eliminating run-time overhead (parameter checking) that guarantees safe execution of the graph. OpenVX must check for (but is not limited to) these conditions:

Parameters To Nodes:

• Each required parameter is given to the node (vx_parameter_state_e). Optional parameters may not be present and therefore are not checked when absent. If present, they are checked.

• Each parameter given to a node must be of the right direction (a value from vx_direction_e).

• Each parameter given to a node must be of the right object type (from the object range of vx_type_e).

• Each parameter attribute or value must be verified. In the case of a scalar value, it may need to be range checked (e.g., 0.5 <= k <= 1.0). The implementation is not required to do run-time range checking of scalar values. If the value of the scalar changes at run time to go outside the range, the results are undefined. The rationale is that the potential performance hit for run-time range checking is too large to be enforced. It will still be checked at graph verification time as a time-zero sanity check. If the scalar is an output parameter of another node, it must be initialized to a legal value. In the case of vxScaleImageNode, the relation of the input image dimensions to the output image dimensions determines the scaling factor. These values or attributes of data objects must be checked for compatibility on each platform.

• Graph Connectivity - the vx_graph must be a Directed Acyclic Graph (DAG). No cycles or feedback is allowed. The vx_delay object has been designed to explicitly address feedback between Graph executions.

• Resolution of Virtual Data Objects - Any changes to Virtual data objects from unspecified to specific format or dimensions, as well as the related creation of objects of specific type that are observable at processing time, takes place at Verification time.

The implementation must check that all node parameters are the correct type at node creation time, unless the parameter value is set to NULL. Additional checks may also be made on non-NULL parameters. The user must be allowed to set parameters to NULL at node creation time, even if they are required parameters, in order to create "exemplar" nodes that are not used in graph execution, or to create nodes incrementally. Therefore the implementation must not generate an error at node creation time for parameters that are explicitly set to NULL. However, the implementation must check that all required parameters are non-NULL and the correct type during vxVerifyGraph. Other more complex checks may also be done during vxVerifyGraph. The implementation should provide specific error reporting of NULL parameters during vxVerifyGraph, e.g., "Parameter<parameter> of Node<node> is NULL."

2.9 Callbacks

Callbacks are a method to control graph flow and to make decisions based on completed work. The vxAssignNodeCallback call takes as a parameter a callback function. This function will be called after the execution of the particular node, but prior to the completion of the graph. If nodes are arranged into independent sets, the order of the callbacks is unspecified. Nodes that are arranged in a serial fashion due to data dependencies perform callbacks in order. The callback function may use the node reference first to extract parameters from the node, and then extract the data references. Data outputs of Nodes with callbacks shall be available (via Map/Unmap/Copy methods) when the callback is called.

2.10 User Kernels

OpenVX supports the concept of client-defined functions that shall be executed as Nodes from inside the Graph or are Graph internal. The purpose of this paradigm is to:

• Further exploit independent operation of nodes within the OpenVX platform.

• Allow componentized functions to be reused elsewhere in OpenVX.

• Formalize strict verification requirements (i.e., Contract Programming).
In this example, to execute client-supplied functions, the graph does not have to be halted and then resumed. These nodes shall be executed in an independent fashion with respect to independent base nodes within OpenVX. This allows implementations to further minimize execution time if hardware to exploit this property exists.

### 2.10.1 Parameter Validation

User Kernels must aid in the Graph Verification effort by providing an explicit validation function for each vision function they implement. Each parameter passed to the instanced Node of a User Kernel is validated using the client-supplied validation function. The client must check these attributes and/or values of each parameter:

- Each attribute or value of the parameter must be checked. For example, the size of array, or the value of a scalar to be within a range, or a dimensionality constraint of an image such as width divisibility. (Some implementations may have restrictions, such as an image width be evenly divisible by some fixed number).
- If the output parameters depend on attributes or values from input parameters, those relationships must be checked.

#### The Meta Format Object

The Meta Format Object is an opaque object used to collect requirements about the output parameter, which then the OpenVX implementation will check. The Client must manually set relevant object attributes to be checked against output parameters, such as dimensionality, format, scaling, etc.

### 2.10.2 User Kernels Naming Conventions

User Kernels must be exported with a unique name (see Naming Conventions for information on OpenVX conventions) and a unique enumeration. Clients of OpenVX may use either the name or enumeration to retrieve a kernel,
so collisions due to non-unique names will cause problems. The kernel enumerations may be extended by following
this example:

```c
#define VX_KERNEL_NAME_KHR_XYZ "org.khronos.example.xyz"
#define VX_LIBRARY_XYZ (0x3) // assigned from Khronos, vendors control their own
enum vx_kernel_xyz_ext_e {
  VX_KERNEL_KHR_XYZ = VX_KERNEL_BASE(VX_ID_DEFAULT, VX_LIBRARY_XYZ) + 0x0,
  // up to 0xFFF kernel enums can be created.
};
```

Each vendor of a vision function or an implementation must apply to Khronos to get a unique identifier (up to a limit
of $2^{12} - 1$ vendors). Until they obtain a unique ID vendors must use VX_ID_DEFAULT.

To construct a kernel enumeration, a vendor must have both their ID and a library ID. The library ID's are
completely vendor defined (however when using the VX_ID_DEFAULT ID, many libraries may collide in namespace).

Once both are defined, a kernel enumeration may be constructed using the VX_KERNEL_BASE macro and an
offset. (The offset is optional, but very helpful for long enumerations.)

### 2.11 Immediate Mode Functions

OpenVX also contains an interface defined within `<VX/vxu.h>` that allows for immediate execution of vision
functions. These interfaces are prefixed with `vxu` to distinguish them from the Node interfaces, which are of the
form `vx<Name>Node`. Each of these interfaces replicates a Node interface with some exceptions. Immediate
mode functions are defined to behave as Single Node Graphs, which have no leaking side-effects (e.g., no Log
entries) within the Graph Framework after the function returns. The following tables refer to both the Immediate
Mode and Graph Mode vision functions. The Module documentation for each vision function draws a distinction on
each API by noting that it is either an immediate mode function with the tag [Immediate] or it is a Graph mode
function by the tag [Graph].

### 2.12 Targets

A 'Target' specifies a physical or logical devices where a node or an immediate mode function is executed. This
allows the use of different implementations of vision functions on different targets. The existence of allowed Tar-
gets is exposed to the applications by the use of defined APIs. The choice of a Target allows for different levels
of control on where the nodes can be executed. An OpenVX implementation must support at least one target.
Additional supported targets are specified using the appropriate enumerations. See `vxSetNodeTarget`, `vx-
SetImmediateModeTarget`, and `vx_target_e`. An OpenVX implementation must support at least one
target VX_TARGET_ANY as well as VX_TARGET_STRING enumerates. An OpenVX implementation may also
support more than these two to indicate the use of specific devices. For example, an implementation may add
VX_TARGET_CPU and VX_TARGET_GPU enumerates to indicate the support of two possible targets to assign
nodes to (or to exucute an immediate mode function). Another way an implementation can indicate the existence
of multiple targets, for example CPU and GPU, is by specifying the target as VX_TARGET_STRING and using
strings 'CPU' and 'GPU'. Thus defining targets using names rather than enumerates. The specific naming of string
or enumerates is not enforced by the specification and it is up to the vendors to document and communicate the
Target naming. Once available in a given implementation Applications can assign a Target to a node to specify
the target that must execute that node by using the API `vxSetNodeTarget`. For immediate mode functions the
target specifies the physical or logical device where the future execution of that function will be attempted. When an
immediate mode function is not supported on the selected target the execution falls back to VX_TARGET_ANY.

### 2.13 Base Vision Functions

OpenVX comes with a standard or base set of vision functions. The following table lists the supported set of vision
functions, their input types (first table) and output types (second table), and the version of OpenVX in which they are
supported.

#### 2.13.1 Inputs
<table>
<thead>
<tr>
<th>Vision Function</th>
<th>S8</th>
<th>U8</th>
<th>U16</th>
<th>S16</th>
<th>U32</th>
<th>F32</th>
<th>color</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsDiff</td>
<td>1.0</td>
<td>0</td>
<td>1.0</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accumulate</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AccumulateSquared</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AccumulateWeighted</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>1.0</td>
<td>0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>And</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BilateralFilter</td>
<td>1.2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Box3x3</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CannyEdgeDetector</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChannelCombine</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChannelExtract</td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ColorConvert</td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConvertDepth</td>
<td>1.0</td>
<td>0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convolve</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Object Copy</td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dilate3x3</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EqualizeHistogram</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Erode3x3</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FastCorners</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaussian3x3</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GaussianPyramid</td>
<td>1.0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HarrisCorners</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HalfScaleGaussian</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Histogram</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HOGCells</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HOGFeatures</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HoughLinesP</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntegralImage</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LaplacianPyramid</td>
<td>1.0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LaplacianReconstruct</td>
<td></td>
<td></td>
<td></td>
<td>1.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Vision Function

<table>
<thead>
<tr>
<th>Vision Function</th>
<th>S8</th>
<th>U8</th>
<th>U16</th>
<th>S16</th>
<th>U32</th>
<th>F32</th>
<th>color</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Magnitude</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>MatchTemplate</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>MeanStdDev</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Median3x3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Max</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Min</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>MinMaxLoc</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Multiply</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>NonLinearFilter</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>NonMaximaSuppression</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Not</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>OpticalFlowPyrLK</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Or</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Phase</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>GaussianPyramid</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Remap</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>ScaleImage</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Sobel3x3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Subtract</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>TableLookup</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.1</td>
</tr>
<tr>
<td>TensorMultiply</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
</tr>
<tr>
<td>TensorAdd</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
</tr>
<tr>
<td>TensorSubtract</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
</tr>
<tr>
<td>TensorMatrixMultiply</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
</tr>
<tr>
<td>TensorTableLookup</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
</tr>
<tr>
<td>TensorTranspose</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
</tr>
<tr>
<td>Threshold</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
</tr>
<tr>
<td>WarpAffine</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
</tr>
</tbody>
</table>

**S8**: 8-bit signed integer

**U8**: 8-bit unsigned integer

**U16**: 16-bit unsigned integer

**S16**: 16-bit signed integer

**U32**: 32-bit unsigned integer

**F32**: 32-bit float

**color**: Color information

**other**: Other information
## 2.13.2 Outputs

<table>
<thead>
<tr>
<th>Vision Function</th>
<th>S8</th>
<th>U8</th>
<th>U16</th>
<th>S16</th>
<th>U32</th>
<th>F32</th>
<th>color</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsDiff</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accumulate</td>
<td>1.→</td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AccumulateSquared</td>
<td>1.→</td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AccumulateWeighted</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>And</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BilateralFilter</td>
<td>1.→</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Box3x3</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CannyEdgeDetector</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChannelCombine</td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChannelExtract</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ColorConvert</td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConvertDepth</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convolve</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Object Copy</td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dilate3x3</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EqualizeHistogram</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Erode3x3</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FastCorners</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaussian3x3</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GaussianPyramid</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HarrisCorners</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HalfScaleGaussian</td>
<td>1.→</td>
<td>0</td>
<td>1.→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Histogram</td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HOGCells</td>
<td>1.→</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HOGFeatures</td>
<td>1.→</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HoughLinesP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
</tr>
</tbody>
</table>
### Vision Function

<table>
<thead>
<tr>
<th>Vision Function</th>
<th>S8</th>
<th>U8</th>
<th>U16</th>
<th>S16</th>
<th>U32</th>
<th>F32</th>
<th>color</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntegralImage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>LaplacianPyramid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.1</td>
</tr>
<tr>
<td>LaplacianReconstruct</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>LBP</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Magnitude</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>MatchTemplate</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MeanStdDev</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Median3x3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Max</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
</tr>
<tr>
<td>Min</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
</tr>
<tr>
<td>MinMaxLoc</td>
<td>1.0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Multiply</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>NonLinearFilter</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>NonMaximaSuppression</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.2</td>
</tr>
<tr>
<td>Not</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>OpticalFlowPyrLK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Or</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Phase</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>GaussianPyramid</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Remap</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScaleImage</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Subtract</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>TableLookup</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>TensorMultiply</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>TensorAdd</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>TensorSubtract</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>TensorMatrixMultiply</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>TensorTableLookup</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>TensorTranspose</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Threshold</td>
<td>1.0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
</tbody>
</table>
### 2.13.3 Parameter ordering convention

For vision functions, the input and output parameter ordering convention is:

1. Mandatory inputs
2. Optional inputs
3. Mandatory in/outs
4. Optional in/outs
5. Mandatory outputs
6. Optional outputs

The known exceptions are:
- `vxConvertDepthNode`
- `vxuConvertDepth`
- `vxOpticalFlowPyrLKNode`
- `vxuOpticalFlowPyrLK`
- `vxScaleImageNode`
- `vxuScaleImage`.

### 2.14 Lifecycles

#### 2.14.1 OpenVX Context Lifecycle

The lifecycle of the context is very simple.

![Diagram](image)

**Figure 2.7:** The lifecycle model for an OpenVX Context.

#### 2.14.2 Graph Lifecycle

OpenVX has four main phases of graph lifecycle:

- **Construction** - Graphs are created via `vxCreateGraph`, and Nodes are connected together by data objects.
- **Verification** - The graphs are checked for consistency, correctness, and other conditions. Memory allocation may occur.
CHAPTER 2. DESIGN OVERVIEW

- Execution - The graphs are executed via `vxProcessGraph` or `vxScheduleGraph`. Between executions data may be updated by the client or some other external mechanism. The client of OpenVX may change reference of input data to a graph, but this may require the graph to be validated again by checking `vxIsGraphVerified`.

- Deconstruction - Graphs are released via `vxReleaseGraph`. All Nodes in the Graph are released.

![Graph Lifecycle Diagram](image)

**Figure 2.8: Graph Lifecycle**

### 2.14.3 Data Object Lifecycle

All objects in OpenVX follow a similar lifecycle model. All objects are

- **Created** via `vxCreate<Object><Method>` or retrieved via `vxGet<Object><Method>` from the parent object if they are internally created.

- **Used** within Graphs or immediate functions as needed.

- **Then** objects must be released via `vxRelease<Object>` or `vxReleaseContext` when all objects are released.

**OpenVX Image Lifecycle**

This is an example of the Image Lifecycle using the OpenVX Framework API. This would also apply to other data types with changes to the types and function names.
2.15 Host Memory Data Object Access Patterns

For objects retrieved from OpenVX that are 2D in nature, such as \texttt{vx\_image}, \texttt{vx\_matrix}, and \texttt{vx\_\textasciitilde convolution}, the manner in which the host-side has access to these memory regions is well-defined. OpenVX uses a row-major storage (that is each unit in a column is memory-adjacent to its row adjacent unit). Two-dimensional objects are always created (using \texttt{vxCreateImage} or \texttt{vxCreateMatrix}) in width (columns) by height (rows) notation, with the arguments in that order. When accessing these structures in “C” with two-dimensional arrays of declared size, the user must therefore provide the array dimensions in the reverse of the order of the arguments to the \texttt{Create} function. This layout ensures row-wise storage in C on the host. A pointer could also be allocated for the matrix data and would have to be indexed in this row-major method.

2.15.1 Matrix Access Example

```c
const \texttt{vx\_size} columns = 3;
const \texttt{vx\_size} rows = 4;
\texttt{vx\_matrix} matrix = \texttt{vxCreateMatrix}(context,
\texttt{VX\_TYPE\_FLOAT32}, columns, rows);
\texttt{vx\_status} status = \texttt{vxGetStatus}((\texttt{vx\_reference})matrix);
if (status == \texttt{VX\_SUCCESS})
    {
        \texttt{vx\_int32} i;
        \#if defined(\texttt{OPENVX\_USE\_C99})
            \texttt{vx\_float32} mat[rows][columns]; /* note: row major */
        \#else
            \texttt{vx\_float32} *mat = (\texttt{vx\_float32} *)malloc(rows*columns*sizeof(\texttt{vx\_float32}));
        \#endif
        if (\texttt{vxCopyMatrix}(matrix, mat, \texttt{VX\_READ\_ONLY},
            \texttt{VX\_MEMORY\_TYPE\_HOST}) == \texttt{VX\_SUCCESS})
            {
                for (i = 0; i < \texttt{vx\_int32} rows; i++)
```

Figure 2.9: Image Object Lifecycle
2.15.2 Image Access Example

Images and Array differ slightly in how they are accessed due to more complex memory layout requirements.

```c
vx_status status = VX_SUCCESS;
void *base_ptr = NULL;
vx_uint32 width = 640, height = 480, plane = 0;
vx_image image = vxCreateImage(context, width, height,
VX_DF_IMAGE_U8);
vx_rectangle_t rect;
vx_imagepatch_addressing_t addr;
vx_map_id map_id;
rect.start_x = rect.start_y = 0;
rect.end_x = rect.end_y = PATCH_DIM;
status = vxMapImagePatch(image, &rect, plane, &map_id,
addr, &base_ptr,
VX_READ_AND_WRITE,
VX_MEMORY_TYPE_HOST, 0);
if (status == VX_SUCCESS)
{
    vx_uint8 pixel = 0;
    /* a couple addressing options */
    /* use linear addressing function/macro */
    for (i = 0; i < addr.dim_x*addr.dim_y; i++) {
        vx_uint8 *ptr2 = vxFormatImagePatchAddress1d(base_ptr,
            i, 4addr);
        *ptr2 = pixel;
    }
    /* 2d addressing option */
    for (y = 0; y < addr.dim_y; y+=addr.step_y) {
        for (x = 0; x < addr.dim_x; x+=addr.step_x) {
            vx_uint8 *ptr2 = vxFormatImagePatchAddress2d(base_ptr,
                x, y, 4addr);
            *ptr2 = pixel;
        }
    }
    /* direct addressing by client
    * for subsampled planes, scale will change */
    for (y = 0; y < addr.dim_y; y+=addr.step_y) {
        for (x = 0; x < addr.dim_x; x+=addr.step_x) {
            vx_uint8 *tmp = (vx_uint8 *)base_ptr;
            i = ((addr.stride_y*y*addr.scale_y) / 
                VX_SCALE_UNITY) +
                ((addr.stride_x*x*addr.scale_x) / 
                VX_SCALE_UNITY);
            tmp[i] = pixel;
        }
    }
    /* more efficient direct addressing by client.
    * for subsampled planes, scale will change. */
    for (y = 0; y < addr.dim_y; y+=addr.step_y) {
        for (x = 0; x < addr.dim_x; x+=addr.step_x) {
            vx_uint8 *tmp = (vx_uint8 *)base_ptr;
            i = j + (addr.stride_y*y*addr.scale_y) / 
                VX_SCALE_UNITY);
            tmp[i] = pixel;
        }
    }
}
```
/* this commits the data back to the image. */
status = vxUnmapImagePatch(image, map_id);
}
vxReleaseImage(&image);

2.15.3 Array Access Example

Arrays only require a single value, the stride, instead of the entire addressing structure that images need.

```c
vx_size i, stride = sizeof(vx_size);
void *base = NULL;
vx_map_id map_id;
/* access entire array at once */
vxMapArrayRange(array, 0, num_items, &map_id, &stride, &base,
VX_READ_AND_WRITE, VX_MEMORY_TYPE_HOST, 0);
for (i = 0; i < num_items; i++)
{
    vxArrayItem(mystruct, base, i, stride).some_uint += i;
    vxArrayItem(mystruct, base, i, stride).some_double = 3.14f;
}
vxUnmapArrayRange(array, map_id);
```

Map/Unmap pairs can also be called on individual elements of array using a method similar to this:

```c
/* access each array item individually */
for (i = 0; i < num_items; i++)
{
    mystruct *myptr = NULL;
    vxMapArrayRange(array, i, i+1, &map_id, &stride, (void **)&myptr,
VX_READ_AND_WRITE, VX_MEMORY_TYPE_HOST, 0);
    myptr->some_uint += 1;
    myptr->some_double = 3.14f;
    vxUnmapArrayRange(array, map_id);
}
```

2.16 Concurrent Data Object Access

Accessing OpenVX data-objects using the functions Map, Copy, Read concurrently to an execution of a graph that is accessing the same data objects is permitted only if all accesses are read-only. That is, for Map, Copy to have a read-only access mode and for nodes in the graph to have that data-object as an input parameter only. In all other cases, including write or read-write modes and Write access function, as well as a graph nodes having the data-object as output or bidirectional, the application must guarantee that the access is not performed concurrently with the graph execution. That can be achieved by calling un-map following a map before calling `vxScheduleGraph` or `vxProcessGraph`. In addition, the application must call `vxWaitGraph` after `vxScheduleGraph` before calling Map, Read, Write or Copy to avoid restricted concurrent access. An application that fails to follow the above might encounter an undefined behavior and/or data loss without being notified by the OpenVX framework. Accessing images created from ROI (`vxCreateImageFromROI`) or created from a channel (`vxCreateImageFromChannel`) must be treated as if the entire image is being accessed.

- Setting an attribute is considered as writing to a data object in this respect.
- For concurrent execution of several graphs please see Execution Model
- Also see the graph formalism section for guidance on accessing ROIs of the same image within a graph.

2.17 Valid Image Region

The valid region mechanism informs the application as to which pixels of the output images of a graph's execution have valid values (see valid pixel definition below). The mechanism also applies to immediate mode (VXU) calls, and supports the communication of the valid region between different graph executions. Some vision functions, mainly those providing statistics and summarization of image information, use the valid region to ignore pixels that are not valid on their inputs (potentially bad or unstable pixel values). A good example of such a function is Min/Max Location. Formalization of the valid region mechanism is given below.

- Valid Pixels - All output pixels of an OpenVX function are considered valid by default, unless their calculation depends on input pixels that are not valid. An input pixel is not valid in one of two situations:
1. The pixel is outside of the image border and the border mode in use is \texttt{VX_BORDER_UNDEFINED}.
2. The pixel is outside the valid region of the input image.

- **Valid Region** - The region in the image that contains all the valid pixels. Theoretically this can be of any shape. OpenVX currently only supports rectangular valid regions. In subsequent text the term 'valid rectangle' denotes a valid region that is rectangular in shape.

- **Valid Rectangle Reset** - In some cases it is not possible to calculate a valid rectangle for the output image of a vision function (for example, warps and remap). In such cases, the vision function is said to reset the valid Region to the entire image. The attribute \texttt{VX_NODE_VALID_RECT_RESET} is a read only attribute and is used to communicate valid rectangle reset behavior to the application. When it is set to \texttt{vx_true_e} for a given node the valid rectangle of the output images will reset to the full image upon execution of the node, when it is set to \texttt{vx_false_e} the valid rectangle will be calculated. All standard OpenVX functions will have this attribute set to \texttt{vx_false_e} by default, except for Warp and Remap where it will be set to \texttt{vx_true_e}.

- **Valid Rectangle Initialization** - Upon the creation of an image, its valid rectangle is the entire image. One exception to this is when creating an image via \texttt{vxCreateImageFromROI}; in that case, the valid region of the ROI image is the subset of the valid region of the parent image that is within the ROI. In other words, the valid region of an image created using an ROI is the largest rectangle that contains valid pixels in the parent image.

- **Valid Rectangle Calculation** - The valid rectangle of an image changes as part of the graph execution, the correct value is guaranteed only when the execution finishes. The valid rectangle of an image remains unchanged between graph executions and persists between graph executions as long as the application doesn’t explicitly change the valid region via \texttt{vxSetImageValidRectangle}. Notice that using \texttt{vxMapImagePatch}, \texttt{vxUnmapImagePatch} or \texttt{vxSwapImageHandle} does not change the valid region of an image. If a non-UNDEFINED border mode is used on an image where the valid region is not the full image, the results at the border and resulting size of the valid region are implementation-dependent. This case can occur when mixing UNDEFINED and other border mode, which is not recommended.

- **Valid Rectangle for Immediate mode (VXU)** - VXU is considered a single node graph execution, thus the valid rectangle of an output of VXU will be propagated for an input to a consequent VXU call (when using the same output image from one call as input to the consecutive call).

- **Valid Region Usage** - For all standard OpenVX functions, the framework must guarantee that all pixel values inside the valid rectangle of the output images are valid. The framework does not guarantee that input pixels outside of the valid rectangle are processed. For the following vision functions, the framework guarantees that pixels outside of the valid rectangle do not participate in calculating the vision function result: Equalize Histogram, Integral Image, Fast Corners, Histogram, Mean and Standard Deviation, Min Max Location, Optical Flow Pyramid (LK) and Canny Edge Detector. An application can get the valid rectangle of an image by using \texttt{vxGetValidRegionImage}.

- **User kernels** - User kernels may change the valid rectangles of their output images. To change the valid rectangle, the programmer of the user kernel must provide a call-back function that sets the valid rectangle. The output validator of the user kernel must provide this callback by setting the value of the \texttt{vx_meta_format attribute VX_VALID_RECT_CALLBACK} during the output validator. The callback function must be callable by the OpenVX framework during graph validation and execution. Assumptions must not be made regarding the order and the frequency by which the valid rectangle callback is called. The framework will recalculate the valid region when a change in the input valid regions is detected. For user nodes, the default value of \texttt{VX_NODE_VALID_RECT_RESET} is \texttt{vx_true_e}. Setting \texttt{VX_VALID_RECT_CALLBACK} during parameter validation to a value other than NULL will result in setting \texttt{VX_NODE_VALID_RECT_RESET} to \texttt{vx_false_e}. Note: the above means that when \texttt{VX_VALID_RECT_CALLBACK} is not set or set to NULL the user-node will reset the valid rectangle to the entire image.

- **In addition, valid rectangle reset occurs in the following scenarios:**
  1. A reset of the valid rectangle of a parent image when a node writes to one of its ROIs. The only case where the reset does not occur is when the child ROI image is identical to the parent image.
  2. For nodes that have the \texttt{VX_NODE_VALID_RECT_RESET} set to \texttt{vx_true_e}

1. The pixel is outside of the image border and the border mode in use is \texttt{VX_BORDER_UNDEFINED}.
2. The pixel is outside the valid region of the input image.
2.18 Extending OpenVX

Beyond User Kernels there are other mechanisms for vendors to extend features in OpenVX. These mechanisms are not available to User Kernels. Each OpenVX official extension has a unique identifier, comprised of capital letters, numbers and the underscore character, prefixed with "KHR_," for example "KHR_NEW_FEATURE".

2.18.1 Extending Attributes

When extending attributes, vendors must use their assigned ID from `vx_vendor_id_e` in conjunction with the appropriate macros for creating new attributes with `VX_ATTRIBUTE_BASE`. The typical mechanism to extend a new attribute for some object type (for example a `vx_node` attribute from `VX_ID_TI`) would look like this:

```c
enum {
    VX_NODE_TI_NEWTHING = VX_ATTRIBUTE_BASE(VX_ID_TI, VX_TYPE_NODE) + 0x0,
}
```

2.18.2 Vendor Custom Kernels

Vendors wanting to add more kernels to the base set supplied to OpenVX should provide a header of the form

```c
#include <VX/vx_ext_<vendor>.h>
```

c contains definitions of each of the following.

- New Node Creation Function Prototype per function.
  ```c
  vx_node vxXYZNode(vx_graph graph, vx_image input,
    vx_uint32 value, vx_image output, vx_array temp);
  ```

- A new Kernel Enumeration(s) and Kernel String per function.
  ```c
  #define VX_KERNEL_NAME_KHR_XYZ "org.khronos.example.xyz"
  #define VX_LIBRARY_XYZ (0x3) // assigned from Khronos, vendors control their own
  enum vx_kernel_xyz_ext_e {
    VX_KERNEL_KHR_XYZ = VX_KERNEL_BASE(VX_ID_DEFAULT, VX_LIBRARY_XYZ) + 0x0,
    // up to 0xFFF kernel enums can be created.
  };
  ```

  ```c
  vx_status vxuXYZ(vx_context context, vx_image input,
    vx_uint32 value, vx_image output, vx_array temp);
  ```

This should come with good documentation for each new part of the extension. Ideally, these sorts of extensions should not require linking to new objects to facilitate usage.

2.18.3 Vendor Custom Extensions

Some extensions affect base vision functions and thus may be invisible to most users. In these circumstances, the vendor must report the supported extensions to the base nodes through the `VX_CONTEXT_EXTENSIONS` attribute on the context.

```c
vx_char *tmp, *extensions = NULL;
vx_size size = 0;
vxQueryContext(context, VX_CONTEXT_EXTENSIONS_SIZE, &size,sizeof(vx_size));
extensions = malloc(size);
.vxQueryContext(context, VX_CONTEXT_EXTENSIONS, extensions, size);
```

Extensions in this list are dependent on the extension itself; they may or may not have a header and new kernels or framework feature or data objects. The common feature is that they are implemented and supported by the implementation vendor.
CHAPTER 2. DESIGN OVERVIEW

2.18.4 Hinting

The specification defines a Hinting API that allows Clients to feed information to the implementation for optional behavior changes. See Framework: Hints. It is assumed that most of the hints will be vendor- or implementation-specific. Check with the OpenVX implementation vendor for information on vendor-specific extensions.

2.18.5 Directives

The specification defines a Directive API to control implementation behavior. See Framework: Directives. This may allow things like disabling parallelism for debugging, enabling cache writing-through for some buffers, or any implementation-specific optimization.
Chapter 3

Module Documentation

3.1 Vision Functions

3.1.1 Detailed Description

These are the base vision functions supported in OpenVX 1.1. These functions were chosen as a subset of a larger pool of possible functions that fall under the following criteria:

- Applicable to Acceleration Hardware
- Very Common Usage
- Encumbrance Free

Modules

- **Absolute Difference**
  Computes the absolute difference between two images. The output image dimensions should be the same as the dimensions of the input images.

- **Accumulate**
  Accumulates an input image into output image. The accumulation image dimensions should be the same as the dimensions of the input image.

- **Accumulate Squared**
  Accumulates a squared value from an input image to an output image. The accumulation image dimensions should be the same as the dimensions of the input image.

- **Data Object Copy**
  Copy a data object to another.

- **Accumulate Weighted**
  Accumulates a weighted value from an input image to an output image. The accumulation image dimensions should be the same as the dimensions of the input image.

- **Arithmetic Addition**
  Performs addition between two images. The output image dimensions should be the same as the dimensions of the input images.

- **Arithmetic Subtraction**
  Performs subtraction between two images. The output image dimensions should be the same as the dimensions of the input images.

- **Bitwise AND**
  Performs a bitwise AND operation between two VX_DF_IMAGE_U8 images. The output image dimensions should be the same as the dimensions of the input images.

- **Bitwise EXCLUSIVE OR**
  Performs a bitwise EXCLUSIVE OR (XOR) operation between two VX_DF_IMAGE_U8 images. The output image dimensions should be the same as the dimensions of the input images.
• Bitwise INCLUSIVE OR
  Performs a bitwise INCLUSIVE OR operation between two VX_DF_IMAGE_U8 images. The output image dimensions should be the same as the dimensions of the input images.

• Bitwise NOT
  Performs a bitwise NOT operation on a VX_DF_IMAGE_U8 input image. The output image dimensions should be the same as the dimensions of the input image.

• Box Filter
  Computes a Box filter over a window of the input image. The output image dimensions should be the same as the dimensions of the input image.

• Non-Maxima Suppression
  Find local maxima in an image, or otherwise suppress pixels that are not local maxima.

• Canny Edge Detector
  Provides a Canny edge detector kernel. The output image dimensions should be the same as the dimensions of the input image.

• Channel Combine
  Implements the Channel Combine Kernel.

• Channel Extract
  Implements the Channel Extraction Kernel.

• Color Convert
  Implements the Color Conversion Kernel. The output image dimensions should be the same as the dimensions of the input image.

• Convert Bit depth
  Converts image bit depth. The output image dimensions should be the same as the dimensions of the input image.

• Custom Convolution
  Convolves the input with the client supplied convolution matrix. The output image dimensions should be the same as the dimensions of the input image.

• Dilate Image
  Implements Dilation, which grows the white space in a VX_DF_IMAGE_U8 Boolean image. The output image dimensions should be the same as the dimensions of the input image.

• Equalize Histogram
  Equalizes the histogram of a grayscale image. The output image dimensions should be the same as the dimensions of the input image.

• Erode Image
  Implements Erosion, which shrinks the white space in a VX_DF_IMAGE_U8 Boolean image. The output image dimensions should be the same as the dimensions of the input image.

• Fast Corners
  Computes the corners in an image using a method based upon FAST9 algorithm suggested in [?] and with some updates from [?] with modifications described below.

• Gaussian Filter
  Computes a Gaussian filter over a window of the input image. The output image dimensions should be the same as the dimensions of the input image.

• Non Linear Filter
  Computes a non-linear filter over a window of the input image. The output image dimensions should be the same as the dimensions of the input image.

• Harris Corners
  Computes the Harris Corners of an image.

• Histogram
  Generates a distribution from an image.

• Gaussian Image Pyramid
  Computes a Gaussian Image Pyramid from an input image.

• Laplacian Image Pyramid
  Computes a Laplacian Image Pyramid from an input image.

• Reconstruction from a Laplacian Image Pyramid
Reconstructs the original image from a Laplacian Image Pyramid.

- **Integral Image**
  Computes the integral image of the input. The output image dimensions should be the same as the dimensions of the input image.

- **Magnitude**
  Implements the Gradient Magnitude Computation Kernel. The output image dimensions should be the same as the dimensions of the input images.

- **Mean and Standard Deviation**
  Computes the mean pixel value and the standard deviation of the pixels in the input image (which has a dimension width and height).

- **Median Filter**
  Computes a median pixel value over a window of the input image. The output image dimensions should be the same as the dimensions of the input image.

- **Min, Max Location**
  Finds the minimum and maximum values in an image and a location for each.

- **Min**
  Implements a pixel-wise minimum kernel. The output image dimensions should be the same as the dimensions of the input image.

- **Max**
  Implements a pixel-wise maximum kernel. The output image dimensions should be the same as the dimensions of the input image.

- **Optical Flow Pyramid (LK)**
  Computes the optical flow using the Lucas-Kanade method between two pyramid images.

- **Phase**
  Implements the Gradient Phase Computation Kernel. The output image dimensions should be the same as the dimensions of the input images.

- **Pixel-wise Multiplication**
  Performs element-wise multiplication between two images and a scalar value. The output image dimensions should be the same as the dimensions of the input images.

- **Remap**
  Maps output pixels in an image from input pixels in an image.

- **Scale Image**
  Implements the Image Resizing Kernel.

- **Sobel 3x3**
  Implements the Sobel Image Filter Kernel. The output images dimensions should be the same as the dimensions of the input image.

- **TableLookup**
  Implements the Table Lookup Image Kernel. The output image dimensions should be the same as the dimensions of the input image.

- **Thresholding**
  Thresholds an input image and produces an output Boolean image. The output image dimensions should be the same as the dimensions of the input image.

- **Warp Affine**
  Performs an affine transform on an image.

- **Warp Perspective**
  Performs a perspective transform on an image.

- **Bilateral Filter**
  The function applies bilateral filtering to the input tensor.

- **MatchTemplate**
  Compares an image template against overlapped image regions.

- **LBP**
  Extracts LBP image from an input image. The output image dimensions should be the same as the dimensions of the input image.
• HOG
  Extracts Histogram of Oriented Gradients features from the input grayscale image.

• HoughLinesP
  Finds the Probabilistic Hough Lines detected in the input binary image.

• Tensor Multiply
  Performs element wise multiplications on element values in the input tensor data with a scale.

• Tensor Add
  Performs arithmetic addition on element values in the input tensor data.

• Tensor Subtract
  Performs arithmetic subtraction on element values in the input tensor data.

• Tensor TableLookUp
  Performs LUT on element values in the input tensor data.

• Tensor Transpose
  Performs transpose on the input tensor.

• Tensor Convert Bit-Depth
  Creates a bit-depth conversion node.

• Tensor Matrix Multiply
  Creates a generalized matrix multiplication node.
3.2 Absolute Difference

3.2.1 Detailed Description

Computes the absolute difference between two images. The output image dimensions should be the same as the dimensions of the input images.

Absolute Difference is computed by:

\[
\text{out}(x,y) = |\text{in}_1(x,y) - \text{in}_2(x,y)|
\]

If one of the input images is of type \texttt{VX_DF_IMAGE_S16}, all values are converted to \texttt{vx_int32} and the overflow policy \texttt{VX_CONVERT_POLICY_SATURATE} is used.

\[
\text{out}(x,y) = \text{saturate}_{\text{int16}}(|(\text{int32})\text{in}_1(x,y) - (\text{int32})\text{in}_2(x,y)|)
\]

The output image can be \texttt{VX_DF_IMAGE_U8} only if both source images are \texttt{VX_DF_IMAGE_U8} and the output image is explicitly set to \texttt{VX_DF_IMAGE_U8}. It is otherwise \texttt{VX_DF_IMAGE_S16}.

Functions

- \texttt{vx_node VX_API_CALL vxAbsDiffNode (vx_graph graph, vx_image in1, vx_image in2, vx_image out)}
  
  [Graph] Creates an AbsDiff node.

- \texttt{vx_status VX_API_CALL vxuAbsDiff (vx_context context, vx_image in1, vx_image in2, vx_image out)}
  
  [Immediate] Computes the absolute difference between two images.

3.2.2 Function Documentation

\texttt{vx_node VX_API_CALL vxAbsDiffNode ( vx_graph graph, vx_image in1, vx_image in2, vx_image out )}

[Graph] Creates an AbsDiff node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>in1</td>
<td>An input image in \texttt{VX_DF_IMAGE_U8} or \texttt{VX_DF_IMAGE_S16} format.</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>An input image in \texttt{VX_DF_IMAGE_U8} or \texttt{VX_DF_IMAGE_S16} format.</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The output image in \texttt{VX_DF_IMAGE_U8} or \texttt{VX_DF_IMAGE_S16} format, which must have the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

Return values

\texttt{vx_node} A node reference. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus}

\texttt{vx_status VX_API_CALL vxuAbsDiff ( vx_context context, vx_image in1, vx_image in2, vx_image out )}

[Immediate] Computes the absolute difference between two images.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>in1</td>
<td>An input image in \texttt{VX_DF_IMAGE_U8} or \texttt{VX_DF_IMAGE_S16} format.</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>An input image in \texttt{VX_DF_IMAGE_U8} or \texttt{VX_DF_IMAGE_S16} format.</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The output image in \texttt{VX_DF_IMAGE_U8} or \texttt{VX_DF_IMAGE_S16} format.</td>
</tr>
</tbody>
</table>
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.3 Accumulate

3.3.1 Detailed Description

Accumulates an input image into output image. The accumulation image dimensions should be the same as the dimensions of the input image.

Accumulation is computed by:

\[ \text{accum}(x, y) = \text{accum}(x, y) + \text{input}(x, y) \]

The overflow policy used is \texttt{VX_CONVERT_POLICY_SATURATE}.

Functions

- \texttt{vx_node VX_API_CALL vxAccumulateImageNode (vx_graph graph, vx_image input, vx_image accum)}
  
  [Graph] Creates an accumulate node.

- \texttt{vx_status VX_API_CALL vxuAccumulateImage (vx_context context, vx_image input, vx_image accum)}
  
  [Immediate] Computes an accumulation.

3.3.2 Function Documentation

\texttt{vx_node VX_API_CALL vxAccumulateImageNode ( vx_graph graph, vx_image input, vx_image accum )}

[Graph] Creates an accumulate node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input \texttt{VX_DF_IMAGE_U8} image.</td>
</tr>
<tr>
<td>in,out</td>
<td>accum</td>
<td>The accumulation image in \texttt{VX_DF_IMAGE_S16}, which must have the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

Returns

\texttt{vx_node}.

Return values

| \texttt{vx_node} | A node reference. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus} |

\texttt{vx_status VX_API_CALL vxuAccumulateImage ( vx_context context, vx_image input, vx_image accum )}

[Immediate] Computes an accumulation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input \texttt{VX_DF_IMAGE_U8} image.</td>
</tr>
<tr>
<td>in,out</td>
<td>accum</td>
<td>The accumulation image in \texttt{VX_DF_IMAGE_S16}</td>
</tr>
</tbody>
</table>

Returns

A \texttt{vx_status_e} enumeration.
Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.4 Accumulate Squared

3.4.1 Detailed Description

Accumulates a squared value from an input image to an output image. The accumulation image dimensions should be the same as the dimensions of the input image.

Accumulate squares is computed by:

\[ \text{accum}(x,y) = \text{saturate}_{int16}(\text{uint16}(\text{input}(x,y)^2) >> \text{shift}) \]

Where \(0 \leq \text{shift} \leq 15\)

The overflow policy used is \texttt{VX\_CONVERT\_POLICY\_SATURATE}.

Functions

- \texttt{vx_node VX\_API\_CALL vxAccumulateSquareImageNode (vx\_graph graph, vx\_image input, vx\_scalar shift, vx\_image accum)}
  
  [Graph] Creates an accumulate square node.

- \texttt{vx_status VX\_API\_CALL vxuAccumulateSquareImage (vx\_context context, vx\_image input, vx\_scalar shift, vx\_image accum)}
  
  [Immediate] Computes a squared accumulation.

3.4.2 Function Documentation

\texttt{vx_node VX\_API\_CALL vxAccumulateSquareImageNode (vx\_graph graph, vx\_image input, vx\_scalar shift, vx\_image accum)}

[Graph] Creates an accumulate square node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input \texttt{VX_DF_IMAGE_U8} image.</td>
</tr>
<tr>
<td>in</td>
<td>shift</td>
<td>The input \texttt{VX_TYPE_UINT32} with a value in the range of (0 \leq \text{shift} \leq 15).</td>
</tr>
<tr>
<td>in,out</td>
<td>accum</td>
<td>The accumulation image in \texttt{VX_DF_IMAGE_S16}, which must have the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

Returns

\texttt{vx\_node}.

Return values

\texttt{vx\_node} A node reference. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus}.

\texttt{vx\_status VX\_API\_CALL vxuAccumulateSquareImage (vx\_context context, vx\_image input, vx\_scalar shift, vx\_image accum)}

[Immediate] Computes a squared accumulation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input \texttt{VX_DF_IMAGE_U8} image.</td>
</tr>
<tr>
<td>in</td>
<td>shift</td>
<td>A \texttt{VX_TYPE_UINT32} type, the input value with the range (0 \leq \text{shift} \leq 15).</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td><em>accum</em></td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>*</td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.5 Data Object Copy

3.5.1 Detailed Description

Copy a data object to another.

Copy data from an input data object into another data object. The input and output object must have the same object type and meta data. If these objects are object arrays, or pyramids then a deep copy shall be performed.

Functions

- vx_node VX_API_CALL vxCopyNode (vx_graph graph, vx_reference input, vx_reference output)
  Copy data from one object to another.

- vx_status VX_API_CALL vxuCopy (vx_context context, vx_reference input, vx_reference output)
  [Immediate] Copy data from one object to another.

3.5.2 Function Documentation

vx_node VX_API_CALL vxCopyNode (vx_graph graph, vx_reference input, vx_reference output)
Copy data from one object to another.

Note
An implementation may optimize away the copy when virtual data objects are used.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input data object.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output data object with meta-data identical to the input data object.</td>
</tr>
</tbody>
</table>

Returns

vx_node.

Return values

vx_node A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus

vx_status VX_API_CALL vxuCopy (vx_context context, vx_reference input, vx_reference output)
[Immediate] Copy data from one object to another.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input data object.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output data object.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.
Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.6 Accumulate Weighted

3.6.1 Detailed Description

Accumulates a weighted value from an input image to an output image. The accumulation image dimensions should be the same as the dimensions of the input image.

Weighted accumulation is computed by:

\[ \text{accum}(x,y) = (1 - \alpha) \times \text{accum}(x,y) + \alpha \times \text{input}(x,y) \]

Where \(0 \leq \alpha \leq 1\) Conceptually, the rounding for this is defined as:

\[ \text{output}(x,y) = \text{uint8}( (1 - \alpha) \times \text{float32}(\text{int32}(\text{output}(x,y))) + \alpha \times \text{float32}(\text{int32}(\text{input}(x,y)))) \]

Functions

- **vx_node VX_API_CALL vxAccumulateWeightedImageNode (vx_graph graph, vx_image input, vx_scalar alpha, vx_image accum)**
  
  [Graph] Creates a weighted accumulate node.

- **vx_status VX_API_CALL vxuAccumulateWeightedImage (vx_context context, vx_image input, vx_scalar alpha, vx_image accum)**
  
  [Immediate] Computes a weighted accumulation.

3.6.2 Function Documentation

**vx_node VX_API_CALL vxAccumulateWeightedImageNode (vx_graph graph, vx_image input, vx_scalar alpha, vx_image accum)**

[Graph] Creates a weighted accumulate node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input VX_DF_IMAGE_U8 image.</td>
</tr>
<tr>
<td>in</td>
<td>alpha</td>
<td>The input VX_TYPE_FLOAT32 scalar value with a value in the range of (0.0 \leq \alpha \leq 1.0).</td>
</tr>
<tr>
<td>in,out</td>
<td>accum</td>
<td>The VX_DF_IMAGE_U8 accumulation image, which must have the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

Returns

**vx_node.**

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

**vx_status VX_API_CALL vxuAccumulateWeightedImage (vx_context context, vx_image input, vx_scalar alpha, vx_image accum)**

[Immediate] Computes a weighted accumulation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input VX_DF_IMAGE_U8 image.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>* alpha</td>
</tr>
<tr>
<td>in, out</td>
<td>accum</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.7 Control Flow

3.7.1 Detailed Description

Defines the predicated execution model of OpenVX.

These features allow for conditional graph flow in OpenVX, via support for a variety of operations between two scalars. The supported scalar data types \(\text{VX\_TYPE\_BOOL}, \text{VX\_TYPE\_INT8}, \text{VX\_TYPE\_UINT8}, \text{VX\_TYPE\_INT16}, \text{VX\_TYPE\_UINT16}, \text{VX\_TYPE\_INT32}, \text{VX\_TYPE\_UINT32}, \text{VX\_TYPE\_SIZE}, \text{VX\_TYPE\_FLOAT}\) are supported.

Summary of logical operations:

<table>
<thead>
<tr>
<th>Scalar Operation</th>
<th>Equation</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SCALAR_OP_AND</td>
<td>(\text{output} = (a&amp;b))</td>
<td>(\text{bool} = \text{bool op bool})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_OR</td>
<td>(\text{output} = (a\mid b))</td>
<td>(\text{bool} = \text{bool op bool})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_XOR</td>
<td>(\text{output} = (a \oplus b))</td>
<td>(\text{bool} = \text{bool op bool})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_NAND</td>
<td>(\text{output} = \neg(a&amp;b))</td>
<td>(\text{bool} = \text{bool op bool})</td>
</tr>
</tbody>
</table>

Summary of comparison operations:

<table>
<thead>
<tr>
<th>Scalar Operation</th>
<th>Equation</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SCALAR_OP_EQUAL</td>
<td>(\text{output} = (a == b))</td>
<td>(\text{bool} = \text{num op num})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_NOTEQUAL</td>
<td>(\text{output} = (a \neq b))</td>
<td>(\text{bool} = \text{num op num})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_LESS</td>
<td>(\text{output} = (a &lt; b))</td>
<td>(\text{bool} = \text{num op num})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_LESSEQ</td>
<td>(\text{output} = (a &lt;= b))</td>
<td>(\text{bool} = \text{num op num})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_GREATER</td>
<td>(\text{output} = (a &gt; b))</td>
<td>(\text{bool} = \text{num op num})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_GREATEREQ</td>
<td>(\text{output} = (a &gt;= b))</td>
<td>(\text{bool} = \text{num op num})</td>
</tr>
</tbody>
</table>

Summary of arithmetic operations:

<table>
<thead>
<tr>
<th>Scalar Operation</th>
<th>Equation</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SCALAR_OP_ADD</td>
<td>(\text{output} = (a+b))</td>
<td>(\text{num} = \text{num op num})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_SUBTRACT</td>
<td>(\text{output} = (a-b))</td>
<td>(\text{num} = \text{num op num})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_MULTIPLY</td>
<td>(\text{output} = (a*b))</td>
<td>(\text{num} = \text{num op num})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_DIVIDE</td>
<td>(\text{output} = (a/b))</td>
<td>(\text{num} = \text{num op num})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_MODULUS</td>
<td>(\text{output} = (a\mod b))</td>
<td>(\text{num} = \text{num op num})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_MIN</td>
<td>(\text{output} = \min(a,b))</td>
<td>(\text{num} = \text{num op num})</td>
</tr>
<tr>
<td>VX_SCALAR_OP_MAX</td>
<td>(\text{output} = \max(a,b))</td>
<td>(\text{num} = \text{num op num})</td>
</tr>
</tbody>
</table>

Please note that in the above tables:

- \(\text{bool}\) denotes a scalar of data type \(\text{VX\_TYPE\_BOOL}\)
- \(\text{num}\) denotes supported scalar data types are \(\text{VX\_TYPE\_INT8}, \text{VX\_TYPE\_UINT8}, \text{VX\_TYPE\_INT16}, \text{VX\_TYPE\_UINT16}, \text{VX\_TYPE\_INT32}, \text{VX\_TYPE\_UINT32}, \text{VX\_TYPE\_SIZE}, \text{VX\_TYPE\_FLOAT}\) or \(\text{AT32}\).
- The \(\text{VX\_SCALAR\_OP\_MODULUS}\) operation supports integer operands.
- The results of \(\text{VX\_SCALAR\_OP\_DIVIDE}\) and \(\text{VX\_SCALAR\_OP\_MODULUS}\) operations with the second argument as zero, must be defined by the implementation.
- For arithmetic and comparison operations with mixed input data types, the results will be mathematically accurate without the side effects of internal data representations.
- If the operation result can not be stored in output data type without data and/or precision loss, the following rules shall be applied:
1. If the operation result is integer and output is floating-point, the operation result is promoted to floating-point.

2. If the operation result is floating-point and output is an integer, the operation result is converted to integer with rounding policy VX_ROUND_POLICY_TO_ZERO and conversion policy VX_CONVERT_POLICY_SATURATE.

3. If both operation result and output are integers, the result is converted to output data type with VX_CONVERT_POLICY_WRAP conversion policy.

Functions

- vx_node VX_API_CALL vxScalarOperationNode (vx_graph graph, vx_enum scalar_operation, vx_scalar a, vx_scalar b, vx_scalar output)
  
  [Graph] Creates a scalar operation node.

- vx_node VX_API_CALL vxSelectNode (vx_graph graph, vx_scalar condition, vx_reference true_value, vx_reference false_value, vx_reference output)
  
  [Graph] Selects one of two data objects depending on the value of a condition (boolean scalar), and copies its data into another data object.

### 3.7.2 Function Documentation

#### vx_node VX_API_CALL vxScalarOperationNode ( vx_graph graph, vx_enum scalar_operation, vx_scalar a, vx_scalar b, vx_scalar output )

[Graph] Creates a scalar operation node.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in scalar_operation</td>
<td>A VX_TYPE_ENUM of the vx_scalar_operation_e enumeration.</td>
</tr>
<tr>
<td>in a</td>
<td>First scalar operand.</td>
</tr>
<tr>
<td>in b</td>
<td>Second scalar operand.</td>
</tr>
<tr>
<td>out output</td>
<td>Result of the scalar operation.</td>
</tr>
</tbody>
</table>

**Returns**

vx_node.

**Return values**

<table>
<thead>
<tr>
<th>vx_node</th>
<th>A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus</th>
</tr>
</thead>
</table>

#### vx_node VX_API_CALL vxSelectNode ( vx_graph graph, vx_scalar condition, vx_reference true_value, vx_reference false_value, vx_reference output )

[Graph] Selects one of two data objects depending on the value of a condition (boolean scalar), and copies its data into another data object.

This node supports predicated execution flow within a graph. All the data objects passed to this kernel shall have the same object type and meta data. It is important to note that an implementation may optimize away the select and copy when virtual data objects are used.

If there is a kernel node that contribute only into virtual data objects during the graph execution due to certain data path being eliminated by not taken argument of select node, then the OpenVX implementation guarantees that there will not be any side effects to graph execution and node state.

If the path to a select node contains non-virtual objects, user nodes, or nodes with completion callbacks, then
that path may not be "optimized out" because the callback must be executed and the non-virtual objects must be modified.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td><strong>in</strong> condition</td>
<td>VX_TYPE_BOOL predicate variable.</td>
</tr>
<tr>
<td><strong>in</strong> true_value</td>
<td>Data object for true.</td>
</tr>
<tr>
<td><strong>in</strong> false_value</td>
<td>Data object for false.</td>
</tr>
<tr>
<td><strong>out</strong> output</td>
<td>Output data object.</td>
</tr>
</tbody>
</table>

Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |
3.8 Arithmetic Addition

3.8.1 Detailed Description

Performs addition between two images. The output image dimensions should be the same as the dimensions of the input images.

Arithmetic addition is performed between the pixel values in two \texttt{VX_DF_IMAGE_U8} or \texttt{VX_DF_IMAGE_S16} images. The output image can be \texttt{VX_DF_IMAGE_U8} only if both source images are \texttt{VX_DF_IMAGE_U8} and the output image is explicitly set to \texttt{VX_DF_IMAGE_U8}. It is otherwise \texttt{VX_DF_IMAGE_S16}. If one of the input images is of type \texttt{VX_DF_IMAGE_S16}, all values are converted to \texttt{VX_DF_IMAGE_S16}. The overflow handling is controlled by an overflow-policy parameter. For each pixel value in the two input images:

\[
    out(x,y) = in_1(x,y) + in_2(x,y)
\]

Functions

- \texttt{vx_node VX_API_CALL vxAddNode (vx_graph graph, vx_image in1, vx_image in2, vx_enum policy, vx_image out)}
  [Graph] Creates an arithmetic addition node.

- \texttt{vx_status VX_API_CALL vxuAdd (vx_context context, vx_image in1, vx_image in2, vx_enum policy, vx_image out)}
  [Immediate] Performs arithmetic addition on pixel values in the input images.

3.8.2 Function Documentation

\texttt{vx_node VX_API_CALL vxAddNode ( vx_graph graph, vx_image in1, vx_image in2, vx_enum policy, vx_image out )}

[Graph] Creates an arithmetic addition node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>in1</td>
<td>An input image, \texttt{VX_DF_IMAGE_U8} or \texttt{VX_DF_IMAGE_S16}.</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>An input image, \texttt{VX_DF_IMAGE_U8} or \texttt{VX_DF_IMAGE_S16}.</td>
</tr>
<tr>
<td>in</td>
<td>policy</td>
<td>A \texttt{VX_TYPE_ENUM} of the \texttt{vx_convert_policy_e} enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The output image, a \texttt{VX_DF_IMAGE_U8} or \texttt{VX_DF_IMAGE_S16} image, which must have the same dimensions as the input images.</td>
</tr>
</tbody>
</table>

Returns

\texttt{vx_node}.

Return values

| \texttt{vx_node} | A node reference. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus}. |

\texttt{vx_status VX_API_CALL vxuAdd ( vx_context context, vx_image in1, vx_image in2, vx_enum policy, vx_image out )}

[Immediate] Performs arithmetic addition on pixel values in the input images.

Parameters

| in    | context | The reference to the overall context. |
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in1</code></td>
<td>A <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> input image.</td>
</tr>
<tr>
<td><code>in2</code></td>
<td>A <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> input image.</td>
</tr>
<tr>
<td><code>policy</code></td>
<td>A <code>vx_convert_policy_e</code> enumeration.</td>
</tr>
<tr>
<td><code>out</code></td>
<td>The output image in <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> format.</td>
</tr>
</tbody>
</table>

### Returns

A `vx_status_e` enumeration.

### Return values

<table>
<thead>
<tr>
<th><code>VX_SUCCESS</code></th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.9 Arithmetic Subtraction

3.9.1 Detailed Description

Performs subtraction between two images. The output image dimensions should be the same as the dimensions of the input images.

Arithmetic subtraction is performed between the pixel values in two VX_DF_IMAGE_U8 or two VX_DF_IMAGE_S16 images. The output image can be VX_DF_IMAGE_U8 only if both source images are VX_DF_IMAGE_U8 and the output image is explicitly set to VX_DF_IMAGE_U8. It is otherwise VX_DF_IMAGE_S16. If one of the input images is of type VX_DF_IMAGE_S16, all values are converted to VX_DF_IMAGE_S16. The overflow handling is controlled by an overflow-policy parameter. For each pixel value in the two input images:

\[ out(x,y) = in_1(x,y) - in_2(x,y) \]

Functions

- **vx_node VX_API_CALL vxSubtractNode (vx_graph graph, vx_image in1, vx_image in2, vx_enum policy, vx_image out)**
  
  [Graph] Creates an arithmetic subtraction node.

- **vx_status VX_API_CALL vxuSubtract (vx_context context, vx_image in1, vx_image in2, vx_enum policy, vx_image out)**
  
  [Immediate] Performs arithmetic subtraction on pixel values in the input images.

3.9.2 Function Documentation

**vx_node VX_API_CALL vxSubtractNode (vx_graph graph, vx_image in1, vx_image in2, vx_enum policy, vx_image out)**

[Graph] Creates an arithmetic subtraction node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>in1</td>
<td>An input image, VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16, the minuend.</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>An input image, VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16, the subtrahend.</td>
</tr>
<tr>
<td>in</td>
<td>policy</td>
<td>A VX_TYPE_ENUM of the vx_convert_policy_e enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The output image, a VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 image, which must have the same dimensions as the input images.</td>
</tr>
</tbody>
</table>

Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

**vx_status VX_API_CALL vxuSubtract (vx_context context, vx_image in1, vx_image in2, vx_enum policy, vx_image out)**

[Immediate] Performs arithmetic subtraction on pixel values in the input images.

Parameters

| in  | context | The reference to the overall context. |
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>in1</th>
<th>A VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 input image, the minuend.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>in2</td>
<td>A VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 input image, the subtrahend.</td>
</tr>
<tr>
<td>in</td>
<td>policy</td>
<td>A vx_convert_policy_e enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The output image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.10 Bitwise AND

3.10.1 Detailed Description
Performs a bitwise AND operation between two VX_DF_IMAGE_U8 images. The output image dimensions should be the same as the dimensions of the input images.

Bitwise AND is computed by the following, for each bit in each pixel in the input images:

\[
out(x,y) = in_1(x,y) \land in_2(x,y)
\]

Or expressed as C code:

\[
out(x,y) = in_1(x,y) \& in_2(x,y)
\]

Functions
- **vx_node VX_API_CALL vxAndNode (vx_graph graph, vx_image in1, vx_image in2, vx_image out)**
  [Graph] Creates a bitwise AND node.

- **vx_status VX_API_CALL vxuAnd (vx_context context, vx_image in1, vx_image in2, vx_image out)**
  [Immediate] Computes the bitwise and between two images.

3.10.2 Function Documentation

**vx_node VX_API_CALL vxAndNode ( vx_graph graph, vx_image in1, vx_image in2, vx_image out )**

[Graph] Creates a bitwise AND node.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph</td>
<td>vx_graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in1</td>
<td>vx_image</td>
<td>A VX_DF_IMAGE_U8 input image.</td>
</tr>
<tr>
<td>in2</td>
<td>vx_image</td>
<td>A VX_DF_IMAGE_U8 input image.</td>
</tr>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The VX_DF_IMAGE_U8 output image, which must have the same dimensions as the input images.</td>
</tr>
</tbody>
</table>

Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

**vx_status VX_API_CALL vxuAnd ( vx_context context, vx_image in1, vx_image in2, vx_image out )**

[Immediate] Computes the bitwise and between two images.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>vx_context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>in1</td>
<td>vx_image</td>
<td>A VX_DF_IMAGE_U8 input image</td>
</tr>
<tr>
<td>in2</td>
<td>vx_image</td>
<td>A VX_DF_IMAGE_U8 input image</td>
</tr>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The VX_DF_IMAGE_U8 output image.</td>
</tr>
</tbody>
</table>
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.11 Bitwise EXCLUSIVE OR

3.11.1 Detailed Description

Performs a bitwise EXCLUSIVE OR (XOR) operation between two VX_DF_IMAGE_U8 images. The output image dimensions should be the same as the dimensions of the input images.

Bitwise XOR is computed by the following, for each bit in each pixel in the input images:

\[ out(x, y) = in_1(x, y) \oplus in_2(x, y) \]

Or expressed as C code:

\[ out(x, y) = in_1(x, y) \oplus in_2(x, y) \]

Functions

- **vx_status VX_API_CALL vxuXor (vx_context context, vx_image in1, vx_image in2, vx_image out)**
  [Immediate] Computes the bitwise exclusive-or between two images.

- **vx_node VX_API_CALL vxXorNode (vx_graph graph, vx_image in1, vx_image in2, vx_image out)**
  [Graph] Creates a bitwise EXCLUSIVE OR node.

3.11.2 Function Documentation

**vx_node VX_API_CALL vxXorNode ( vx_graph graph, vx_image in1, vx_image in2, vx_image out )**

[Graph] Creates a bitwise EXCLUSIVE OR node.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td><strong>in</strong> in1</td>
<td>A VX_DF_IMAGE_U8 input image.</td>
</tr>
<tr>
<td><strong>in</strong> in2</td>
<td>A VX_DF_IMAGE_U8 input image.</td>
</tr>
<tr>
<td><strong>out</strong> out</td>
<td>The VX_DF_IMAGE_U8 output image, which must have the same dimensions as the input images.</td>
</tr>
</tbody>
</table>

Returns

**vx_node.**

Return values

| **vx_node** | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

**vx_status VX_API_CALL vxuXor ( vx_context context, vx_image in1, vx_image in2, vx_image out )**

[Immediate] Computes the bitwise exclusive-or between two images.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td><strong>in</strong> in1</td>
<td>A VX_DF_IMAGE_U8 input image</td>
</tr>
<tr>
<td><strong>in</strong> in2</td>
<td>A VX_DF_IMAGE_U8 input image</td>
</tr>
<tr>
<td><strong>out</strong> out</td>
<td>The VX_DF_IMAGE_U8 output image.</td>
</tr>
</tbody>
</table>
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.12 Bitwise INCLUSIVE OR

3.12.1 Detailed Description

Performs a bitwise INCLUSIVE OR operation between two VX_DF_IMAGE_U8 images. The output image dimensions should be the same as the dimensions of the input images.

Bitwise INCLUSIVE OR is computed by the following, for each bit in each pixel in the input images:

\[ \text{out}(x,y) = \text{in}_1(x,y) \lor \text{in}_2(x,y) \]

Or expressed as C code:

\[ \text{out}(x,y) = \text{in}_1(x,y) | \text{in}_2(x,y) \]

Functions

- **vx_node VX_API_CALL vxOrNode (vx_graph graph, vx_image in1, vx_image in2, vx_image out)**
  
  [Graph] Creates a bitwise INCLUSIVE OR node.

- **vx_status VX_API_CALL vxuOr (vx_context context, vx_image in1, vx_image in2, vx_image out)**
  
  [Immediate] Computes the bitwise inclusive-or between two images.

3.12.2 Function Documentation

**vx_node VX_API_CALL vxOrNode ( vx_graph graph, vx_image in1, vx_image in2, vx_image out )**

[Graph] Creates a bitwise INCLUSIVE OR node.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td><strong>in</strong> in1</td>
<td>A VX_DF_IMAGE_U8 input image.</td>
</tr>
<tr>
<td><strong>in</strong> in2</td>
<td>A VX_DF_IMAGE_U8 input image.</td>
</tr>
<tr>
<td><strong>out</strong> out</td>
<td>The VX_DF_IMAGE_U8 output image, which must have the same dimensions as the input images.</td>
</tr>
</tbody>
</table>

Returns

**vx_node**.

Return values

| **vx_node** | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

**vx_status VX_API_CALL vxuOr ( vx_context context, vx_image in1, vx_image in2, vx_image out )**

[Immediate] Computes the bitwise inclusive-or between two images.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td><strong>in</strong> in1</td>
<td>A VX_DF_IMAGE_U8 input image</td>
</tr>
<tr>
<td><strong>in</strong> in2</td>
<td>A VX_DF_IMAGE_U8 input image</td>
</tr>
<tr>
<td><strong>out</strong> out</td>
<td>The VX_DF_IMAGE_U8 output image.</td>
</tr>
</tbody>
</table>
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong><code>*</code></strong></td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.13 Bitwise NOT

3.13.1 Detailed Description

Performs a bitwise NOT operation on a VX_DF_IMAGE_U8 input image. The output image dimensions should be the same as the dimensions of the input image.

Bitwise NOT is computed by the following, for each bit in each pixel in the input image:

\[ \text{out}(x, y) = \overline{\text{in}(x, y)} \]

Or expressed as C code:

\[ \text{out}(x, y) = \neg \text{in}_1(x, y) \]

Functions

- **vx_node VX_API_CALL vxNotNode (vx_graph graph, vx_image input, vx_image output)**
  - [Graph] Creates a bitwise NOT node.

- **vx_status VX_API_CALL vxuNot (vx_context context, vx_image input, vx_image output)**
  - [Immediate] Computes the bitwise not of an image.

3.13.2 Function Documentation

**vx_node VX_API_CALL vxNotNode (vx_graph graph, vx_image input, vx_image output)**

[Graph] Creates a bitwise NOT node.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in input</td>
<td>A VX_DF_IMAGE_U8 input image.</td>
</tr>
<tr>
<td>out output</td>
<td>The VX_DF_IMAGE_U8 output image, which must have the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

Returns

* vx_node.

Return values

<table>
<thead>
<tr>
<th>Return type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_node</td>
<td>A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus</td>
</tr>
</tbody>
</table>

**vx_status VX_API_CALL vxuNot (vx_context context, vx_image input, vx_image output)**

[Immediate] Computes the bitwise not of an image.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>in input</td>
<td>The VX_DF_IMAGE_U8 input image</td>
</tr>
<tr>
<td>out output</td>
<td>The VX_DF_IMAGE_U8 output image.</td>
</tr>
</tbody>
</table>
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th><strong>VX_SUCCESS</strong></th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.14 Box Filter

3.14.1 Detailed Description

Computes a Box filter over a window of the input image. The output image dimensions should be the same as the dimensions of the input image.

This filter uses the following convolution matrix:

\[
K_{\text{box}} = \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix} \ast \frac{1}{9}
\]

Functions

- \texttt{vx_node VX\_API\_CALL vxBox3x3Node (vx\_graph graph, vx\_image input, vx\_image output)}
  \hspace{1cm} [Graph] Creates a Box Filter Node.
- \texttt{vx\_status VX\_API\_CALL vxuBox3x3 (vx\_context context, vx\_image input, vx\_image output)}
  \hspace{1cm} [Immediate] Computes a box filter on the image by a 3x3 window.

3.14.2 Function Documentation

\texttt{vx\_node VX\_API\_CALL vxBox3x3Node ( vx\_graph graph, vx\_image input, vx\_image output )}

[Graph] Creates a Box Filter Node.

Parameters

| in | graph | The reference to the graph. |
| in | input | The input image in \texttt{VX\_DF\_IMAGE\_U8} format. |
| out | output | The output image in \texttt{VX\_DF\_IMAGE\_U8} format, which must have the same dimensions as the input image. |

Returns

\texttt{vx\_node}.

Return values

| \texttt{vx\_node} | A node reference. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus} |

\texttt{vx\_status VX\_API\_CALL vxuBox3x3 ( vx\_context context, vx\_image input, vx\_image output )}

[Immediate] Computes a box filter on the image by a 3x3 window.

Parameters

| in | context | The reference to the overall context. |
| in | input | The input image in \texttt{VX\_DF\_IMAGE\_U8} format. |
| out | output | The output image in \texttt{VX\_DF\_IMAGE\_U8} format. |

Returns

A \texttt{vx\_status\_e} enumeration.
Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <strong>vx_status_e</strong>.</td>
</tr>
</tbody>
</table>
3.15 Non-Maxima Suppression

3.15.1 Detailed Description

Find local maxima in an image, or otherwise suppress pixels that are not local maxima.

The input to the Non-Maxima Suppressor is either a VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 image. In the case of a VX_DF_IMAGE_S16 image, suppressed pixels shall take the value of INT16_MIN.

An optional mask image may be used to restrict the suppression to a region-of-interest. If a mask pixel is non-zero, then the associated pixel in the input is completely ignored and not considered during suppression; that is, it is not suppressed and not considered as part of any suppression window.

A pixel with coordinates \((x, y)\) is kept if and only if it is greater than or equal to its top left neighbours; and greater than its bottom right neighbours. For example, for a window size of 3, \(P(x, y)\) is retained if the following condition holds:

\[
P(x, y) \geq P(x-1, y-1) \quad \text{and} \quad P(x, y) \geq P(x, y-1) \\
P(x, y) \geq P(x+1, y-1) \quad \text{and} \quad P(x, y) \geq P(x-1, y) \\
P(x, y) > P(x+1, y) \quad \text{and} \quad P(x, y) > P(x-1, y+1) \\
P(x, y) > P(x, y+1) \quad \text{and} \quad P(x, y) > P(x+1, y+1)
\]

Functions

- `vx_node VX_API_CALL vxNonMaxSuppressionNode (vx_graph graph, vx_image input, vx_image mask, vx_int32 win_size, vx_image output)`
  [Graph] Creates a Non-Maxima Suppression node.

- `vx_status VX_API_CALL vxuNonMaxSuppression (vx_context context, vx_image input, vx_image mask, vx_int32 win_size, vx_image output)`
  [Immediate] Performs Non-Maxima Suppression on an image, producing an image of the same type.

3.15.2 Function Documentation

`vx_node VX_API_CALL vxNonMaxSuppressionNode (vx_graph graph, vx_image input, vx_image mask, vx_int32 win_size, vx_image output)`

[Graph] Creates a Non-Maxima Suppression node.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>graph</code></td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td><code>input</code></td>
<td>The input image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td><code>mask</code></td>
<td>[optional] Constrict suppression to a ROI. The mask image is of type VX_DF_IMAGE_U8 and must be the same dimensions as the input image.</td>
</tr>
<tr>
<td><code>win_size</code></td>
<td>The size of window over which to perform the localized non-maxima suppression. Must be odd, and less than or equal to the smallest dimension of the input image.</td>
</tr>
<tr>
<td><code>output</code></td>
<td>The output image, of the same type and size as the input, that has been non-maxima suppressed.</td>
</tr>
</tbody>
</table>

Returns

`vx_node`.

Return values

| `vx_node` | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |
vx_status VX_API_CALL vxuNonMaxSuppression ( vx_context context, vx_image input, vx_image mask, vx_int32 win_size, vx_image output )

[Immediate] Performs Non-Maxima Suppression on an image, producing an image of the same type.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td>in</td>
<td>mask</td>
<td>[optional] Constrict suppression to a ROI. The mask image is of type VX_DF_IMAGE_U8 and must be the same dimensions as the input image.</td>
</tr>
<tr>
<td>in</td>
<td>win_size</td>
<td>The size of window over which to perform the localized non-maxima suppression. Must be odd, and less than or equal to the smallest dimension of the input image.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image, of the same type as the input, that has been non-maxima suppressed.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>* An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.16 Canny Edge Detector

3.16.1 Detailed Description

Provides a Canny edge detector kernel. The output image dimensions should be the same as the dimensions of the input image.

This function implements an edge detection algorithm similar to that described in [7]. The main components of the algorithm are:

- Gradient magnitude and orientation computation using a noise resistant operator (Sobel).
- Non-maximum suppression of the gradient magnitude, using the gradient orientation information.
- Tracing edges in the modified gradient image using hysteresis thresholding to produce a binary result.

The details of each of these steps are described below.

- **Gradient Computation:** Conceptually, the input image is convolved with vertical and horizontal Sobel kernels of the size indicated by the gradient_size parameter. The Sobel kernels used for the gradient computation shall be as shown below. The two resulting directional gradient images (\(dx\) and \(dy\)) are then used to compute a gradient magnitude image and a gradient orientation image. The norm used to compute the gradient magnitude is indicated by the norm_type parameter, so the magnitude may be \(|dx| + |dy|\) for VX_NORM_L1 or \(\sqrt{dx^2 + dy^2}\) for VX_NORM_L2. The gradient orientation image is quantized into 4 values: 0, 45, 90, and 135 degrees.

- For gradient size 3:

  \[
  \begin{pmatrix}
  -1 & 0 & 1 \\
  -2 & 0 & 2 \\
  -1 & 0 & 1 \\
  \end{pmatrix}
  \]

  \[
  \text{sobel}_x = \text{transpose(sobel}_y) = \begin{pmatrix}
  -1 & 0 & -2 \\
  0 & 0 & 0 \\
  1 & 2 & 1 \\
  \end{pmatrix}
  \]

- For gradient size 5:

  \[
  \begin{pmatrix}
  -1 & -2 & 0 & 2 & 1 \\
  -4 & -8 & 0 & 8 & 4 \\
  -4 & -8 & 0 & 8 & 4 \\
  -1 & -2 & 0 & 2 & 1 \\
  \end{pmatrix}
  \]

  \[
  \text{sobel}_x = \text{transpose(sobel}_y) = \begin{pmatrix}
  -1 & -4 & -5 & 0 & 5 & 4 & 1 \\
  -6 & -24 & -30 & 0 & 30 & 24 & 6 \\
  -15 & -60 & -75 & 0 & 75 & 60 & 15 \\
  -20 & -80 & -100 & 0 & 100 & 80 & 20 \\
  -15 & -60 & -75 & 0 & 75 & 60 & 15 \\
  -6 & -24 & -30 & 0 & 30 & 24 & 6 \\
  -1 & -4 & -5 & 0 & 5 & 4 & 1 \\
  \end{pmatrix}
  \]

- For gradient size 7:

  \[
  \begin{pmatrix}
  -1 & -4 & -5 & 0 & 5 & 4 & 1 \\
  -6 & -24 & -30 & 0 & 30 & 24 & 6 \\
  -15 & -60 & -75 & 0 & 75 & 60 & 15 \\
  -20 & -80 & -100 & 0 & 100 & 80 & 20 \\
  -15 & -60 & -75 & 0 & 75 & 60 & 15 \\
  -6 & -24 & -30 & 0 & 30 & 24 & 6 \\
  -1 & -4 & -5 & 0 & 5 & 4 & 1 \\
  \end{pmatrix}
  \]

  \[
  \text{sobel}_x = \text{transpose(sobel}_y) = \begin{pmatrix}
  -1 & -4 & -5 & 0 & 5 & 4 & 1 \\
  -6 & -24 & -30 & 0 & 30 & 24 & 6 \\
  -15 & -60 & -75 & 0 & 75 & 60 & 15 \\
  -20 & -80 & -100 & 0 & 100 & 80 & 20 \\
  -15 & -60 & -75 & 0 & 75 & 60 & 15 \\
  -6 & -24 & -30 & 0 & 30 & 24 & 6 \\
  -1 & -4 & -5 & 0 & 5 & 4 & 1 \\
  \end{pmatrix}
  \]

- **Non-Maximum Suppression:** This is then applied such that a pixel is retained as a potential edge pixel if and only if its magnitude is greater than or equal to the pixels in the direction perpendicular to its edge orientation. For example, if the pixel's orientation is 0 degrees, it is only retained if its gradient magnitude is larger than that of the pixels at 90 and 270 degrees to it. If a pixel is suppressed via this condition, it must not appear as an edge pixel in the final output, i.e., its value must be 0 in the final output.
• **Edge Tracing:** The final edge pixels in the output are identified via a double thresholded hysteresis procedure. All retained pixels with magnitude above the high threshold are marked as known edge pixels (valued 255) in the final output image. All pixels with magnitudes less than or equal to the low threshold must not be marked as edge pixels in the final output. For the pixels in between the thresholds, edges are traced and marked as edges (255) in the output. This can be done by starting at the known edge pixels and moving in all eight directions recursively until the gradient magnitude is less than or equal to the low threshold.

• **Caveats:** The intermediate results described above are conceptual only; so for example, the implementation may not actually construct the gradient images and non-maximum-suppressed images. Only the final binary (0 or 255 valued) output image must be computed so that it matches the result of a final image constructed as described above.

### Enumerations

- **enum vx_norm_type_e:**
  
  ```c
  enum vx_norm_type_e {
    VX_NORM_L1 = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_NORM_TYPE << 12)) + 0x0,
    VX_NORM_L2 = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_NORM_TYPE << 12)) + 0x1
  }  
  
  A normalization type.
  ```

### Functions

- **vx_node VX_API_CALL vxCannyEdgeDetectorNode (vx_graph graph, vx_image input, vx_threshold hyst, vx_int32 gradient_size, vx_enum norm_type, vx_image output)**

  [Graph] Creates a Canny Edge Detection Node.

- **vx_status VX_API_CALL vxuCannyEdgeDetector (vx_context context, vx_image input, vx_threshold hyst, vx_int32 gradient_size, vx_enum norm_type, vx_image output)**

  [Immediate] Computes Canny Edges on the input image into the output image.

#### 3.16.2 Enumeration Type Documentation

- **enum vx_norm_type_e:**

  A normalization type.

  See also

  - Canny Edge Detector

  Enumerator

  
  ```c
  VX_NORM_L1 The L1 normalization.
  VX_NORM_L2 The L2 normalization.
  ```

  Definition at line 1378 of file vx_types.h.

#### 3.16.3 Function Documentation

- **vx_node VX_API_CALL vxCannyEdgeDetectorNode (vx_graph graph, vx_image input, vx_threshold hyst, vx_int32 gradient_size, vx_enum norm_type, vx_image output )**

  [Graph] Creates a Canny Edge Detection Node.

  **Parameters**

  - **in graph** The reference to the graph.
  - **in input** The input VX_DF_IMAGE_U8 image.
  - **in hyst** The double threshold for hysteresis. The VX_THRESHOLD_INPUT_FORMAT shall be either VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16. The VX_THRESHOLD_OUTPUT_FORMAT is ignored.
  - **in gradient_size** The size of the Sobel filter window, must support at least 3, 5, and 7.
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>norm_type</th>
<th>A flag indicating the norm used to compute the gradient, VX_NORM_L1 or VX_NORM_L2.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in VX_DF_IMAGE_U8 format with values either 0 or 255.</td>
</tr>
</tbody>
</table>

### Returns

vx_node.

### Return values

<table>
<thead>
<tr>
<th>vx_node</th>
<th>A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus.</th>
</tr>
</thead>
</table>

```c
vx_status VX_API_CALL vxuCannyEdgeDetector ( vx_context context, vx_image input, vx_threshold hyst, vx_int32 gradient_size, vx_enum norm_type, vx_image output )
```

[Immediate] Computes Canny Edges on the input image into the output image.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input VX_DF_IMAGE_U8 image.</td>
</tr>
<tr>
<td>in</td>
<td>hyst</td>
<td>The double threshold for hysteresis. The VX_THRESHOLD_INPUT_FORMAT shall be either VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16. The VX_THRESHOLD_OUTPUT_FORMAT is ignored.</td>
</tr>
<tr>
<td>in</td>
<td>gradient_size</td>
<td>The size of the Sobel filter window, must support at least 3, 5 and 7.</td>
</tr>
<tr>
<td>in</td>
<td>norm_type</td>
<td>A flag indicating the norm used to compute the gradient, VX_NORM_L1 or VX_NORM_L2.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in VX_DF_IMAGE_U8 format with values either 0 or 255.</td>
</tr>
</tbody>
</table>

### Returns

A vx_status_e enumeration.

### Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>


3.17 Channel Combine

3.17.1 Detailed Description

Implements the Channel Combine Kernel.

This kernel takes multiple VX_DF_IMAGE_U8 planes to recombine them into a multi-planar or interleaved format from vx_df_image_e. The user must specify only the number of channels that are appropriate for the combining operation. If a user specifies more channels than necessary, the operation results in an error. For the case where the destination image is a format with subsampling, the input channels are expected to have been subsampled before combining (by stretching and resizing).

Functions

- **vx_node VX_API_CALL vxChannelCombineNode (vx_graph graph, vx_image plane0, vx_image plane1, vx_image plane2, vx_image plane3, vx_image output)**
  
  [Graph] Creates a channel combine node.

- **vx_status VX_API_CALL vxuChannelCombine (vx_context context, vx_image plane0, vx_image plane1, vx_image plane2, vx_image plane3, vx_image output)**
  
  [Immediate] Invokes an immediate Channel Combine.

3.17.2 Function Documentation

**vx_node VX_API_CALL vxChannelCombineNode (vx_graph graph, vx_image plane0, vx_image plane1, vx_image plane2, vx_image plane3, vx_image output)**

[Graph] Creates a channel combine node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The graph reference.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>plane0</td>
<td>The plane that forms channel 0. Must be VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>in</td>
<td>plane1</td>
<td>The plane that forms channel 1. Must be VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>in</td>
<td>plane2</td>
<td>[optional] The plane that forms channel 2. Must be VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image. The format of the image must be defined, even if the image is virtual. Must have the same dimensions as the input images</td>
</tr>
</tbody>
</table>

See also

VX_KERNEL_CHANNEL_COMBINE

Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

**vx_status VX_API_CALL vxuChannelCombine (vx_context context, vx_image plane0, vx_image plane1, vx_image plane2, vx_image plane3, vx_image output)**

[Immediate] Invokes an immediate Channel Combine.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>plane0</td>
<td>The plane that forms channel 0. Must be VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>in</td>
<td>plane1</td>
<td>The plane that forms channel 1. Must be VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>in</td>
<td>plane2</td>
<td>[optional] The plane that forms channel 2. Must be VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.18 Channel Extract

3.18.1 Detailed Description

Implements the Channel Extraction Kernel. This kernel removes a single VX_DF_IMAGE_U8 channel (plane) from a multi-planar or interleaved image format from vx_df_image_e.

Functions

- **vx_node VX_API_CALL vxChannelExtractNode (vx_graph graph, vx_image input, vx_enum channel, vx_image output)**
  
  [Graph] Creates a channel extract node.

- **vx_status VX_API_CALL vxuChannelExtract (vx_context context, vx_image input, vx_enum channel, vx_image output)**
  
  [Immediate] Invokes an immediate Channel Extract.

3.18.2 Function Documentation

**vx_node VX_API_CALL vxChannelExtractNode ( vx_graph graph, vx_image input, vx_enum channel, vx_image output )**

[Graph] Creates a channel extract node.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in input</td>
<td>The input image. Must be one of the defined vx_df_image_e multi-channel formats.</td>
</tr>
<tr>
<td>in channel</td>
<td>The vx_channel_e channel to extract.</td>
</tr>
<tr>
<td>out output</td>
<td>The output image. Must be VX_DF_IMAGE_U8, and must have the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

See also

- VX_KERNEL_CHANNEL_EXTRACT

Returns

- **vx_node**

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

**vx_status VX_API_CALL vxuChannelExtract ( vx_context context, vx_image input, vx_enum channel, vx_image output )**

[Immediate] Invokes an immediate Channel Extract.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>in input</td>
<td>The input image. Must be one of the defined vx_df_image_e multi-channel formats.</td>
</tr>
<tr>
<td>in channel</td>
<td>The vx_channel_e enumeration to extract.</td>
</tr>
<tr>
<td>out output</td>
<td>The output image. Must be VX_DF_IMAGE_U8.</td>
</tr>
</tbody>
</table>
Returns

A *vx_status_e* enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <em>vx_status_e</em>.</td>
</tr>
</tbody>
</table>
3.19 Color Convert

3.19.1 Detailed Description

Implements the Color Conversion Kernel. The output image dimensions should be the same as the dimensions of the input image.

This kernel converts an image of a designated `vx_df_image_e` format to another `vx_df_image_e` format for those combinations listed in the below table, where the columns are output types and the rows are input types. The API version first supporting the conversion is also listed.

<table>
<thead>
<tr>
<th>I/O</th>
<th>RGB</th>
<th>RGBX</th>
<th>NV12</th>
<th>NV21</th>
<th>UYVY</th>
<th>YUYV</th>
<th>IYUV</th>
<th>YUV4</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>RGBX</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>NV12</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>NV21</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>UYVY</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>YUYV</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>IYUV</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>YUV4</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The `vx_df_image_e` encoding, held in the `VX_IMAGE_FORMAT` attribute, describes the data layout. The interpretation of the colors is determined by the `VX_IMAGE_SPACE` (see `vx_color_space_e`) and `VX_IMAGE_RANGE` (see `vx_channel_range_e`) attributes of the image. OpenVX 1.1 implementations are required only to support images of `VX_COLOR_SPACE_BT709` and `VX_CHANNEL_RANGE_FULL`.

If the channel range is defined as `VX_CHANNEL_RANGE_FULL`, the conversion between the real number and integer quantizations of color channels is defined for red, green, blue, and Y as:

\[
value_{\text{real}} = \frac{value_{\text{integer}}}{256.0}
\]

\[
value_{\text{integer}} = \max(0, \min(255, \text{floor}(value_{\text{real}} \times 256.0)))
\]

For the U and V channels, the conversion between real number and integer quantizations is:

\[
value_{\text{real}} = \frac{(value_{\text{integer}} - 128.0)}{256.0}
\]

\[
value_{\text{integer}} = \max(0, \min(255, \text{floor}((value_{\text{real}} \times 256.0) + 128)))
\]

If the channel range is defined as `VX_CHANNEL_RANGE_RESTRICTED`, the conversion between the integer quantizations of color channels and the continuous representations is defined for red, green, blue, and Y as:

\[
value_{\text{real}} = \frac{(value_{\text{integer}} - 16.0)}{219.0}
\]

\[
value_{\text{integer}} = \max(0, \min(255, \text{floor}((value_{\text{real}} \times 219.0) + 16.5)))
\]

For the U and V channels, the conversion between real number and integer quantizations is:

\[
value_{\text{real}} = \frac{(value_{\text{integer}} - 128.0)}{224.0}
\]

\[
value_{\text{integer}} = \max(0, \min(255, \text{floor}((value_{\text{real}} \times 224.0) + 128.5)))
\]

The conversions between nonlinear-intensity Y'PbPr and R'G'B' real numbers are:

\[
R' = Y' + 2(1 - K_r)P_r
\]

\[
B' = Y' + 2(1 - K_b)P_b
\]

\[
G' = Y' - \frac{2(K_r(1 - K_r)P_r + K_b(1 - K_b)P_b)}{1 - K_r - K_b}
\]

\[
Y' = (K_r \times R') + (K_b \times B') + (1 - K_r - K_b)G'
\]
coordinates into the new RGB space. Working with linear RGB values, the conversion formulae are:

$$P_b = \frac{B'}{2} - \frac{(R' \ast K_r) + G'(1 - K_r - K_b)}{2(1 - K_b)}$$

$$P_r = \frac{R'}{2} - \frac{(B' \ast K_r) + G'(1 - K_r - K_b)}{2(1 - K_r)}$$

The means of reconstructing $P_b$ and $P_r$ values from chroma-downsampled formats is implementation-defined.

In `VX_COLOR_SPACE_BT601_525` or `VX_COLOR_SPACE_BT601_625`:

$$K_r = 0.299$$

$$K_b = 0.114$$

In `VX_COLOR_SPACE_BT709`:

$$K_r = 0.2126$$

$$K_b = 0.0722$$

In all cases, for the purposes of conversion, these colour representations are interpreted as nonlinear in intensity, as defined by the BT.601, BT.709, and sRGB specifications. That is, the encoded colour channels are nonlinear $R'$, $G'$ and $B'$, $Y'$, $P_b$, and $P_r$.

Each channel of the R'G'B' representation can be converted to and from a linear-intensity RGB channel by these formulae:

$$value_{nonlinear} = 1.099 \ast value_{linear}^{0.45} - 0.099 \quad for \quad 1 \geq value_{linear} \geq 0.018$$

$$value_{nonlinear} = 4.500 \ast value_{linear} \quad for \quad 0.018 > value_{linear} \geq 0$$

$$value_{linear} = \left(\frac{value_{nonlinear} + 0.099}{1.099}\right)^{4.5} \quad for \quad 1 \geq value_{nonlinear} > 0.081$$

$$value_{linear} = \frac{value_{nonlinear}}{4.5} \quad for \quad 0.081 \geq value_{nonlinear} \geq 0$$

As the different color spaces have different RGB primaries, a conversion between them must transform the color coordinates into the new RGB space. Working with linear RGB values, the conversion formulae are:

$$R_{BT601_525} = R_{BT601_625} \ast 1.112302 + G_{BT601_625} \ast -0.102441 + B_{BT601_625} \ast -0.009860$$

$$G_{BT601_525} = R_{BT601_625} \ast -0.020497 + G_{BT601_625} \ast 1.037030 + B_{BT601_625} \ast -0.016533$$

$$B_{BT601_525} = R_{BT601_625} \ast 0.001704 + G_{BT601_625} \ast 0.016063 + B_{BT601_625} \ast 0.982233$$

$$R_{BT601_625} = R_{BT709} \ast 1.065379 + G_{BT709} \ast -0.055401 + B_{BT709} \ast -0.009978$$

$$G_{BT601_525} = R_{BT709} \ast -0.019633 + G_{BT709} \ast 1.036363 + B_{BT709} \ast -0.016731$$

$$B_{BT601_525} = R_{BT709} \ast 0.001632 + G_{BT709} \ast 0.004412 + B_{BT709} \ast 0.993956$$

$$R_{BT601_625} = R_{BT601_525} \ast 0.900657 + G_{BT601_525} \ast 0.088807 + B_{BT601_525} \ast 0.010536$$

$$G_{BT601_525} = R_{BT601_525} \ast 0.017772 + G_{BT601_525} \ast 0.965793 + B_{BT601_525} \ast 0.016435$$

$$B_{BT601_525} = R_{BT601_525} \ast -0.001853 + G_{BT601_525} \ast -0.015948 + B_{BT601_525} \ast 1.017801$$

$$R_{BT601_625} = R_{BT709} \ast 0.957815 + G_{BT709} \ast 0.042185$$

$$G_{BT601_625} = G_{BT709}$$

$$B_{BT601_625} = G_{BT709} \ast -0.011934 + B_{BT709} \ast 1.011934$$

$$R_{BT709} = R_{BT601_525} \ast 0.939542 + G_{BT601_525} \ast 0.050181 + B_{BT601_525} \ast 0.010277$$

$$G_{BT709} = R_{BT601_525} \ast 0.017772 + G_{BT601_525} \ast 0.965793 + B_{BT601_525} \ast 0.016435$$

$$B_{BT709} = R_{BT601_525} \ast -0.001622 + G_{BT601_525} \ast -0.004370 + B_{BT601_525} \ast 1.005991$$
A conversion between one YUV color space and another may therefore consist of the following transformations:

1. Convert quantized YCbCr ("YUV") to continuous, nonlinear Y’PbPr.
2. Convert continuous Y’PbPr to continuous, nonlinear R’G’B’.
3. Convert nonlinear R’G’B’ to linear-intensity RGB (gamma-correction).
4. Convert linear RGB from the first color space to linear RGB in the second color space.
5. Convert linear RGB to nonlinear R’G’B’ (gamma-conversion).
6. Convert nonlinear R’G’B’ to YPbPr.
7. Convert continuous YPbPr to quantized Y’CbCr ("YUV").

The above formulae and constants are defined in the ITU BT.601 and BT.709 specifications. The formulae for converting between RGB primaries can be derived from the specified primary chromaticity values and the specified white point by solving for the relative intensity of the primaries.

Functions

- **vx_node VX_API_CALL vxColorConvertNode (vx_graph graph, vx_image input, vx_image output)**
  [Graph] Creates a color conversion node.

- **vx_status VX_API_CALL vxuColorConvert (vx_context context, vx_image input, vx_image output)**
  [Immediate] Invokes an immediate Color Conversion.

### 3.19.2 Function Documentation

**vx_node VX_API_CALL vxColorConvertNode ( vx_graph graph, vx_image input, vx_image output )**

[Graph] Creates a color conversion node.

#### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>graph</strong></td>
</tr>
<tr>
<td>in</td>
<td><strong>input</strong></td>
</tr>
<tr>
<td>out</td>
<td><strong>output</strong></td>
</tr>
</tbody>
</table>

See also

**VX_KERNEL_COLOR_CONVERT**

Returns

**vx_node**

Return values

<table>
<thead>
<tr>
<th><strong>vx_node</strong></th>
<th>A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus</th>
</tr>
</thead>
</table>
vx_status VX_API_CALL vxuColorConvert ( vx_context context, vx_image input, vx_image output )

[Immediate] Invokes an immediate Color Conversion.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.20 Convert Bit depth

3.20.1 Detailed Description

Converts image bit depth. The output image dimensions should be the same as the dimensions of the input image.

This kernel converts an image from some source bit-depth to another bit-depth as described by the table below. If the input value is unsigned the shift must be in zeros. If the input value is signed, the shift used must be an arithmetic shift. The columns in the table below are the output types and the rows are the input types. The API version on which conversion is supported is also listed. (An X denotes an invalid operation.)

<table>
<thead>
<tr>
<th>I/O</th>
<th>U8</th>
<th>U16</th>
<th>S16</th>
<th>U32</th>
<th>S32</th>
</tr>
</thead>
<tbody>
<tr>
<td>U8</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U16</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S16</td>
<td>1.0</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U32</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S32</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Conversion Type** The table below identifies the conversion types for the allowed bith depth conversions.

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Conversion Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>U8</td>
<td>S16</td>
<td>Up-conversion</td>
</tr>
<tr>
<td>S16</td>
<td>U8</td>
<td>Down-conversion</td>
</tr>
</tbody>
</table>

**Convert Policy** Down-conversions with VX_CONVERT_POLICY_WRAP follow this equation:

\[
\text{output}(x,y) = \{\text{uint8}(\text{input}(x,y) \gg \text{shift})\};
\]

Down-conversions with VX_CONVERT_POLICY_SATURATE follow this equation:

\[
\text{int16 value} = \text{input}(x,y) \gg \text{shift};
\]

\[
\text{value} = \text{value} < 0 ? 0 : \text{value};
\]

\[
\text{value} = \text{value} > 255 ? 255 : \text{value};
\]

\[
\text{output}(x,y) = \{\text{uint8}\text{value}\};
\]

Up-conversions ignore the policy and perform this operation:

\[
\text{output}(x,y) = \{\{\text{int16}\text{input}(x,y)\} \ll \text{shift}\};
\]

The valid values for 'shift' are as specified below, all other values produce undefined behavior.

\[0 \leq \text{shift} < 8;\]

**Functions**

- **vx_node VX_API_CALL vxConvertDepthNode (vx_graph graph, vx_image input, vx_image output, vx_enum policy, vx_scalar shift)**
  
  [Graph] Creates a bit-depth conversion node.

- **vx_status VX_API_CALL vxuConvertDepth (vx_context context, vx_image input, vx_image output, vx_enum policy, vx_int32 shift)**

  [Immediate] Converts the input images bit-depth into the output image.

3.20.2 Function Documentation

**vx_node VX_API_CALL vxConvertDepthNode ( vx_graph graph, vx_image input, vx_image output, vx_enum policy, vx_scalar shift )**

[Graph] Creates a bit-depth conversion node.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image with the same dimensions of the input image.</td>
</tr>
<tr>
<td>in</td>
<td>policy</td>
<td>A VX_TYPE_ENUM of the vx_convert_policy_e enumeration.</td>
</tr>
<tr>
<td>in</td>
<td>shift</td>
<td>A scalar containing a VX_TYPE_INT32 of the shift value.</td>
</tr>
</tbody>
</table>

Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

vx_status VX_API_CALL vxuConvertDepth ( vx_context context, vx_image input, vx_image output, vx_enum policy, vx_int32 shift )

[Immediate] Converts the input images bit-depth into the output image.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image.</td>
</tr>
<tr>
<td>in</td>
<td>policy</td>
<td>A VX_TYPE_ENUM of the vx_convert_policy_e enumeration.</td>
</tr>
<tr>
<td>in</td>
<td>shift</td>
<td>A scalar containing a VX_TYPE_INT32 of the shift value.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e..</td>
</tr>
</tbody>
</table>
3.21 Custom Convolution

3.21.1 Detailed Description

Convolves the input with the client supplied convolution matrix. The output image dimensions should be the same as the dimensions of the input image.

The client can supply a \( \text{vx}_{\text{int16}} \) typed convolution matrix \( C_{m,n} \). Outputs will be in the \( \text{VX}_{\text{DF}} \_\text{IMAGE} \_\text{S16} \) format unless a \( \text{VX}_{\text{DF}} \_\text{IMAGE} \_\text{U8} \) image is explicitly provided. If values would have been out of range of U8 for \( \text{VX}_{\text{DF}} \_\text{IMAGE} \_\text{U8} \), the values are clamped to 0 or 255.

\[
k_0 = \frac{m}{2} \quad (3.1) \\
l_0 = \frac{n}{2} \quad (3.2) \\
sum = \sum_{k=0,l=0}^{m-1,n-1} \text{input}(x + k_0 - k, y + l_0 - l)C_{k,l} \quad (3.3)
\]

Note

The above equation for this function is different than an equivalent operation suggested by the OpenCV Filter2D function.

This translates into the C declaration:

```c
// A horizontal Scharr gradient operator with different scale.
static vx_int16 gx[3][3] = {
    { 3, 0, -3},
    { 10, 0,-10},
    { 3, 0, -3},
};

vx_convolution scharr_x = vxCreateConvolution(context, 3, 3);
vxCopyConvolutionCoefficients(scharr_x, (vx_int16*)gx, VX_WRITE_ONLY, VX_MEMORY_TYPE_HOST);
vxSetConvolutionAttribute(scharr_x, VX_CONVOLUTION_SCALE, &scale, sizeof(scale));
```

For \( \text{VX}_{\text{DF}} \_\text{IMAGE} \_\text{U8} \) output, an additional step is taken:

\[
\text{output}(x,y) = \begin{cases} 
0 & \text{if } \text{sum} < 0 \\
255 & \text{if } \text{sum}/\text{scale} > 255 \\
\text{sum}/\text{scale} & \text{otherwise}
\end{cases}
\]

For \( \text{VX}_{\text{DF}} \_\text{IMAGE} \_\text{S16} \) output, the summation is simply set to the output

\[
\text{output}(x,y) = \text{sum}/\text{scale}
\]

The overflow policy used is \text{VX}_{\text{CONVERT}_\text{POLICY} \_\text{SATURATE}}.

Functions

- \text{vx}_{\text{node}} \text{ VX}_{\text{API} \_\text{CALL}} \text{ vxConvolveNode} (\text{vx}_{\text{graph}} \text{ graph}, \text{vx}_{\text{image}} \text{ input}, \text{vx}_{\text{convolution}} \text{ conv}, \text{vx}_{\text{image}} \text{ output})

  [Graph] Creates a custom convolution node.

- \text{vx}_{\text{status}} \text{ VX}_{\text{API} \_\text{CALL}} \text{ vxuConvolve} (\text{vx}_{\text{context}} \text{ context}, \text{vx}_{\text{image}} \text{ input}, \text{vx}_{\text{convolution}} \text{ conv}, \text{vx}_{\text{image}} \text{ output})

  [Immediate] Computes a convolution on the input image with the supplied matrix.

3.21.2 Function Documentation

\text{vx}_{\text{node}} \text{ VX}_{\text{API} \_\text{CALL}} \text{ vxConvolveNode} (\text{vx}_{\text{graph}} \text{ graph}, \text{vx}_{\text{image}} \text{ input}, \text{vx}_{\text{convolution}} \text{ conv}, \text{vx}_{\text{image}} \text{ output})

[Graph] Creates a custom convolution node.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>graph</strong></td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>input</strong></td>
<td>The input image in VX_DF_IMAGE_U8 format.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>conv</strong></td>
<td>The vx_int16 convolution matrix.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td><strong>output</strong></td>
<td>The output image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format, which must have the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

Returns

vx_node.

Return values

| **vx_node** | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

vx_status VX_API_CALL vxuConvolve ( vx_context context, vx_image input, vx_convolution conv, vx_image output )

[Immediate] Computes a convolution on the input image with the supplied matrix.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>context</strong></td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>input</strong></td>
<td>The input image in VX_DF_IMAGE_U8 format.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>conv</strong></td>
<td>The vx_int16 convolution matrix.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td><strong>output</strong></td>
<td>The output image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th><strong>VX_SUCCESS</strong></th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.22 Dilate Image

3.22.1 Detailed Description

Implements Dilation, which grows the white space in a \texttt{VX_DF_IMAGE_U8} Boolean image. The output image dimensions should be the same as the dimensions of the input image.

This kernel uses a 3x3 box around the output pixel used to determine value.

\[
dst(x, y) = \max_{x' \leq x' \leq x+1, y' \leq y' \leq y+1} \ src(x', y')
\]

Note

For kernels that use other structuring patterns than 3x3 see \texttt{vxNonLinearFilterNode} or \texttt{vxuNonLinearFilter}.

Functions

- \texttt{vx_node VX_API_CALL vxDilate3x3Node (vx_graph graph, vx_image input, vx_image output)}
  [Graph] Creates a Dilation Image Node.

- \texttt{vx_status VX_API_CALL vxuDilate3x3 (vx_context context, vx_image input, vx_image output)}
  [Immediate] Dilates an image by a 3x3 window.

3.22.2 Function Documentation

\texttt{vx_node VX_API_CALL vxDilate3x3Node ( vx_graph graph, vx_image input, vx_image output )}

[Graph] Creates a Dilation Image Node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in \texttt{VX_DF_IMAGE_U8} format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in \texttt{VX_DF_IMAGE_U8} format, which must have the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

Returns

\texttt{vx_node}.

Return values

\texttt{vx_node} A node reference. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus}.

\texttt{vx_status VX_API_CALL vxuDilate3x3 ( vx_context context, vx_image input, vx_image output )}

[Immediate] Dilates an image by a 3x3 window.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in \texttt{VX_DF_IMAGE_U8} format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in \texttt{VX_DF_IMAGE_U8} format.</td>
</tr>
</tbody>
</table>
Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_Success</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.23 Equalize Histogram

3.23.1 Detailed Description

Equalizes the histogram of a grayscale image. The output image dimensions should be the same as the dimensions of the input image.

This kernel uses Histogram Equalization to modify the values of a grayscale image so that it will automatically have a standardized brightness and contrast.

Functions

- **vx_node VX_API_CALL vxEqualizeHistNode (vx_graph graph, vx_image input, vx_image output)**
  
  [Graph] Creates a Histogram Equalization node.

- **vx_status VX_API_CALL vxuEqualizeHist (vx_context context, vx_image input, vx_image output)**
  
  [Immediate] Equalizes the Histogram of a grayscale image.

3.23.2 Function Documentation

**vx_node VX_API_CALL vxEqualizeHistNode ( vx_graph graph, vx_image input, vx_image output )**

[Graph] Creates a Histogram Equalization node.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in input</td>
<td>The grayscale input image in VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>out output</td>
<td>The grayscale output image of type VX_DF_IMAGE_U8 with equalized brightness and contrast and same size as the input image.</td>
</tr>
</tbody>
</table>

Returns

vx_node.

Return values

- **vx_node** A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus.

**vx_status VX_API_CALL vxuEqualizeHist ( vx_context context, vx_image input, vx_image output )**

[Immediate] Equalizes the Histogram of a grayscale image.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>in input</td>
<td>The grayscale input image in VX_DF_IMAGE_U8</td>
</tr>
<tr>
<td>out output</td>
<td>The grayscale output image of type VX_DF_IMAGE_U8 with equalized brightness and contrast.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.24 Erode Image

3.24.1 Detailed Description

Implements Erosion, which shrinks the white space in a VX_DF_IMAGE_U8 Boolean image. The output image dimensions should be the same as the dimensions of the input image.

This kernel uses a 3x3 box around the output pixel used to determine value.

\[ \text{dst}(x,y) = \min_{x-1 \leq x' \leq x+1, y-1 \leq y' \leq y+1} \text{src}(x', y') \]

Note

For kernels that use other structuring patterns than 3x3 see \texttt{vxNonLinearFilterNode} or \texttt{vxuNonLinearFilter}.

Functions

- \texttt{vx_node VX_API_CALL vxErode3x3Node (vx_graph graph, vx_image input, vx_image output)}
  
  [Graph] Creates an Erosion Image Node.

- \texttt{vx_status VX_API_CALL vxuErode3x3 (vx_context context, vx_image input, vx_image output)}

  [Immediate] Erodes an image by a 3x3 window.

3.24.2 Function Documentation

\texttt{vx_node VX_API_CALL vxErode3x3Node ( vx_graph graph, vx_image input, vx_image output )}

[Graph] Creates an Erosion Image Node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in VX_DF_IMAGE_U8 format, which must have the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

Returns

\texttt{vx_node}. 

Return values

- \texttt{vx_node} : A node reference. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus}

\texttt{vx_status VX_API_CALL vxuErode3x3 ( vx_context context, vx_image input, vx_image output )}

[Immediate] Erodes an image by a 3x3 window.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in VX_DF_IMAGE_U8 format.</td>
</tr>
</tbody>
</table>
Returns

A \texttt{vx\_status\_e} enumeration.

Return values

<table>
<thead>
<tr>
<th>\texttt{VX_SUCCESS}</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An error occurred. See \texttt{vx_status_e}.</td>
</tr>
</tbody>
</table>
3.25 Fast Corners

3.25.1 Detailed Description

Computes the corners in an image using a method based upon FAST9 algorithm suggested in [? ] and with some updates from [? ] with modifications described below.

It extracts corners by evaluating pixels on the Bresenham circle around a candidate point. If \( N \) contiguous pixels are brighter than the candidate point by at least a threshold value \( t \) or darker by at least \( t \), then the candidate point is considered to be a corner. For each detected corner, its strength is computed. Optionally, a non-maxima suppression step is applied on all detected corners to remove multiple or spurious responses.

3.25.2 Segment Test Detector

The FAST corner detector uses the pixels on a Bresenham circle of radius 3 (16 pixels) to classify whether a candidate point \( p \) is actually a corner, given the following variables.

\[
\begin{align*}
I & \quad \text{input image} \\
p & \quad \text{candidate point position for a corner} \\
I_p & \quad \text{image intensity of the candidate point in image } I \\
x & \quad \text{pixel on the Bresenham circle around the candidate point } p \\
I_x & \quad \text{image intensity of the candidate point} \\
t & \quad \text{intensity difference threshold for a corner} \\
N & \quad \text{minimum number of contiguous pixel to detect a corner} \\
S & \quad \text{set of contiguous pixel on the Bresenham circle around the candidate point} \\
C_p & \quad \text{corner response at corner location } p 
\end{align*}
\]

The two conditions for FAST corner detection can be expressed as:

- C1: A set of \( N \) contiguous pixels \( S \), \( \forall x \in S, I_x > I_p + t \)
- C2: A set of \( N \) contiguous pixels \( S \), \( \forall x \in S, I_x < I_p - t \)

So when either of these two conditions is met, the candidate \( p \) is classified as a corner.

In this version of the FAST algorithm, the minimum number of contiguous pixels \( N \) is 9 (FAST9).

The value of the intensity difference threshold \( \text{strength\_thresh} \) of type \( \text{VX\_TYPE\_FLOAT32} \) must be within:

\[
\text{UINT}_8\text{MIN} < t < \text{UINT}_8\text{MAX}
\]

These limits are established due to the input data type \( \text{VX\_DF\_IMAGE\_U8} \).

**Corner Strength Computation** Once a corner has been detected, its strength (response, saliency, or score) shall be computed if \text{nonmax\_suppression} is set to true, otherwise the value of strength is undefined. The corner response \( C_p \) function is defined as the largest threshold \( t \) for which the pixel \( p \) remains a corner.

**Non-maximum suppression** If the \text{nonmax\_suppression} flag is true, a non-maxima suppression step is applied on the detected corners. The corner with coordinates \((x, y)\) is kept if and only if

\[
\begin{align*}
C_p(x, y) & \geq C_p(x - 1, y - 1) \text{ and } C_p(x, y) \geq C_p(x, y - 1) \text{ and } C_p(x, y) \geq C_p(x + 1, y - 1) \text{ and } C_p(x, y) \geq C_p(x - 1, y + 1) \\
C_p(x, y) & > C_p(x + 1, y) \text{ and } C_p(x, y) > C_p(x - 1, y + 1) \text{ and } C_p(x, y) > C_p(x + 1, y + 1)
\end{align*}
\]

See also

http://www.edwardrosten.com/work/fast.html
http://en.wikipedia.org/wiki/Features_from_accelerated_segment_test
Functions

- **vx_node VX_API_CALL vxFastCornersNode (vx_graph graph, vx_image input, vx_scalar strength_thresh, vx_bool nonmax_suppression, vx_array corners, vx_scalar num_corners)**

  [Graph] Creates a FAST Corners Node.

- **vx_status VX_API_CALL vxuFastCorners (vx_context context, vx_image input, vx_scalar strength_thresh, vx_bool nonmax_suppression, vx_array corners, vx_scalar num_corners)**

  [Immediate] Computes corners on an image using FAST algorithm and produces the array of feature points.

### 3.25.3 Function Documentation

**vx_node VX_API_CALL vxFastCornersNode ( vx_graph graph, vx_image input, vx_scalar strength_thresh, vx_bool nonmax_suppression, vx_array corners, vx_scalar num_corners )**

[Graph] Creates a FAST Corners Node.

#### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>graph</td>
</tr>
<tr>
<td>in</td>
<td>input</td>
</tr>
<tr>
<td>in</td>
<td>strength_thresh</td>
</tr>
<tr>
<td>in</td>
<td>nonmax_suppression</td>
</tr>
<tr>
<td>out</td>
<td>corners</td>
</tr>
<tr>
<td>out</td>
<td>num_corners</td>
</tr>
</tbody>
</table>

#### Returns

**vx_node.**

#### Return values

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_node</td>
<td>A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus</td>
</tr>
</tbody>
</table>

**vx_status VX_API_CALL vxuFastCorners ( vx_context context, vx_image input, vx_scalar strength_thresh, vx_bool nonmax_suppression, vx_array corners, vx_scalar num_corners )**

[Immediate] Computes corners on an image using FAST algorithm and produces the array of feature points.

#### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>context</td>
</tr>
<tr>
<td>in</td>
<td>input</td>
</tr>
<tr>
<td>in</td>
<td>strength_thresh</td>
</tr>
<tr>
<td>in</td>
<td>nonmax_suppression</td>
</tr>
</tbody>
</table>
Parameters

| out  | corners | Output corner `vx_array` of `VX_TYPE_KEYPOINT`. The order of the keypoints in this array is implementation dependent. |
| out  | num_corners | [optional] The total number of detected corners in image. Use a `VX_TYPE_SIZE` scalar. |

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th><code>VX_SUCCESS</code></th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>* An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.26 Gaussian Filter

3.26.1 Detailed Description

Computes a Gaussian filter over a window of the input image. The output image dimensions should be the same as the dimensions of the input image.

This filter uses the following convolution matrix:

\[
K_{\text{gaussian}} = \begin{bmatrix}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{bmatrix} \ast \frac{1}{16}
\]

Functions

- **vx_node VX_API_CALL vxGaussian3x3Node (vx_graph graph, vx_image input, vx_image output)**
  [Graph] Creates a Gaussian Filter Node.

- **vx_status VX_API_CALL vxuGaussian3x3 (vx_context context, vx_image input, vx_image output)**
  [Immediate] Computes a gaussian filter on the image by a 3x3 window.

3.26.2 Function Documentation

**vx_node VX_API_CALL vxGaussian3x3Node ( vx_graph graph, vx_image input, vx_image output )**

[Graph] Creates a Gaussian Filter Node.

Parameters

<table>
<thead>
<tr>
<th>In</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8 format.</td>
</tr>
<tr>
<td>Out</td>
<td>output</td>
<td>The output image in VX_DF_IMAGE_U8 format, which must have the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

Returns

**vx_node**

Return values

- **vx_node** A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus.

**vx_status VX_API_CALL vxuGaussian3x3 ( vx_context context, vx_image input, vx_image output )**

[Immediate] Computes a gaussian filter on the image by a 3x3 window.

Parameters

<table>
<thead>
<tr>
<th>In</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8 format.</td>
</tr>
<tr>
<td>Out</td>
<td>output</td>
<td>The output image in VX_DF_IMAGE_U8 format.</td>
</tr>
</tbody>
</table>

Returns

A **vx_status_e** enumeration.
Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.27 Non Linear Filter

3.27.1 Detailed Description

Computes a non-linear filter over a window of the input image. The output image dimensions should be the same as the dimensions of the input image.

The attribute \texttt{VX\_CONTEXT\_NONLINEAR\_MAX\_DIMENSION} enables the user to query the largest non-linear filter supported by the implementation of \texttt{vxNonLinearFilterNode}. The implementation must support all dimensions (height or width, not necessarily the same) up to the value of this attribute. The lowest value that must be supported for this attribute is 9.

Functions

- \texttt{vx\_node VX\_API\_CALL vxNonLinearFilterNode (vx\_graph graph, vx\_enum function, vx\_image input, vx\_matrix mask, vx\_image output)}

  \texttt{[Graph]} Creates a Non-linear Filter Node.

- \texttt{vx\_status VX\_API\_CALL vxuNonLinearFilter (vx\_context context, vx\_enum function, vx\_image input, vx\_matrix mask, vx\_image output)}

  \texttt{[Immediate]} Performs Non-linear Filtering.

3.27.2 Function Documentation

\texttt{vx\_node VX\_API\_CALL vxNonLinearFilterNode ( vx\_graph graph, vx\_enum function, vx\_image input, vx\_matrix mask, vx\_image output )}

\texttt{[Graph]} Creates a Non-linear Filter Node.

Parameters

\begin{tabular}{|c|c|}
\hline
\textbf{in} & \textit{graph} & The reference to the graph. \\
\hline
\textbf{in} & \textit{function} & The non-linear filter function. See \texttt{vx\_non\_linear\_filter\_e}. \\
\hline
\textbf{in} & \textit{input} & The input image in \texttt{VX\_DF\_IMAGE\_U8} format. \\
\hline
\textbf{in} & \textit{mask} & The mask to be applied to the Non-linear function. \texttt{VX\_MATRIX\_ORIGIN} attribute is used to place the mask appropriately when computing the resulting image. See \texttt{vxCreateMatrixFromPattern}. \\
\hline
\textbf{out} & \textit{output} & The output image in \texttt{VX\_DF\_IMAGE\_U8} format, which must have the same dimensions as the input image. \\
\hline
\end{tabular}

Returns

\texttt{vx\_node}.

Return values

\begin{tabular}{|c|c|}
\hline
\texttt{vx\_node} & A node reference. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus}. \\
\hline
\end{tabular}

\texttt{vx\_status VX\_API\_CALL vxuNonLinearFilter ( vx\_context context, vx\_enum function, vx\_image input, vx\_matrix mask, vx\_image output )}

\texttt{[Immediate]} Performs Non-linear Filtering.

Parameters

\begin{tabular}{|c|c|}
\hline
\textbf{in} & \textit{context} & The reference to the overall context. \\
\hline
\textbf{in} & \textit{function} & The non-linear filter function. See \texttt{vx\_non\_linear\_filter\_e}. \\
\hline
\end{tabular}
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>input</th>
<th>The input image in <code>VX_DF_IMAGE_U8</code> format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mask</td>
<td>The mask to be applied to the Non-linear function. <code>VX_MATRIX_ORIGIN</code> attribute is used to place the mask appropriately when computing the resulting image. See <code>vxCreateMatrixFromPattern</code> and <code>vxCreateMatrixFromPatternAndOrigin</code>.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in <code>VX_DF_IMAGE_U8</code> format.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.28 Harris Corners

3.28.1 Detailed Description

Computes the Harris Corners of an image.

The Harris Corners are computed with several parameters:

\[ I = \text{input image} \quad \text{(3.14)} \]

\[ T_c = \text{corner strength threshold} \quad \text{(3.15)} \]

\[ r = \text{euclidean radius} \quad \text{(3.16)} \]

\[ k = \text{sensitivity threshold} \quad \text{(3.17)} \]

\[ w = \text{window size} \quad \text{(3.18)} \]

\[ b = \text{block size} \quad \text{(3.19)} \]

\[ (3.20) \]

The computation to find the corner values or scores can be summarized as:

\[ G_x = \text{Sobel}_x(w, I) \quad \text{(3.21)} \]

\[ G_y = \text{Sobel}_y(w, I) \quad \text{(3.22)} \]

\[ A = \text{window}(G_x, G_y) \quad \text{(3.23)} \]

\[ \text{trace}(A) = \sum A_{xx} + \sum A_{yy} \quad \text{(3.24)} \]

\[ \text{det}(A) = \left( \sum A_{xx} \sum A_{yy} - \sum (A_{x,y})^2 \right)^2 \quad \text{(3.25)} \]

\[ M_c(x,y) = \text{det}(A) - k \times \text{trace}(A)^2 \quad \text{(3.26)} \]

\[ V_c(x,y) = \begin{cases} M_c(x,y) & \text{if } M_c(x,y) > T_c \\ 0 & \text{otherwise} \end{cases} \quad \text{(3.27)} \]

where \( V_c \) is the thresholded corner value.

The normalized Sobel kernels used for the gradient computation shall be as shown below:

- For gradient size 3:

\[ \text{Sobel}_x(\text{Normalized}) = \frac{1}{4 \times 255 \times b} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \]

\[ \text{Sobel}_y(\text{Normalized}) = \frac{1}{4 \times 255 \times b} \begin{bmatrix} 0 & 0 & -1 \\ 1 & 2 & 1 \end{bmatrix} \]

- For gradient size 5:

\[ \text{Sobel}_x(\text{Normalized}) = \frac{1}{16 \times 255 \times b} \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \\ -4 & -8 & 0 & 8 & 4 \end{bmatrix} \]

\[ \text{Sobel}_y(\text{Normalized}) = \frac{1}{16 \times 255 \times b} \begin{bmatrix} -6 & -12 & 0 & 12 & 6 \\ -4 & -8 & 0 & 8 & 4 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix} \]

- For gradient size 7:

\[ \text{Sobel}_x(\text{Normalized}) = \frac{1}{64 \times 255 \times b} \begin{bmatrix} -1 & -4 & -5 & 0 & 5 & 4 & 1 \\ -6 & -24 & -30 & 0 & 30 & 24 & 6 \\ -15 & -60 & -75 & 0 & 75 & 60 & 15 \\ -20 & -80 & -100 & 0 & 100 & 80 & 20 \\ -15 & -60 & -75 & 0 & 75 & 60 & 15 \\ -6 & -24 & -30 & 0 & 30 & 24 & 6 \\ -1 & -4 & -5 & 0 & 5 & 4 & 1 \end{bmatrix} \]
\[
\text{Sobel}_{\text{Normalized}} = \frac{1}{64 \times 255} \cdot \text{transpose}(\text{sobel}_x)
\]

\(V_c\) is then non-maximally suppressed, returning the same results as using the following algorithm:

- Filter the features using the non-maximum suppression algorithm defined for \texttt{vxFastCornersNode}.
- Create an array of features sorted by \(V_c\) in descending order: \(V_c(j) > V_c(j+1)\).
- Initialize an empty feature set \(F = \{\}\)
- For each feature \(j\) in the sorted array, while \(V_c(j) > T_c\):
  - If there is no feature \(i\) in \(F\) such that the Euclidean distance between pixels \(i\) and \(j\) is less than \(r\), add the feature \(j\) to the feature set \(F\).

An implementation shall support all values of Euclidean distance \(r\) that satisfy:

\[0 \leq \text{max\_dist} \leq 30\]

The feature set \(F\) is returned as a \texttt{vx\_array} of \texttt{vx\_keypoint\_t} structs.

### Functions

- \texttt{vx\_node VX\_API\_CALL vxHarrisCornersNode (vx\_graph graph, vx\_image input, vx\_scalar strength\_thresh, vx\_scalar min\_distance, vx\_scalar sensitivity, vx\_int32 gradient\_size, vx\_int32 block\_size, vx\_array corners, vx\_scalar num\_corners)}
  - \texttt{[Graph]} Creates a Harris Corners Node.
- \texttt{vx\_status VX\_API\_CALL vxuHarrisCorners (vx\_context context, vx\_image input, vx\_scalar strength\_thresh, vx\_scalar min\_distance, vx\_scalar sensitivity, vx\_int32 gradient\_size, vx\_int32 block\_size, vx\_array corners, vx\_scalar num\_corners)}
  - \texttt{[Immediate]} Computes the Harris Corners over an image and produces the array of scored points.

### 3.28.2 Function Documentation

\texttt{vx\_node VX\_API\_CALL vxHarrisCornersNode ( vx\_graph graph, vx\_image input, vx\_scalar strength\_thresh, vx\_scalar min\_distance, vx\_scalar sensitivity, vx\_int32 gradient\_size, vx\_int32 block\_size, vx\_array corners, vx\_scalar num\_corners )}

\texttt{[Graph]} Creates a Harris Corners Node.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input \texttt{VX_DF_IMAGE_U8} image.</td>
</tr>
<tr>
<td>in</td>
<td>strength_thresh</td>
<td>The \texttt{VX_TYPE_FLOAT32} minimum threshold with which to eliminate Harris Corner scores (computed using the normalized Sobel kernel).</td>
</tr>
<tr>
<td>in</td>
<td>min_distance</td>
<td>The \texttt{VX_TYPE_FLOAT32} radial Euclidean distance for non-maximum suppression.</td>
</tr>
<tr>
<td>in</td>
<td>sensitivity</td>
<td>The \texttt{VX_TYPE_FLOAT32} scalar sensitivity threshold (k) from the Harris-Stephens equation.</td>
</tr>
<tr>
<td>in</td>
<td>gradient_size</td>
<td>The gradient window size to use on the input. The implementation must support at least 3, 5, and 7.</td>
</tr>
<tr>
<td>in</td>
<td>block_size</td>
<td>The block window size used to compute the Harris Corner score. The implementation must support at least 3, 5, and 7.</td>
</tr>
<tr>
<td>out</td>
<td>corners</td>
<td>The array of \texttt{VX_TYPE_KEYPOINT} objects. The order of the keypoints in this array is implementation dependent.</td>
</tr>
<tr>
<td>out</td>
<td>num_corners</td>
<td>[optional] The total number of detected corners in image. Use a \texttt{VX_TYPE_SIZE} scalar.</td>
</tr>
</tbody>
</table>
Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

vx_status VX_API_CALL vxuHarrisCorners ( vx_context context, vx_image input, vx_scalar strength_thresh, vx_scalar min_distance, vx_scalar sensitivity, vx_int32 gradient_size, vx_int32 block_size, vx_array corners, vx_scalar num_corners )

[Immediate] Computes the Harris Corners over an image and produces the array of scored points.

Parameters

| in | context | The reference to the overall context. |
| in | input | The input VX_DF_IMAGE_U8 image. |
| in | strength_thresh | The VX_TYPE_FLOAT32 minimum threshold which to eliminate Harris Corner scores (computed using the normalized Sobel kernel). |
| in | min_distance | The VX_TYPE_FLOAT32 radial Euclidean distance for non-maximum suppression. |
| in | sensitivity | The VX_TYPE_FLOAT32 scalar sensitivity threshold k from the Harris-Stephens equation. |
| in | gradient_size | The gradient window size to use on the input. The implementation must support at least 3, 5, and 7. |
| in | block_size | The block window size used to compute the harris corner score. The implementation must support at least 3, 5, and 7. |
| out | corners | The array of VX_TYPE_KEYPOINT structs. The order of the keypoints in this array is implementation dependent. |
| out | num_corners | [optional] The total number of detected corners in image. Use a VX_TYPE_SIZE scalar |

Returns

A vx_status_e enumeration.

Return values

| VX_SUCCESS | Success |
| An error occurred. See vx_status_e. |
3.29 Histogram

3.29.1 Detailed Description

Generates a distribution from an image. This kernel counts the number of occurrences of each pixel value within the window size of a pre-calculated number of bins. A pixel with intensity 'I' will result in incrementing histogram bin 'i' where

\[ i = (I - offset) \times \text{numBins/\text{range}} \text{for} I \geq offset \]

and

\[ I < offset + \text{range}. \]

Pixels with intensities that don't meet these conditions will have no effect on the histogram. Here offset, range and numBins are values of histogram attributes (see VX_DISTRIBUTION_OFFSET, VX_DISTRIBUTION_RANGE, VX_DISTRIBUTION_BINS).

Functions

- **vx_node VX_API_CALL vxHistogramNode (vx_graph graph, vx_image input, vx_distribution distribution)**
  [Graph] Creates a Histogram node.

- **vx_status VX_API_CALL vxuHistogram (vx_context context, vx_image input, vx_distribution distribution)**
  [Immediate] Generates a distribution from an image.

3.29.2 Function Documentation

**vx_node VX_API_CALL vxHistogramNode ( vx_graph graph, vx_image input, vx_distribution distribution )**

[Graph] Creates a Histogram node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>out</td>
<td>distribution</td>
<td>The output distribution.</td>
</tr>
</tbody>
</table>

Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

**vx_status VX_API_CALL vxuHistogram ( vx_context context, vx_image input, vx_distribution distribution )**

[Immediate] Generates a distribution from an image.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8</td>
</tr>
<tr>
<td>out</td>
<td>distribution</td>
<td>The output distribution.</td>
</tr>
</tbody>
</table>
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th><code>VX_SUCCESS</code></th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.30 Gaussian Image Pyramid

3.30.1 Detailed Description

Computes a Gaussian Image Pyramid from an input image.

This vision function creates the Gaussian image pyramid from the input image using the particular 5x5 Gaussian Kernel:

\[
G = \frac{1}{256} \begin{bmatrix}
1 & 4 & 6 & 4 & 1 \\
4 & 16 & 24 & 16 & 4 \\
6 & 24 & 36 & 24 & 6 \\
4 & 16 & 24 & 16 & 4 \\
1 & 4 & 6 & 4 & 1 \\
\end{bmatrix}
\]

on each level of the pyramid then scales the image to the next level using VX_INTERPOLATION_NEAREST←_NEIGHBOR. For the Gaussian pyramid, level 0 shall always have the same resolution and contents as the input image. Pyramids configured with one of the following level scaling must be supported:

- VX_SCALE_PYRAMID_HALF
- VX_SCALE_PYRAMID_ORB

Functions

- vx_node VX_API_CALL vxGaussianPyramidNode (vx_graph graph, vx_image input, vx_pyramid gaussian)
  
  [Graph] Creates a node for a Gaussian Image Pyramid.

- vx_status VX_API_CALL vxuGaussianPyramid (vx_context context, vx_image input, vx_pyramid gaussian)
  
  [Immediate] Computes a Gaussian pyramid from an input image.

3.30.2 Function Documentation

vx_node VX_API_CALL vxGaussianPyramidNode ( vx_graph graph, vx_image input, vx_pyramid gaussian )

[Graph] Creates a node for a Gaussian Image Pyramid.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>gaussian</td>
<td>The Gaussian pyramid with VX_DF_IMAGE_U8 to construct.</td>
</tr>
</tbody>
</table>

See also

- Object: Pyramid

Returns

- vx_node.

Return values

- vx_node A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus.

vx_status VX_API_CALL vxuGaussianPyramid ( vx_context context, vx_image input, vx_pyramid gaussian )

[Immediate] Computes a Gaussian pyramid from an input image.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8</td>
</tr>
<tr>
<td>out</td>
<td>gaussian</td>
<td>The Gaussian pyramid with VX_DF_IMAGE_U8 to construct.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
### 3.31 Laplacian Image Pyramid

#### 3.31.1 Detailed Description

Computes a Laplacian Image Pyramid from an input image.

This vision function creates the Laplacian image pyramid from the input image. First, a Gaussian pyramid is created with the scale attribute `VX_SCALE_PYRAMID_HALF` and the number of levels equal to \( N + 1 \), where \( N \) is the number of levels in the laplacian pyramid. The border mode for the Gaussian pyramid calculation should be `VX_BORDER_REPLICATE`. Then, for each \( i = 0..N - 1 \), the Laplacian level \( L_i \) is computed as:

\[
L_i = G_i - \text{UpSample}(G_{i+1}).
\]

Here \( G_i \) is the \( i \)-th level of the Gaussian pyramid.

The \( \text{UpSample}(I) \) is computed by injecting even zero rows and columns and then convolves the result with the Gaussian 5x5 filter multiplied by 4.

\[
\text{UpSample}(I)_{x,y} = 4 \sum_{k=-2}^{2} \sum_{l=-2}^{2} I'_{x-k,y-l} W_{k+2,l+2}
\]

\[
I'_{x,y} = \begin{cases} 
I_{x,y} & \text{if } x \text{ and } y \text{ are even} \\
0 & \text{otherwise}
\end{cases}
\]

\[
W = \frac{1}{256} \begin{bmatrix}
1 & 4 & 6 & 4 & 1 \\
4 & 16 & 24 & 16 & 4 \\
6 & 24 & 36 & 24 & 6 \\
4 & 16 & 24 & 16 & 4 \\
1 & 4 & 6 & 4 & 1
\end{bmatrix}
\]

\( L_0 \) shall always have the same resolution as the input image. The output image is equal to \( G_N \).

The border mode for the \( \text{UpSample} \) calculation should be `VX_BORDER_REPLICATE`.

#### Functions

- `vx_node VX_API_CALL vxLaplacianPyramidNode (vx_graph graph, vx_image input, vx_pyramid laplacian, vx_image output)`

  [Graph] Creates a node for a Laplacian Image Pyramid.

- `vx_status VX_API_CALL vxuLaplacianPyramid (vx_context context, vx_image input, vx_pyramid laplacian, vx_image output)`

  [Immediate] Computes a Laplacian pyramid from an input image.

#### 3.31.2 Function Documentation

**vx_node VX_API_CALL vxLaplacianPyramidNode (vx_graph graph, vx_image input, vx_pyramid laplacian, vx_image output)**

[Graph] Creates a node for a Laplacian Image Pyramid.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in graph</code></td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td><code>in input</code></td>
<td>The input image in <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> format.</td>
</tr>
<tr>
<td><code>out laplacian</code></td>
<td>The Laplacian pyramid with <code>VX_DF_IMAGE_S16</code> to construct.</td>
</tr>
<tr>
<td><code>out output</code></td>
<td>The lowest resolution image in <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> format necessary to reconstruct the input image from the pyramid. The output image format should be same as input image format.</td>
</tr>
</tbody>
</table>
See also

Object: Pyramid

Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

vx_status VX_API_CALL vxuLaplacianPyramid ( vx_context context, vx_image input, vx_pyramid laplacian, vx_image output )

[Immediate] Computes a Laplacian pyramid from an input image.

Parameters

| in | context | The reference to the overall context. |
| in | input | The input image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format. |
| out | laplacian | The Laplacian pyramid with VX_DF_IMAGE_S16 to construct. |
| out | output | The lowest resolution image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format necessary to reconstruct the input image from the pyramid. The output image format should be same as input image format. |

See also

Object: Pyramid

Returns

A vx_status enumeration.

Return values

| VX_SUCCESS | Success. |
| * | An error occurred. See vx_status_e |
3.32 Reconstruction from a Laplacian Image Pyramid

3.32.1 Detailed Description

Reconstructs the original image from a Laplacian Image Pyramid.

This vision function reconstructs the image of the highest possible resolution from a Laplacian pyramid. The upscaled input image is added to the last level of the Laplacian pyramid $L_{N-1}$:

$$I_{N-1} = UpSample(input) + L_{N-1}$$

For the definition of the $UpSample$ function please see $vxLaplacianPyramidNode$. Correspondingly, for each pyramid level $i = 0..N-2$:

$$I_i = UpSample(I_{i+1}) + L_i$$

Finally, the output image is:

$$output = I_0$$

Functions

- **vx_node VX_API_CALL vxLaplacianReconstructNode (vx_graph graph, vx_pyramid laplacian, vx_image input, vx_image output)**
  
  [Graph] Reconstructs an image from a Laplacian Image pyramid.

- **vx_status VX_API_CALL vxuLaplacianReconstruct (vx_context context, vx_pyramid laplacian, vx_image input, vx_image output)**
  
  [Immediate] Reconstructs an image from a Laplacian Image pyramid.

3.32.2 Function Documentation

**vx_node VX_API_CALL vxLaplacianReconstructNode (vx_graph graph, vx_pyramid laplacian, vx_image input, vx_image output)**

[Graph] Reconstructs an image from a Laplacian Image pyramid.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in laplacian</td>
<td>The Laplacian pyramid with VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td>in input</td>
<td>The lowest resolution image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format for the Laplacian pyramid.</td>
</tr>
<tr>
<td>out output</td>
<td>The output image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format with the highest possible resolution reconstructed from the Laplacian pyramid. The output image format should be same as input image format.</td>
</tr>
</tbody>
</table>

See also

Object: Pyramid

Returns

- **vx_node.**

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Node could not be created.</td>
</tr>
<tr>
<td>*</td>
<td>Node handle.</td>
</tr>
</tbody>
</table>
vx_status VX_API_CALL vxuLaplacianReconstruct ( vx_context context, vx_pyramid laplacian, vx_image input, vx_image output )

[Immediate] Reconstructs an image from a Laplacian Image pyramid.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>laplacian</td>
<td>The Laplacian pyramid with VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td>in</td>
<td>input</td>
<td>The lowest resolution image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format for the Laplacian pyramid.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format with the highest possible resolution reconstructed from the Laplacian pyramid. The output image format should be same as input image format.</td>
</tr>
</tbody>
</table>

See also

Object: Pyramid

Returns

A vx_status enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success.</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e</td>
</tr>
</tbody>
</table>
3.33 Integral Image

3.33.1 Detailed Description

Computes the integral image of the input. The output image dimensions should be the same as the dimensions of the input image.

Each output pixel is the sum of the corresponding input pixel and all other pixels above and to the left of it.

\[ dst(x, y) = sum(x, y) \]

where, for \( x \geq 0 \) and \( y \geq 0 \)

\[ sum(x, y) = src(x, y) + sum(x - 1, y) + sum(x, y - 1) - sum(x - 1, y - 1) \]

otherwise,

\[ sum(x, y) = 0 \]

The overflow policy used is VX_CONVERT_POLICY_WRAP.

Functions

- \texttt{vx_node VX_API_CALL vxIntegralImageNode (vx_graph graph, vx_image input, vx_image output)}
  [Graph] Creates an Integral Image Node.

- \texttt{vx_status VX_API_CALL vxuIntegralImage (vx_context context, vx_image input, vx_image output)}
  [Immediate] Computes the integral image of the input.

3.33.2 Function Documentation

\texttt{vx_node VX_API_CALL vxIntegralImageNode ( vx_graph graph, vx_image input, vx_image output )}
[Graph] Creates an Integral Image Node.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in    graph</td>
<td>The reference to the graph.</td>
<td></td>
</tr>
<tr>
<td>in    input</td>
<td>The input image in VX_DF_IMAGE_U8 format.</td>
<td></td>
</tr>
<tr>
<td>out   output</td>
<td>The output image in VX_DF_IMAGE_U32 format, which must have the same dimensions as the input image.</td>
<td></td>
</tr>
</tbody>
</table>

Returns

\texttt{vx_node}.

Return values

- \texttt{vx_node} A node reference. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus}.

\texttt{vx_status VX_API_CALL vxuIntegralImage ( vx_context context, vx_image input, vx_image output )}
[Immediate] Computes the integral image of the input.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in    context</td>
<td>The reference to the overall context.</td>
<td></td>
</tr>
<tr>
<td>in    input</td>
<td>The input image in VX_DF_IMAGE_U8 format.</td>
<td></td>
</tr>
<tr>
<td>out   output</td>
<td>The output image in VX_DF_IMAGE_U32 format.</td>
<td></td>
</tr>
</tbody>
</table>
Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.34 Magnitude

3.34.1 Detailed Description

Implements the Gradient Magnitude Computation Kernel. The output image dimensions should be the same as the dimensions of the input images.

This kernel takes two gradients in VX_DF_IMAGE_S16 format and computes the VX_DF_IMAGE_S16 normalized magnitude. Magnitude is computed as:

\[ mag(x, y) = \sqrt{grad_x(x, y)^2 + grad_y(x, y)^2} \]

The conceptual definition describing the overflow is given as:

```c
uint16 z = uint16( sqrt( double( uint32( int32(x) * int32(x) ) + uint32( int32(y) * int32(y) ) ) ) + 0.5);
int16 mag = z > 32767 ? 32767 : z;
```

Functions

- `vx_node VX_API_CALL vxMagnitudeNode (vx_graph graph, vx_image grad_x, vx_image grad_y, vx_image mag)` [Graph] Create a Magnitude node.

- `vx_status VX_API_CALL vxuMagnitude (vx_context context, vx_image grad_x, vx_image grad_y, vx_image mag)` [Immediate] Invokes an immediate Magnitude.

3.34.2 Function Documentation

`vx_node VX_API_CALL vxMagnitudeNode ( vx_graph graph, vx_image grad_x, vx_image grad_y, vx_image mag )`

[Graph] Create a Magnitude node.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in</td>
<td>grad_x</td>
<td>The input x image. This must be in VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td>in</td>
<td>grad_y</td>
<td>The input y image. This must be in VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td>out</td>
<td>mag</td>
<td>The magnitude image. This is in VX_DF_IMAGE_S16 format. Must have the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

See also

VX_KERNEL_MAGNITUDE

Returns

vx_node.

Return values

<table>
<thead>
<tr>
<th>vx_node</th>
<th>A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus</th>
</tr>
</thead>
</table>
vx_status VX_API_CALL vxuMagnitude ( vx_context context, vx_image grad_x, vx_image grad_y, vx_image mag )

[Immediate] Invokes an immediate Magnitude.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>grad←_x</td>
<td>The input x image. This must be in VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td>in</td>
<td>grad←_y</td>
<td>The input y image. This must be in VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td>out</td>
<td>mag</td>
<td>The magnitude image. This will be in VX_DF_IMAGE_S16 format.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.35 Mean and Standard Deviation

3.35.1 Detailed Description

Computes the mean pixel value and the standard deviation of the pixels in the input image (which has a dimension width and height).

The mean value is computed as:

\[
\mu = \frac{\sum_{y=0}^{h} \sum_{x=0}^{w} \text{src}(x,y)}{(width \times height)}
\]

The standard deviation is computed as:

\[
\sigma = \sqrt{\frac{\sum_{y=0}^{h} \sum_{x=0}^{w} (\mu - \text{src}(x,y))^2}{(width \times height)}}
\]

Functions

- \text{vx_node VX_API_CALL vxMeanStdDevNode (vx_graph graph, vx_image input, vx_scalar mean, vx_scalar stddev)}
  
  [Graph] Creates a mean value and optionally, a standard deviation node.

- \text{vx_status VX_API_CALL vxuMeanStdDev (vx_context context, vx_image input, vx_float32 \ast mean, vx_float32 \ast stddev)}

  [Immediate] Computes the mean value and optionally the standard deviation.

3.35.2 Function Documentation

\text{vx_node VX_API_CALL vxMeanStdDevNode ( vx_graph graph, vx_image input, vx_scalar mean, vx_scalar stddev )}

[Graph] Creates a mean value and optionally, a standard deviation node.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{in graph}</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>\text{in input}</td>
<td>The input image. \text{VX_DF_IMAGE_U8} is supported.</td>
</tr>
<tr>
<td>\text{out mean}</td>
<td>The \text{VX_TYPE_FLOAT32} average pixel value.</td>
</tr>
<tr>
<td>\text{out stddev}</td>
<td>[optional] The \text{VX_TYPE_FLOAT32} standard deviation of the pixel values.</td>
</tr>
</tbody>
</table>

Returns

\text{vx_node}.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{vx_node}</td>
<td>A node reference. Any possible errors preventing a successful creation should be checked using \text{vxGetStatus}.</td>
</tr>
</tbody>
</table>

\text{vx_status VX_API_CALL vxuMeanStdDev ( vx_context context, vx_image input, vx_float32 \ast mean, vx_float32 \ast stddev )}

[Immediate] Computes the mean value and optionally the standard deviation.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{in context}</td>
<td>The reference to the overall context.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>input</th>
<th>The input image. VX_DF_IMAGE_U8 is supported.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>mean</td>
<td>The average pixel value.</td>
</tr>
<tr>
<td>out</td>
<td>stddev</td>
<td>[optional] The standard deviation of the pixel values.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
### 3.36 Median Filter

#### 3.36.1 Detailed Description

Computes a median pixel value over a window of the input image. The output image dimensions should be the same as the dimensions of the input image.

The median is the middle value over an odd-numbered, sorted range of values.

**Note**

For kernels that use other structuring patterns than 3x3 see `vxNonLinearFilterNode` or `vxuNonLinearFilter`.

**Functions**

- **vx_node VX_API_CALL vxMedian3x3Node (vx_graph graph, vx_image input, vx_image output)**
  
  [Graph] Creates a Median Image Node.

- **vx_status VX_API_CALL vxuMedian3x3 (vx_context context, vx_image input, vx_image output)**
  
  [Immediate] Computes a median filter on the image by a 3x3 window.

#### 3.36.2 Function Documentation

**vx_node VX_API_CALL vxMedian3x3Node (vx_graph graph, vx_image input, vx_image output)**

[Graph] Creates a Median Image Node.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in VX_DF_IMAGE_U8 format, which must have the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

**Returns**

`vx_node`

**Return values**

| `vx_node` | A node reference. Any possible errors preventing a successful creation should be checked using `vxGetStatus` |

**vx_status VX_API_CALL vxuMedian3x3 (vx_context context, vx_image input, vx_image output)**

[Immediate] Computes a median filter on the image by a 3x3 window.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in VX_DF_IMAGE_U8 format.</td>
</tr>
</tbody>
</table>

**Returns**

A `vx_status_e` enumeration.
Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.37 Min, Max Location

3.37.1 Detailed Description
Finds the minimum and maximum values in an image and a location for each.

If the input image has several minimums/maximuns, the kernel returns all of them.

\[
\begin{align*}
minVal &= \min_{0 \leq x' \leq width, \ 0 \leq y' \leq height} src(x', y') \\
maxVal &= \max_{0 \leq x' \leq width, \ 0 \leq y' \leq height} src(x', y')
\end{align*}
\]

Functions

- **vx_node VX_API_CALL vxMinMaxLocNode** (vx_graph graph, vx_image input, vx_scalar minVal, vx_scalar maxVal, vx_array minLoc, vx_array maxLoc, vx_scalar minCount, vx_scalar maxCount)
  [Graph] Creates a min,max,loc node.

- **vx_status VX_API_CALL vxuMinMaxLoc** (vx_context context, vx_image input, vx_scalar minVal, vx_scalar maxVal, vx_array minLoc, vx_array maxLoc, vx_scalar minCount, vx_scalar maxCount)
  [Immediate] Computes the minimum and maximum values of the image.

3.37.2 Function Documentation

**vx_node VX_API_CALL vxMinMaxLocNode** ( vx_graph graph, vx_image input, vx_scalar minVal, vx_scalar maxVal, vx_array minLoc, vx_array maxLoc, vx_scalar minCount, vx_scalar maxCount )

[Graph] Creates a min,max,loc node.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> graph</td>
<td>The reference to create the graph.</td>
</tr>
<tr>
<td><strong>in</strong> input</td>
<td>The input image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td><strong>out</strong> minVal</td>
<td>The minimum value in the image, which corresponds to the type of the input.</td>
</tr>
<tr>
<td><strong>out</strong> maxVal</td>
<td>The maximum value in the image, which corresponds to the type of the input.</td>
</tr>
<tr>
<td><strong>out</strong> minLoc</td>
<td>[optional] The minimum VX_TYPE_COORDINATES2D locations. If the input image has several minimums, the kernel will return up to the capacity of the array.</td>
</tr>
<tr>
<td><strong>out</strong> maxLoc</td>
<td>[optional] The maximum VX_TYPE_COORDINATES2D locations. If the input image has several maximums, the kernel will return up to the capacity of the array.</td>
</tr>
<tr>
<td><strong>out</strong> minCount</td>
<td>[optional] The total number of detected minimums in image. Use a VX_TYPE_SIZE scalar.</td>
</tr>
<tr>
<td><strong>out</strong> maxCount</td>
<td>[optional] The total number of detected maximums in image. Use a VX_TYPE_SIZE scalar.</td>
</tr>
</tbody>
</table>

Returns

**vx_node**

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus. |
vx_status VX_API_CALL vxuMinMaxLoc ( vx_context context, vx_image input, vx_scalar minVal, vx_scalar maxVal, vx_array minLoc, vx_array maxLoc, vx_scalar minCount, vx_scalar maxCount )

[Immediate] Computes the minimum and maximum values of the image.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>in input</td>
<td>The input image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td>out minVal</td>
<td>The minimum value in the image, which corresponds to the type of the input.</td>
</tr>
<tr>
<td>out maxVal</td>
<td>The maximum value in the image, which corresponds to the type of the input.</td>
</tr>
<tr>
<td>out minLoc</td>
<td>[optional] The minimum VX_TYPE_COORDINATES2D locations. If the input image has several minimums, the kernel will return up to the capacity of the array.</td>
</tr>
<tr>
<td>out maxLoc</td>
<td>[optional] The maximum VX_TYPE_COORDINATES2D locations. If the input image has several maximums, the kernel will return up to the capacity of the array.</td>
</tr>
<tr>
<td>out minCount</td>
<td>[optional] The total number of detected minimums in image. Use a VX_TYPE_SIZE scalar.</td>
</tr>
<tr>
<td>out maxCount</td>
<td>[optional] The total number of detected maximums in image. Use a VX_TYPE_SIZE scalar.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.38 Min

3.38.1 Detailed Description

Implements a pixel-wise minimum kernel. The output image dimensions should be the same as the dimensions of the input image.

Performing a pixel-wise minimum on a VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 images. All data types of the input and output images must match.

\[ out[i, j] = (in1[i, j] < in2[i, j]?in1[i, j]:in2[i, j]). \]

Functions

- `vx_node VX_API_CALL vxMinNode (vx_graph graph, vx_image in1, vx_image in2, vx_image out)`  
  [Graph] Creates a pixel-wise minimum kernel.

- `vx_status VX_API_CALL vxuMin (vx_context context, vx_image in1, vx_image in2, vx_image out)`  
  [Immediate] Computes pixel-wise minimum values between two images.

3.38.2 Function Documentation

`vx_node VX_API_CALL vxMinNode (vx_graph graph, vx_image in1, vx_image in2, vx_image out)`  
[Graph] Creates a pixel-wise minimum kernel.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph where to create the node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>in1</td>
<td>The first input image. Must be of type VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>The second input image. Must be of type VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The output image which will hold the result of min and will have the same type and dimensions of the input images.</td>
</tr>
</tbody>
</table>

Returns

`vx_node`.

Return values

- `vx_node` A node reference. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

`vx_status VX_API_CALL vxuMin (vx_context context, vx_image in1, vx_image in2, vx_image out)`  
[Immediate] Computes pixel-wise minimum values between two images.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>in1</td>
<td>The first input image. Must be of type VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>The second input image. Must be of type VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The output image which will hold the result of min.</td>
</tr>
</tbody>
</table>
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th><code>VX_SUCCESS</code></th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.39 Max

3.39.1 Detailed Description

Implements a pixel-wise maximum kernel. The output image dimensions should be the same as the dimensions of the input image.

Performing a pixel-wise maximum on a VX_DF_IMAGE_U8 images or VX_DF_IMAGE_S16. All data types of the input and output images must match.

\[
\text{out}[i, j] = \text{out}[i, j] = (\text{in1}[i, j] > \text{in2}[i, j] ? \text{in1}[i, j] : \text{in2}[i, j]).
\]

Functions

- **vx_node VX_API_CALL vxMaxNode (vx_graph graph, vx_image in1, vx_image in2, vx_image out)**
  [Graph] Creates a pixel-wise maximum kernel.

- **vx_status VX_API_CALL vxuMax (vx_context context, vx_image in1, vx_image in2, vx_image out)**
  [Immediate] Computes pixel-wise maximum values between two images.

3.39.2 Function Documentation

**vx_node VX_API_CALL vxMaxNode ( vx_graph graph, vx_image in1, vx_image in2, vx_image out )**

[Graph] Creates a pixel-wise maximum kernel.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in graph</td>
<td>The reference to the graph where to create the node.</td>
</tr>
<tr>
<td>in in1</td>
<td>The first input image. Must be of type VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>in in2</td>
<td>The second input image. Must be of type VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>out out</td>
<td>The output image which will hold the result of max and will have the same type and dimensions of the input images.</td>
</tr>
</tbody>
</table>

### Returns

**vx_node.**

### Return values

**vx_node** A node reference. Any possible errors preventing a successful creation should be checked using `vxGetStatus`

**vx_status VX_API_CALL vxuMax ( vx_context context, vx_image in1, vx_image in2, vx_image out )**

[Immediate] Computes pixel-wise maximum values between two images.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>in in1</td>
<td>The first input image. Must be of type VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>in in2</td>
<td>The second input image. Must be of type VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>out out</td>
<td>The output image which will hold the result of max.</td>
</tr>
</tbody>
</table>
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.40 Optical Flow Pyramid (LK)

3.40.1 Detailed Description

Computes the optical flow using the Lucas-Kanade method between two pyramid images.

The function is an implementation of the algorithm described in [8]. The function inputs are two vx_pyramid objects, old and new, along with a vx_array of vx_keypoint_t structs to track from the old vx_pyramid. Both pyramids old and new pyramids must have the same dimensionality. VX_SCALE_PYRAMID_HALF pyramidal scaling must be supported.

The function outputs a vx_array of vx_keypoint_t structs that were tracked from the old vx_pyramid to the new vx_pyramid. Each element in the vx_array of vx_keypoint_t structs in the new array may be valid or not. The implementation shall return the same number of vx_keypoint_t structs in the new vx_array that were in the old vx_array.

In more detail: The Lucas-Kanade method finds the affine motion vector $V$ for each point in the old image tracking points array, using the following equation:

$$
\begin{bmatrix}
V_x \\
V_y
\end{bmatrix} = 
\begin{bmatrix}
\sum I_x^2 & \sum I_x I_y \\
\sum I_x I_y & \sum I_y^2
\end{bmatrix}^{-1}
\begin{bmatrix}
-\sum I_x I_t \\
-\sum I_y I_t
\end{bmatrix}
$$

Where $I_x$ and $I_y$ are obtained using the Scharr gradients on the input image:

$$
G_x = 
\begin{bmatrix}
+3 & 0 & -3 \\
+10 & 0 & -10 \\
+3 & 0 & -3
\end{bmatrix}
$$

$$
G_y = 
\begin{bmatrix}
+3 & +10 & +3 \\
0 & 0 & 0 \\
-3 & -10 & -3
\end{bmatrix}
$$

$I_t$ is obtained by a simple difference between the same pixel in both images. $I$ is defined as the adjacent pixels to the point $p(x, y)$ under consideration. With a given window size of $M$, $I$ is $M^2$ points. The pixel $p(x, y)$ is centered in the window. In practice, to get an accurate solution, it is necessary to iterate multiple times on this scheme (in a Newton-Raphson fashion) until:

- the residual of the affine motion vector is smaller than a threshold
- And/or maximum number of iteration achieved. Each iteration, the estimation of the previous iteration is used by changing $I_t$ to be the difference between the old image and the pixel with the estimated coordinates in the new image. Each iteration the function checks if the pixel to track was lost. The criteria for lost tracking is

  - the matrix above is invertible. (The determinant of the matrix is less than a threshold $10^{-7}$.) Or the minimum eigenvalue of the matrix is smaller then a threshold $10^{-4}$.
  - Also lost tracking happens when the point tracked coordinate is outside the image coordinates. When vx_true_e is given as the input to use←_initial_estimates, the algorithm starts by calculating $I_t$ as the difference between the old image and the pixel with the initial estimated coordinates in the new image. The input vx_array of vx_keypoint_t structs with tracking_status set to zero (lost) are copied to the new vx_array.

Clients are responsible for editing the output vx_array of vx_keypoint_t structs array before applying it as the input vx_array of vx_keypoint_t structs for the next frame. For example, vx_keypoint_t structs with tracking_status set to zero may be removed by a client for efficiency.

This function changes just the $x$, $y$, and tracking_status members of the vx_keypoint_t structure and behaves as if it copied the rest of the old tracking vx_keypoint_t to new image vx_keypoint_t.

Functions

- **vx_node VX_API_CALL vxuOpticalFlowPyrLKNode (vx_graph graph, vx_pyramid old_images, vx_pyramid new_images, vx_array old_points, vx_array new_points_estimates, vx_array new_points, vx_enum termination, vx_scalar epsilon, vx_scalar num_iterations, vx_scalar use_initial_estimate, vx_size window_dimension)**
  
  [Graph] Creates a Lucas Kanade Tracking Node.

- **vx_status VX_API_CALL vxuOpticalFlowPyrLK (vx_context context, vx_pyramid old_images, vx_pyramid new_images, vx_array old_points, vx_array new_points_estimates, vx_array new_points, vx_enum termination, vx_scalar epsilon, vx_scalar num_iterations, vx_scalar use_initial_estimate, vx_size window_dimension)**

  [Immediate] Computes an optical flow on two images.
### 3.40.2 Function Documentation

**vx_node VX_API_CALL vxOpticalFlowPyrLKNode (vx_graph graph, vx_pyramid old_images, vx_pyramid new_images, vx_array old_points, vx_array new_points_estimates, vx_array new_points, vx_enum termination, vx_scalar epsilon, vx_scalar num_iterations, vx_scalar use_initial_estimate, vx_size window_dimension )**

[Graph] Creates a Lucas Kanade Tracking Node.

#### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><em>graph</em> The reference to the graph.</td>
</tr>
<tr>
<td>in</td>
<td><em>old_images</em> Input of first (old) image pyramid in <strong>VX_DF_IMAGE_U8</strong>.</td>
</tr>
<tr>
<td>in</td>
<td><em>new_images</em> Input of destination (new) image pyramid <strong>VX_DF_IMAGE_U8</strong>.</td>
</tr>
<tr>
<td>in</td>
<td><em>old_points</em> An array of key points in a vx_array of <strong>VX_TYPE_KEYPOINT</strong>; those key points are defined at the old_images high resolution pyramid.</td>
</tr>
<tr>
<td>in</td>
<td><em>new_points_estimates</em> An array of estimation on what is the output key points in a vx_array of <strong>VX_TYPE_KEYPOINT</strong>; those keypoints are defined at the new_images high resolution pyramid.</td>
</tr>
<tr>
<td>out</td>
<td><em>new_points</em> An output array of key points in a vx_array of <strong>VX_TYPE_KEYPOINT</strong>; those key points are defined at the new_images high resolution pyramid.</td>
</tr>
<tr>
<td>in</td>
<td><em>termination</em> The termination can be <strong>VX_TERM_CRITERIA_ITERATIONS</strong> or <strong>VX_TERM_CRITERIA_EPSILON</strong> or <strong>VX_TERM_CRITERIA_BOTH</strong>.</td>
</tr>
<tr>
<td>in</td>
<td><em>epsilon</em> The <strong>vx_float32</strong> error for terminating the algorithm.</td>
</tr>
<tr>
<td>in</td>
<td><em>num_iterations</em> The number of iterations. Use a <strong>VX_TYPE_UINT32</strong> scalar.</td>
</tr>
<tr>
<td>in</td>
<td><em>use_initial_estimate</em> Use a <strong>VX_TYPE_BOOL</strong> scalar.</td>
</tr>
<tr>
<td>in</td>
<td><em>window_dimension</em> The size of the window on which to perform the algorithm. See <strong>VX_CONTEXT_OPTICAL_FLOW_MAX_WINDOW_DIMENSION</strong></td>
</tr>
</tbody>
</table>

#### Returns

**vx_node.**

#### Return values

| **vx_node** | A node reference. Any possible errors preventing a successful creation should be checked using **vxGetStatus** |

**vx_status VX_API_CALL vxuOpticalFlowPyrLK (vx_context context, vx_pyramid old_images, vx_pyramid new_images, vx_array old_points, vx_array new_points_estimates, vx_array new_points, vx_enum termination, vx_scalar epsilon, vx_scalar num_iterations, vx_scalar use_initial_estimate, vx_size window_dimension )**

[Immediate] Computes an optical flow on two images.

#### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><em>context</em> The reference to the overall context.</td>
</tr>
<tr>
<td>in</td>
<td><em>old_images</em> Input of first (old) image pyramid in <strong>VX_DF_IMAGE_U8</strong>.</td>
</tr>
<tr>
<td>in</td>
<td><em>new_images</em> Input of destination (new) image pyramid <strong>VX_DF_IMAGE_U8</strong>.</td>
</tr>
<tr>
<td>in</td>
<td><em>old_points</em> An array of key points in a vx_array of <strong>VX_TYPE_KEYPOINT</strong>; those key points are defined at the old_images high resolution pyramid</td>
</tr>
<tr>
<td>in</td>
<td><em>new_points_estimates</em> An array of estimation on what is the output key points in a vx_array of <strong>VX_TYPE_KEYPOINT</strong>; those keypoints are defined at the new_images high resolution pyramid</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out new_points</td>
<td>an output array of key points in a vx_array of VX_TYPE_KEYPOINT</td>
</tr>
<tr>
<td></td>
<td>those key points are defined at the new_images high resolution pyramid</td>
</tr>
<tr>
<td>in termination</td>
<td>termination can be VX_TERM_CRITERIA_ITERATIONS or VX_TERM_CRITERIA_EPSILON or VX_TERM_CRITERIA_BOTH</td>
</tr>
<tr>
<td>in epsilon</td>
<td>is the vx_float32 error for terminating the algorithm</td>
</tr>
<tr>
<td>in num_iterations</td>
<td>is the number of iterations. Use a VX_TYPE_UINT32 scalar.</td>
</tr>
<tr>
<td>in use_initial_estimate</td>
<td>Can be set to either vx_false_e or vx_true_e.</td>
</tr>
<tr>
<td>in window_dimension</td>
<td>The size of the window on which to perform the algorithm. See VX_CONTEXT_OPTICAL_FLOW_MAX_WINDOW_DIMENSION</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.41 Phase

3.41.1 Detailed Description

Implements the Gradient Phase Computation Kernel. The output image dimensions should be the same as the dimensions of the input images.

This kernel takes two gradients in VX_DF_IMAGE_S16 format and computes the angles for each pixel and stores this in a VX_DF_IMAGE_U8 image.

\[ \phi = \tan^{-1} \left( \frac{\text{grad}_y(x,y)}{\text{grad}_x(x,y)} \right) \]

Where \( \phi \) is then translated to \( 0 \leq \phi < 2\pi \). Each \( \phi \) value is then mapped to the range 0 to 255 inclusive.

Functions

- **vx_node VX_API_CALL vxPhaseNode (vx_graph graph, vx_image grad_x, vx_image grad_y, vx_image orientation)**  
  [Graph] Creates a Phase node.

- **vx_status VX_API_CALL vxuPhase (vx_context context, vx_image grad_x, vx_image grad_y, vx_image orientation)**  
  [Immediate] Invokes an immediate Phase.

3.41.2 Function Documentation

**vx_node VX_API_CALL vxPhaseNode ( vx_graph graph, vx_image grad_x, vx_image grad_y, vx_image orientation )**  
[Graph] Creates a Phase node.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in grad_x</td>
<td>The input x image. This must be in VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td>in grad_y</td>
<td>The input y image. This must be in VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td>out orientation</td>
<td>The phase image. This is in VX_DF_IMAGE_U8 format, and must have the same dimensions as the input images.</td>
</tr>
</tbody>
</table>

See also

VX_KERNEL_PHASE

Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

**vx_status VX_API_CALL vxuPhase ( vx_context context, vx_image grad_x, vx_image grad_y, vx_image orientation )**  
[Immediate] Invokes an immediate Phase.
CHAPTER 3. MODULE DOCUMENTATION

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>grad_x</td>
<td>The input x image. This must be in VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td>in</td>
<td>grad_y</td>
<td>The input y image. This must be in VX_DF_IMAGE_S16 format.</td>
</tr>
<tr>
<td>out</td>
<td>orientation</td>
<td>The phase image. This will be in VX_DF_IMAGE_U8 format.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.42 Pixel-wise Multiplication

3.42.1 Detailed Description

Performs element-wise multiplication between two images and a scalar value. The output image dimensions should be the same as the dimensions of the input images.

Pixel-wise multiplication is performed between the pixel values in two VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 images and a scalar floating-point number scale. The output image can be VX_DF_IMAGE_U8 only if both source images are VX_DF_IMAGE_U8 and the output image is explicitly set to VX_DF_IMAGE_U8. It is otherwise VX_DF_IMAGE_S16. If one of the input images is of type VX_DF_IMAGE_S16 and the output image is explicitly set to VX_DF_IMAGE_S16, all values are converted to VX_DF_IMAGE_S16.

The scale with a value of $1/2^n$, where $n$ is an integer and $0 \leq n \leq 15$, and $1/255$ (0x1.010102p-8 C99 float hex) must be supported. The support for other values of scale is not prohibited. Furthermore, for scale with a value of $1/255$ the rounding policy of VX_ROUND_POLICY_TO_NEAREST_EVEN must be supported whereas for the scale with value of $1/2^n$ the rounding policy of VX_ROUND_POLICY_TO_ZERO must be supported. The support of other rounding modes for any values of scale is not prohibited.

The rounding policy VX_ROUND_POLICY_TO_ZERO for this function is defined as:

$$\text{reference}(x, y, scale) = \text{truncate}((\text{int}_32)\text{in}_1(x, y)) \times ((\text{int}_32)\text{in}_2(x, y)) \times (\text{double})\text{scale}$$

The rounding policy VX_ROUND_POLICY_TO_NEAREST_EVEN for this function is defined as:

$$\text{reference}(x, y, scale) = \text{round}}_\text{to}_\text{nearest}_\text{even}((\text{int}_32)\text{in}_1(x, y)) \times ((\text{int}_32)\text{in}_2(x, y)) \times (\text{double})\text{scale}$$

The overflow handling is controlled by an overflow-policy parameter. For each pixel value in the two input images:

$$\text{out}(x, y) = \text{in}_1(x, y) \times \text{in}_2(x, y) \times \text{scale}$$

Functions

- **vx_node VX_API_CALL vxMultiplyNode (vx_graph graph, vx_image in1, vx_image in2, vx_scalar scale, vx Enumerable overflow_policy, vx_enum rounding_policy, vx_image out)**

  [Graph] Creates a pixelwise-multiplication node.

- **vx_status VX_API_CALL vxuMultiply (vx_context context, vx_image in1, vx_image in2, vx_float32 scale, vx Enumerable overflow_policy, vx_enum rounding_policy, vx_image out)**

  [Immediate] Performs elementwise multiplications on pixel values in the input images and a scale.

3.42.2 Function Documentation

**vx_node VX_API_CALL vxMultiplyNode ( vx_graph graph, vx_image in1, vx_image in2, vx_scalar scale, vx Enumerable overflow_policy, vx_enum rounding_policy, vx_image out )**

[Graph] Creates a pixelwise-multiplication node.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in in1</td>
<td>An input image, VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>in in2</td>
<td>An input image, VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>in scale</td>
<td>A non-negative VX_TYPE_FLOAT32 multiplied to each product before overflow handling.</td>
</tr>
<tr>
<td>in overflow_policy</td>
<td>A VX_TYPE_ENUM of the vx_convert_policy_e enumeration.</td>
</tr>
<tr>
<td>in rounding_policy</td>
<td>A VX_TYPE_ENUM of the vx_round_policy_e enumeration.</td>
</tr>
<tr>
<td>out out</td>
<td>The output image, a VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 image. Must have the same type and dimensions of the input images.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

vx_status VX_API_CALL vxuMultiply( vx_context context, vx_image in1, vx_image in2, vx_float32 scale, vx_enum overflow_policy, vx_enum rounding_policy, vx_image out )

[Immediate] Performs elementwise multiplications on pixel values in the input images and a scale.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>in1</td>
<td>A VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 input image.</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>A VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 input image.</td>
</tr>
<tr>
<td>in</td>
<td>scale</td>
<td>A non-negative VX_TYPE_FLOAT32 multiplied to each product before overflow handling.</td>
</tr>
<tr>
<td>in</td>
<td>overflow_policy</td>
<td>A vx_convert_policy_e enumeration.</td>
</tr>
<tr>
<td>in</td>
<td>rounding_policy</td>
<td>A vx_round_policy_e enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The output image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.43 Remap

3.43.1 Detailed Description

Maps output pixels in an image from input pixels in an image.

Remap takes a remap table object `vx_remap` to map a set of output pixels back to source input pixels. A remap is typically defined as:

\[
\text{output}(x,y) = \text{input}(\text{mapx}(x,y), \text{mapy}(x,y));
\]

for every \((x,y)\) in the destination image.

However, the mapping functions are contained in the `vx_remap` object.

Functions

- **vx_node VX_API_CALL vxRemapNode (vx_graph graph, vx_image input, vx_remap table, vx_enum policy, vx_image output)**
  
  [Graph] Creates a Remap Node.

- **vx_status VX_API_CALL vxuRemap (vx_context context, vx_image input, vx_remap table, vx_enum policy, vx_image output)**
  
  [Immediate] Remaps an output image from an input image.

3.43.2 Function Documentation

`vx_node VX_API_CALL vxRemapNode ( vx_graph graph, vx_image input, vx_remap table, vx_enum policy, vx_image output )`

[Graph] Creates a Remap Node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph that will contain the node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input <code>VX_DF_IMAGE_U8</code> image.</td>
</tr>
<tr>
<td>in</td>
<td>table</td>
<td>The remap table object.</td>
</tr>
<tr>
<td>in</td>
<td>policy</td>
<td>An interpolation type from <code>vx_interpolation_type_e.VX_INTERPOLATION_AREA</code> is not supported.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output <code>VX_DF_IMAGE_U8</code> image with the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

Note

The border modes `VX_NODE_BORDER` value `VX_BORDER_UNDEFINED` and `VX_BORDER_CONSTANT` are supported.

Returns

A `vx_node`.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using `vxGetStatus`. |

`vx_status VX_API_CALL vxuRemap ( vx_context context, vx_image input, vx_remap table, vx_enum policy, vx_image output )`

[Immediate] Remaps an output image from an input image.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>context</strong></td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>input</strong></td>
<td>The input VX_DF_IMAGE_U8 image.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>table</strong></td>
<td>The remap table object.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>policy</strong></td>
<td>The interpolation policy from vx_interpolation_type_e. VX_INTERPOLATION_AREA is not supported.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td><strong>output</strong></td>
<td>The output VX_DF_IMAGE_U8 image.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.
3.44 Scale Image

3.44.1 Detailed Description

Implements the Image Resizing Kernel. This kernel resizes an image from the source to the destination dimensions. The supported interpolation types are currently:

- VX_INTERPOLATION_NEAREST_NEIGHBOR
- VX_INTERPOLATION_AREA
- VX_INTERPOLATION_BILINEAR

The sample positions used to determine output pixel values are generated by scaling the outside edges of the source image pixels to the outside edges of the destination image pixels. As described in the documentation for \texttt{vx_interpolation_type_e}, samples are taken at pixel centers. This means that, unless the scale is 1:1, the sample position for the top left destination pixel typically does not fall exactly on the top left source pixel but will be generated by interpolation.

That is, the sample positions corresponding in source and destination are defined by the following equations:

\[
\begin{align*}
    x_{\text{input}} &= \left( x_{\text{output}} + 0.5 \right) \frac{\text{width}_{\text{input}}}{\text{width}_{\text{output}}} - 0.5 \\
    y_{\text{input}} &= \left( y_{\text{output}} + 0.5 \right) \frac{\text{height}_{\text{input}}}{\text{height}_{\text{output}}} - 0.5 \\
    x_{\text{output}} &= \left( x_{\text{input}} + 0.5 \right) \frac{\text{width}_{\text{output}}}{\text{width}_{\text{input}}} - 0.5 \\
    y_{\text{output}} &= \left( y_{\text{input}} + 0.5 \right) \frac{\text{height}_{\text{output}}}{\text{height}_{\text{input}}} - 0.5
\end{align*}
\]

- For VX_INTERPOLATION_NEAREST_NEIGHBOR, the output value is that of the pixel whose centre is closest to the sample point.

- For VX_INTERPOLATION_BILINEAR, the output value is formed by a weighted average of the nearest source pixels to the sample point. That is:

\[
\begin{align*}
    &x_{\text{lower}} = \lfloor x_{\text{input}} \rfloor \\
    &y_{\text{lower}} = \lfloor y_{\text{input}} \rfloor \\
    &s = x_{\text{input}} - x_{\text{lower}} \\
    &t = y_{\text{input}} - y_{\text{lower}} \\
    &\text{output}(x_{\text{input}}, y_{\text{input}}) = (1 - s)(1 - t) \ast \text{input}(x_{\text{lower}}, y_{\text{lower}}) + s(1 - t) \ast \text{input}(x_{\text{lower}} + 1, y_{\text{lower}}) \\
    &\quad + (1 - s)t \ast \text{input}(x_{\text{lower}}, y_{\text{lower}} + 1) + s \ast t \ast \text{input}(x_{\text{lower}} + 1, y_{\text{lower}} + 1)
\end{align*}
\]

- For VX_INTERPOLATION_AREA, the implementation is expected to generate each output pixel by sampling all the source pixels that are at least partly covered by the area bounded by:

\[
\begin{align*}
    &\left( x_{\text{output}} \ast \frac{\text{width}_{\text{input}}}{\text{width}_{\text{output}}} \right) - 0.5, \left( y_{\text{output}} \ast \frac{\text{height}_{\text{input}}}{\text{height}_{\text{output}}} \right) - 0.5
\end{align*}
\]

and

\[
\begin{align*}
    &\left( (x_{\text{output}} + 1) \ast \frac{\text{width}_{\text{input}}}{\text{width}_{\text{output}}} \right) - 0.5, \left( (y_{\text{output}} + 1) \ast \frac{\text{height}_{\text{input}}}{\text{height}_{\text{output}}} \right) - 0.5
\end{align*}
\]

The details of this sampling method are implementation-defined. The implementation should perform enough sampling to avoid aliasing, but there is no requirement that the sample areas for adjacent output pixels be disjoint, nor that the pixels be weighted evenly.
The above diagram shows three sampling methods used to shrink a 7x3 image to 3x1.

The topmost image pair shows nearest-neighbor sampling, with crosses on the left image marking the sample positions in the source that are used to generate the output image on the right. As the pixel centre closest to the sample position is white in all cases, the resulting 3x1 image is white.

The middle image pair shows bilinear sampling, with black squares on the left image showing the region in the source being sampled to generate each pixel on the destination image on the right. This sample area is always the size of an input pixel. The outer destination pixels partly sample from the outermost green pixels, so their resulting value is a weighted average of white and green.

The bottom image pair shows area sampling. The black rectangles in the source image on the left show the bounds of the projection of the destination pixels onto the source. The destination pixels on the right are formed by averaging at least those source pixels whose areas are wholly or partly contained within those rectangles. The manner of this averaging is implementation-defined; the example shown here weights the contribution of each source pixel by the amount of that pixel’s area contained within the black rectangle.

Functions

- vx_node VX_API_CALL vxHalfScaleGaussianNode (vx_graph graph, vx_image input, vx_image output, vx_int32 kernel_size)
  [Graph] Performs a Gaussian Blur on an image then half-scales it. The interpolation mode used is nearest-neighbor.
- vx_node VX_API_CALL vxScaleImageNode (vx_graph graph, vx_image src, vx_image dst, vx_enum type)
  [Graph] Creates a Scale Image Node.
- vx_status VX_API_CALL vxuHalfScaleGaussian (vx_context context, vx_image input, vx_image output, vx_int32 kernel_size)
  [Immediate] Performs a Gaussian Blur on an image then half-scales it. The interpolation mode used is nearest-neighbor.
- vx_status VX_API_CALL vxuScaleImage (vx_context context, vx_image src, vx_image dst, vx_enum type)
  [Immediate] Scales an input image to an output image.
3.44.2 Function Documentation

vx_node VX_API_CALL vxScaleImageNode ( vx_graph graph, vx_image src, vx_image dst, vx_enum type )

[Graph] Creates a Scale Image Node.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in src</td>
<td>The source image of type VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>out dst</td>
<td>The destination image of type VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>in type</td>
<td>The interpolation type to use.</td>
</tr>
</tbody>
</table>

See also
vx_interpolation_type_e.

Note
The destination image must have a defined size and format. The border modes VX_NODE_BORDER value VX_BORDER_UNDEFINED, VX_BORDER_REPLICATE and VX_BORDER_CONSTANT are supported.

Returns
vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

vx_node VX_API_CALL vxHalfScaleGaussianNode ( vx_graph graph, vx_image input, vx_image output, vx_int32 kernel_size )

[Graph] Performs a Gaussian Blur on an image then half-scales it. The interpolation mode used is nearest-neighbor.

The output image size is determined by:

\[ W_{output} = \frac{W_{input} + 1}{2}, H_{output} = \frac{H_{input} + 1}{2} \]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in input</td>
<td>The input VX_DF_IMAGE_U8 image.</td>
</tr>
<tr>
<td>out output</td>
<td>The output VX_DF_IMAGE_U8 image.</td>
</tr>
<tr>
<td>in kernel_size</td>
<td>The input size of the Gaussian filter. Supported values are 1, 3 and 5.</td>
</tr>
</tbody>
</table>

Returns
vx_node.
Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

vx_status VX_API_CALL vxuScaleImage ( vx_context context, vx_image src, vx_image dst, vx_enum type )

[Immediate] Scales an input image to an output image.

Parameters

| in context | The reference to the overall context. |
| in src | The source image of type VX_DF_IMAGE_U8. |
| out dst | The destination image of type VX_DF_IMAGE_U8. |
| in type | The interpolation type. |

See also

vx_interpolation_type_e.

Returns

A vx_status_e enumeration.

Return values

| VX_SUCCESS | Success |
| * | An error occurred. See vx_status_e. |

vx_status VX_API_CALL vxuHalfScaleGaussian ( vx_context context, vx_image input, vx_image output, vx_int32 kernel_size )

[Immediate] Performs a Gaussian Blur on an image then half-scales it. The interpolation mode used is nearest-neighbor.

Parameters

| in context | The reference to the overall context. |
| in input | The input VX_DF_IMAGE_U8 image. |
| out output | The output VX_DF_IMAGE_U8 image. |
| in kernel_size | The input size of the Gaussian filter. Supported values are 1, 3 and 5. |

Returns

A vx_status_e enumeration.

Return values

| VX_SUCCESS | Success |
| * | An error occurred. See vx_status_e. |
3.45 Sobel 3x3

3.45.1 Detailed Description

Implements the Sobel Image Filter Kernel. The output images dimensions should be the same as the dimensions of the input image.

This kernel produces two output planes (one can be omitted) in the x and y plane. The Sobel Operators $G_x, G_y$ are defined as:

$$G_x = \begin{vmatrix}
-1 & 0 & +1 \\
-2 & 0 & +2 \\
-1 & 0 & +1 \\
\end{vmatrix}, \\
G_y = \begin{vmatrix}
-1 & -2 & -1 \\
0 & 0 & 0 \\
+1 & +2 & +1 \\
\end{vmatrix}$$

Functions

- **vx_node VX_API_CALL vxSobel3x3Node (vx_graph graph, vx_image input, vx_image output_x, vx_image output_y)**  
  [Graph] Creates a Sobel3x3 node.

- **vx_status VX_API_CALL vxuSobel3x3 (vx_context context, vx_image input, vx_image output_x, vx_image output_y)**  
  [Immediate] Invokes an immediate Sobel 3x3.

3.45.2 Function Documentation

**vx_node VX_API_CALL vxSobel3x3Node (vx_graph graph, vx_image input, vx_image output_x, vx_image output_y)**  
[Graph] Creates a Sobel3x3 node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output_x</td>
<td>[optional] The output gradient in the x direction in VX_DF_IMAGE_S16. Must have the same dimensions as the input image.</td>
</tr>
<tr>
<td>out</td>
<td>output_y</td>
<td>[optional] The output gradient in the y direction in VX_DF_IMAGE_S16. Must have the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

See also

VX_KERNEL_SOBEL_3x3

Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

vx_status VX_API_CALL vxuSobel3x3 (vx_context context, vx_image input, vx_image output_x, vx_image output_y)  
[Immediate] Invokes an immediate Sobel 3x3.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output-&gt;_x</td>
<td>[optional] The output gradient in the x direction in VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>out</td>
<td>output-&gt;_y</td>
<td>[optional] The output gradient in the y direction in VX_DF_IMAGE_S16.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.46 TableLookup

3.46.1 Detailed Description

Implements the Table Lookup Image Kernel. The output image dimensions should be the same as the dimensions of the input image.

This kernel uses each pixel in an image to index into a LUT and put the indexed LUT value into the output image. The formats supported are VX_DF_IMAGE_U8 and VX_DF_IMAGE_S16.

Functions

- vx_node VX_API_CALL vxTableLookupNode (vx_graph graph, vx_image input, vx_lut lut, vx_image output)
  [Graph] Creates a Table Lookup node. If a value from the input image is not present in the lookup table, the result is undefined.

- vx_status VX_API_CALL vxuTableLookup (vx_context context, vx_image input, vx_lut lut, vx_image output)
  [Immediate] Processes the image through the LUT.

3.46.2 Function Documentation

vx_node VX_API_CALL vxTableLookupNode (vx_graph graph, vx_image input, vx_lut lut, vx_image output)

[Graph] Creates a Table Lookup node. If a value from the input image is not present in the lookup table, the result is undefined.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>in</td>
<td>lut</td>
<td>The LUT which is of type VX_TYPE_UINT8 if input image is VX_DF_IMAGE_U8 or VX_TYPE_INT16 if input image is VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image of the same type and size as the input image.</td>
</tr>
</tbody>
</table>

Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus. |

vx_status VX_API_CALL vxuTableLookup (vx_context context, vx_image input, vx_lut lut, vx_image output)

[Immediate] Processes the image through the LUT.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>in</td>
<td>lut</td>
<td>The LUT which is of type VX_TYPE_UINT8 if input image is VX_DF_IMAGE_U8 or VX_TYPE_INT16 if input image is VX_DF_IMAGE_S16.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image of the same type as the input image.</td>
</tr>
</tbody>
</table>
 Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>* An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.47 Thresholding

3.47.1 Detailed Description

Thresholds an input image and produces an output Boolean image. The output image dimensions should be the same as the dimensions of the input image.

In `VX_THRESHOLD_TYPE_BINARY`, the output is determined by:

\[
dst(x, y) = \begin{cases} 
    \text{true value} & \text{if } src(x, y) > \text{threshold} \\
    \text{false value} & \text{otherwise}
\end{cases}
\]

In `VX_THRESHOLD_TYPE_RANGE`, the output is determined by:

\[
dst(x, y) = \begin{cases} 
    \text{false value} & \text{if } src(x, y) > \text{upper} \\
    \text{false value} & \text{if } src(x, y) < \text{lower} \\
    \text{true value} & \text{otherwise}
\end{cases}
\]

Where 'false value' and 'true value' are defined by the of the `thresh` parameter dependent upon the threshold output format with default values as discussed in the description of `vxCreateThresholdForImage` or as set by a call to `vxCopyThresholdOutput` with the `thresh` parameter as the first argument.

Functions

- `vx_node VX_API_CALL vxThresholdNode (vx_graph graph, vx_image input, vx_threshold thresh, vx_image output)`
  
  [Graph] Creates a Threshold node and returns a reference to it.

- `vx_status VX_API_CALL vxuThreshold (vx_context context, vx_image input, vx_threshold thresh, vx_image output)`
  
  [Immediate] Threshold's an input image and produces a `VX_DF_IMAGE_U8` boolean image.

3.47.2 Function Documentation

`vx_node VX_API_CALL vxThresholdNode ( vx_graph graph, vx_image input, vx_threshold thresh, vx_image output )`

[Graph] Creates a Threshold node and returns a reference to it.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph in which the node is created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image. Only images with format <code>VX_DF_IMAGE_U8</code> and <code>VX_DF_IMAGE_S16</code> are supported.</td>
</tr>
<tr>
<td>in</td>
<td>thresh</td>
<td>The thresholding object that defines the parameters of the operation. The <code>VX_THRESHOLD_INPUT_FORMAT</code> must be the same as the input image format and the <code>VX_THRESHOLD_OUTPUT_FORMAT</code> must be the same as the output image format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image, that will contain as pixel value true and false values defined by <code>thresh</code>. Only images with format <code>VX_DF_IMAGE_U8</code> are supported. The dimensions are the same as the input image.</td>
</tr>
</tbody>
</table>

Returns

`vx_node`.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using `vxGetStatus`. |
vx_status VX_API_CALL vxuThreshold ( vx_context context, vx_image input, vx_threshold thresh, vx_image output )

[Immediate] Threshold's an input image and produces a VX_DF_IMAGE_U8 boolean image.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>input</td>
<td>The input image. Only images with format VX_DF_IMAGE_U8 and VX_DF_IMAGE_S16 are supported.</td>
</tr>
<tr>
<td></td>
<td>thresh</td>
<td>The thresholding object that defines the parameters of the operation. The VX_THRESHOLD_INPUT_FORMAT must be the same as the input image format and the VX_THRESHOLD_OUTPUT_FORMAT must be the same as the output image format.</td>
</tr>
<tr>
<td></td>
<td>output</td>
<td>The output image, that will contain as pixel value true and false values defined by thresh. Only images with format VX_DF_IMAGE_U8 are supported.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
### 3.48 Warp Affine

#### 3.48.1 Detailed Description

Performs an affine transform on an image.

This kernel performs an affine transform with a 2x3 Matrix $M$ with this method of pixel coordinate translation:

\[
\begin{align*}
x_0 &= M_{1,1}x + M_{1,2}y + M_{1,3} \\
y_0 &= M_{2,1}x + M_{2,2}y + M_{2,3}
\end{align*}
\]

This translates into the C declaration:

```c
// x0 = a x + b y + c;
// y0 = d x + e y + f;
vx_float32 mat[3][2] = {
    {a, d}, // 'x' coefficients
    {b, e}, // 'y' coefficients
    {c, f}, // 'offsets'
};
vx_matrix matrix = vxCreateMatrix(context,
    VX_TYPE_FLOAT32, 2, 3);
vxCopyMatrix(matrix, mat, VX_WRITE_ONLY,
    VX_MEMORY_TYPE_HOST);
```

#### Functions

- **vx_status VX_API_CALL vxuWarpAffine (vx_context context, vx_image input, vx_matrix matrix, vx_enum type, vx_image output)**
  
  [Immediate] Performs an Affine warp on an image.

- **vx_node VX_API_CALL vxWarpAffineNode (vx_graph graph, vx_image input, vx_matrix matrix, vx_enum type, vx_image output)**
  
  [Graph] Creates an Affine Warp Node.

### 3.48.2 Function Documentation

**vx_node VX_API_CALL vxWarpAffineNode ( vx_graph graph, vx_image input, vx_matrix matrix, vx_enum type, vx_image output )**

[Graph] Creates an Affine Warp Node.

#### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>graph</td>
</tr>
<tr>
<td>in</td>
<td>input</td>
</tr>
<tr>
<td>in</td>
<td>matrix</td>
</tr>
<tr>
<td>in</td>
<td>type</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
</tr>
</tbody>
</table>

#### Note

The border modes VX_NODE_BORDER value VX_BORDER_UNDEFINED and VX_BORDER_CONSTANT are supported.

#### Returns

vx_node.
Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

vx_status VX_API_CALL vxuWarpAffine ( vx_context context, vx_image input, vx_matrix matrix, vx_enum type, vx_image output )

[Immediate] Performs an Affine warp on an image.

Parameters

| in  | context | The reference to the overall context. |
| in  | input   | The input VX_DF_IMAGE_U8 image.      |
| in  | matrix  | The affine matrix. Must be 2x3 of type VX_TYPE_FLOAT32. |
| in  | type    | The interpolation type from vx_interpolation_type_e. VX_INTERPOLATION_AREA is not supported. |
| out | output  | The output VX_DF_IMAGE_U8 image.     |

Returns

A vx_status_e enumeration.

Return values

| VX_SUCCESS | Success |
| *           | An error occurred. See vx_status_e. |
3.49 Warp Perspective

3.49.1 Detailed Description

Performs a perspective transform on an image.

This kernel performs an perspective transform with a 3x3 Matrix $M$ with this method of pixel coordinate translation:

\[
\begin{align*}
x_0 &= M_{1,1}x + M_{1,2}y + M_{1,3} \\
y_0 &= M_{2,1}x + M_{2,2}y + M_{2,3} \\
z_0 &= M_{3,1}x + M_{3,2}y + M_{3,3}
\end{align*}
\]

\[\text{output}(x, y) = \text{input}(\frac{x_0}{z_0}, \frac{y_0}{z_0})\]

This translates into the C declaration:

```c
// x0 = a x + b y + c;
// y0 = d x + e y + f;
// z0 = g x + h y + i;

vx_float32 mat[3][3] = {
    {a, d, g}, // 'x' coefficients
    {b, e, h}, // 'y' coefficients
    {c, f, i}, // 'offsets'
};

vx_matrix matrix = vxCreateMatrix(context,
    VX_TYPE_FLOAT32, 3, 3);

vxCopyMatrix(matrix, mat, VX_WRITE_ONLY,
    VX_MEMORY_TYPE_HOST);
```

Functions

- **vx_status VX_API_CALL vxuWarpPerspective (vx_context context, vx_image input, vx_matrix matrix, vx_enum type, vx_image output)**

  [Immediate] Performs an Perspective warp on an image.

- **vx_node VX_API_CALL vxWarpPerspectiveNode (vx_graph graph, vx_image input, vx_matrix matrix, vx_enum type, vx_image output)**

  [Graph] Creates a Perspective Warp Node.

3.49.2 Function Documentation

**vx_node VX_API_CALL vxWarpPerspectiveNode (vx_graph graph, vx_image input, vx_matrix matrix, vx_enum type, vx_image output)**

[Graph] Creates a Perspective Warp Node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input VX_DF_IMAGE_U8 image.</td>
</tr>
<tr>
<td>in</td>
<td>matrix</td>
<td>The perspective matrix. Must be 3x3 of type VX_TYPE_FLOAT32.</td>
</tr>
<tr>
<td>in</td>
<td>type</td>
<td>The interpolation type from vx_interpolation_type_e. VX_INTERPOLATION_AREA is not supported.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output VX_DF_IMAGE_U8 image with the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

Note

The border modes VX_NODE_BORDER value VX_BORDER_UNDEFINED and VX_BORDER_CONSTANT are supported.
Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

vx_status VX_API_CALL vxuWarpPerspective ( vx_context context, vx_image input, vx_matrix matrix, vx_enum type, vx_image output )

[Immediate] Performs an Perspective warp on an image.

Parameters

| in  | context | The reference to the overall context. |
| in  | input   | The input VX_DF_IMAGE_U8 image.       |
| in  | matrix  | The perspective matrix. Must be 3x3 of type VX_TYPE_FLOAT32. |
| in  | type    | The interpolation type from vx_interpolation_type_e. VX_INTERPOLATION_AREA is not supported. |
| out | output  | The output VX_DF_IMAGE_U8 image.      |

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
</table>

* An error occurred. See vx_status_e.
3.50 Bilateral Filter

3.50.1 Detailed Description

The function applies bilateral filtering to the input tensor.

A bilateral filter is a non-linear, edge-preserving and noise-reducing smoothing filter. The input and output are tensors with the same dimensions and data type. The tensor dimensions are divided into spatial and non-spatial dimensions. The spatial dimensions are isometric distance which is Cartesian. And they are the last 2. The non-spatial dimension is the first, and we call him the radiometric. The radiometric value at each spatial position is replaced by a weighted average of radiometric values from nearby pixels. This weight can be based on a Gaussian distribution. Crucially, the weights depend not only on Euclidean distance of spatial dimensions, but also on the radiometric differences (e.g. range differences, such as color intensity, depth distance, etc.). This preserves sharp edges by systematically looping through each pixel and adjusting weights to the adjacent pixels accordingly. The equations are as follows:

\[
h(x, \tau) = \sum f(y, t) g_1(y-x) g_2(t-\tau) dydt
\]

\[
g_1 = \frac{1}{\sqrt{2\pi \sigma_y}} \exp \left( -\frac{1}{2} \left( \frac{y^2}{\sigma_y^2} \right) \right)
\]

\[
g_2(t) = \frac{1}{\sqrt{2\pi \sigma_t}} \exp \left( -\frac{1}{2} \left( \frac{t^2}{\sigma_t^2} \right) \right)
\]

where \( x, y \) are in the spatial euclidean space. \( t, \tau \) are vectors in radiometric space. Can be color, depth or movement. In case of 3 dimensions the 1st dimension of the \( \text{vx\_tensor} \). Which can be of size 1 or 2. Or the value in the tensor in the case of tensor with 2 dimensions.

Functions

- \text{vx\_node VX\_API\_CALL vxBilateralFilterNode (vx\_graph graph, vx\_tensor src, vx\_int32 diameter, vx\_float32 sigmaSpace, vx\_float32 sigmaValues, vx\_tensor dst)}
  
  [Graph] The function applies bilateral filtering to the input tensor.

- \text{vx\_status VX\_API\_CALL vxuBilateralFilter (vx\_context context, vx\_tensor src, vx\_int32 diameter, vx\_float32 sigmaSpace, vx\_float32 sigmaValues, vx\_tensor dst)}
  
  [Immediate] The function applies bilateral filtering to the input tensor.

3.50.2 Function Documentation

\text{vx\_node VX\_API\_CALL vxBilateralFilterNode ( vx\_graph graph, vx\_tensor src, vx\_int32 diameter, vx\_float32 sigmaSpace, vx\_float32 sigmaValues, vx\_tensor dst )}

[Graph] The function applies bilateral filtering to the input tensor.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in src</td>
<td>The input data a \text{vx_tensor}. maximum 3 dimension and minimum 2. The tensor is of type \text{VX_TYPE_UINT8} or \text{VX_TYPE_INT16}. dimensions are [radiometric, width, height] or [width, height]. See \text{vxCreateTensor} and \text{vxCreateVirtualTensor}.</td>
</tr>
<tr>
<td>in diameter</td>
<td>of each pixel neighbourhood that is used during filtering. Values of diameter must be odd. Bigger than 3 and smaller than 10.</td>
</tr>
<tr>
<td>in sigmaValues</td>
<td>Filter sigma in the radiometric space. Supported values are bigger than 0 and smaller or equal 20.</td>
</tr>
<tr>
<td>in sigmaSpace</td>
<td>Filter sigma in the spatial space. Supported values are bigger than 0 and smaller or equal 20.</td>
</tr>
<tr>
<td>out dst</td>
<td>The output data a \text{vx_tensor}. Of type \text{VX_TYPE_UINT8} or \text{VX_TYPE_INT16}. And must be the same type and size of the input.</td>
</tr>
</tbody>
</table>
Note
The border modes \texttt{VX\_NODE\_BORDER} value \texttt{VX\_BORDER\_REPLICATE} and \texttt{VX\_BORDER\_CONSTANT} are supported.

Returns
\texttt{vx\_node}.

Return values
\begin{verbatim}
\begin{tabular}{|l|l|}
\hline
\texttt{vx\_node} & A node reference. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus} \\
\hline
\end{tabular}
\end{verbatim}

\texttt{vx\_status} \texttt{VX\_API\_CALL} \texttt{vxuBilateralFilter ( context, src, diameter, sigmaSpace, sigmaValues, dst )}

[Immediate] The function applies bilateral filtering to the input tensor.

Parameters
\begin{verbatim}
\begin{tabular}{|l|l|
\hline
\texttt{context} & The reference to the overall context. \\
\hline
\texttt{src} & The input data a \texttt{vx\_tensor}. maximum 3 dimension and minimum 2. The tensor is of type \texttt{VX\_TYPE\_UINT8} or \texttt{VX\_TYPE\_INT16}. dimensions are \[\text{radiometric, width, height}\] or \[\text{width, height}\] \\
\hline
\texttt{diameter} & of each pixel neighbourhood that is used during filtering. Values of diameter must be odd. Bigger than 3 and smaller than 10. \\
\hline
\texttt{sigmaValues} & Filter sigma in the radiometric space. Supported values are bigger than 0 and smaller or equal 20. \\
\hline
\texttt{sigmaSpace} & Filter sigma in the spatial space. Supported values are bigger than 0 and smaller or equal 20. \\
\hline
\texttt{dst} & The output data a \texttt{vx\_tensor}. Of type \texttt{VX\_TYPE\_UINT8} or \texttt{VX\_TYPE\_INT16}. And must be the same type and size of the input. \\
\hline
\end{tabular}
\end{verbatim}

Note
The border modes \texttt{VX\_NODE\_BORDER} value \texttt{VX\_BORDER\_REPLICATE} and \texttt{VX\_BORDER\_CONSTANT} are supported.

Returns
A \texttt{vx\_status\_e} enumeration.

Return values
\begin{verbatim}
\begin{tabular}{|l|l|}
\hline
\texttt{VX\_SUCCESS} & Success \\
\hline
* & An error occurred. See \texttt{vx\_status\_e}. \\
\hline
\end{tabular}
\end{verbatim}
3.51 MatchTemplate

3.51.1 Detailed Description

Compares an image template against overlapped image regions.

The detailed equation to the matching can be found in \texttt{vx\_comp\_metric\_e}. The output of the template matching node is a comparison map. The output comparison map should be the same size as the input image. The template image size (width+height) shall not be larger than 65535. If the valid region of the template image is smaller than the entire template image, the result in the destination image is implementation-dependent.

Enumerations

- \texttt{enum vx\_comp\_metric\_e \{.vx\_compare\_hamming = ((\text{VX\_ID\_KHRONOS} << 20) | (\text{VX\_ENUM\_COMP\_METRIC} << 12)) + 0x0,
  vx\_compare\_l1 = ((\text{VX\_ID\_KHRONOS} << 20) | (\text{VX\_ENUM\_COMP\_METRIC} << 12)) + 0x1,
  vx\_compare\_l2 = ((\text{VX\_ID\_KHRONOS} << 20) | (\text{VX\_ENUM\_COMP\_METRIC} << 12)) + 0x2,
  vx\_compare\_ccorr = ((\text{VX\_ID\_KHRONOS} << 20) | (\text{VX\_ENUM\_COMP\_METRIC} << 12)) + 0x3,
  vx\_compare\_l2\_norm = ((\text{VX\_ID\_KHRONOS} << 20) | (\text{VX\_ENUM\_COMP\_METRIC} << 12)) + 0x4,
  vx\_compare\_ccorr\_norm = ((\text{VX\_ID\_KHRONOS} << 20) | (\text{VX\_ENUM\_COMP\_METRIC} << 12)) + 0x5 \}}}

comparing metrics.

Functions

- \texttt{vx\_node \text{VX\_API\_CALL} vxu\_MatchTemplateNode (vx\_graph graph, vx\_image src, vx\_image templateImage, \text{vx\_enum} matchingMethod, vx\_image output)}

  [Graph] The Node Compares an image template against overlapped image regions.

- \texttt{vx\_status \text{VX\_API\_CALL} vxu\_MatchTemplate (vx\_context context, vx\_image src, vx\_image templateImage, \text{vx\_enum} matchingMethod, vx\_image output)}

  [Immediate] The function compares an image template against overlapped image regions.

3.5.2 Enumeration Type Documentation

\texttt{enum vx\_comp\_metric\_e}

comparing metrics.

In all the equations below \( w \) and \( h \) are width and height of the template image respectively. \( R \) is the compare map. \( T \) is the template image. \( I \) is the image on which the template is searched.

Enumerator

- \texttt{VX\_COMPARE\_HAMMING} hamming distance \( R(x,y) = \frac{1}{w*h} \sum_{i,j} X\text{OR}(T(i,j),I(x+i,y+j)) \)

- \texttt{VX\_COMPARE\_L1} L1 distance \( R(x,y) = \frac{1}{w*h} \sum_{i,j} |T(i,j) - I(x+i,y+j)| \).

- \texttt{VX\_COMPARE\_L2} L2 distance normalized by image size \( R(x,y) = \frac{1}{w*h} \sum_{i,j} (T(i,j) - I(x+i,y+j))^2 \).

- \texttt{VX\_COMPARE\_CCORR} cross correlation distance \( R(x,y) = \frac{1}{w*h} \sum_{i,j} (T(i,j) * I(x+i,y+j)) \).

- \texttt{VX\_COMPARE\_L2\_NORM} L2 normalized distance \( R(x,y) = \frac{\sum_{i,j} (T(i,j) - I(x+i,y+j))^2}{\sqrt{\sum_{i,j} T(i,j)^2 * I(x+i,y+j)^2}} \).

- \texttt{VX\_COMPARE\_CCORR\_NORM} cross correlation normalized distance \( R(x,y) = \frac{\sum_{i,j} T(i,j) * I(x+i,y+j)^2}{\sqrt{\sum_{i,j} T(i,j)^2 * I(x+i,y+j)^2}} \).

Definition at line 1447 of file \texttt{vx\_types.h}. 
3.51.3 Function Documentation

**vx_node VX_API_CALL vxMatchTemplateNode ( vx_graph graph, vx_image src, vx_image templateImage, vx_enum matchingMethod, vx_image output )**

[Graph] The Node Compares an image template against overlapped image regions. The detailed equation to the matching can be found in vx_comp_metric_e. The output of the template matching node is a comparison map as described in vx_comp_metric_e. The Node have a limitation on the template image size (width\times height). It should not be larger then 65535. If the valid region of the template image is smaller than the entire template image, the result in the destination image is implementation-dependent.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in</td>
<td>src</td>
<td>The input image of type VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>in</td>
<td>templateImage</td>
<td>Searched template of type VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>in</td>
<td>matchingMethod</td>
<td>attribute specifying the comparison method vx_comp_metric_e. This function support only VX_COMPARE_CCORR_NORM and VX_COMPARE_L2.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>Map of comparison results. The output is an image of type VX_DF_IMAGE_S16</td>
</tr>
</tbody>
</table>

Returns

vx_node.

Return values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_node</td>
<td>A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus</td>
</tr>
</tbody>
</table>

**vx_status VX_API_CALL vxuMatchTemplate ( vx_context context, vx_image src, vx_image templateImage, vx_enum matchingMethod, vx_image output )**

[Immediate] The function compares an image template against overlapped image regions. The detailed equation to the matching can be found in vx_comp_metric_e. The output of the template matching node is a comparison map as described in vx_comp_metric_e. The Node have a limitation on the template image size (width\times height). It should not be larger then 65535. If the valid region of the template image is smaller than the entire template image, the result in the destination image is implementation-dependent.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>in</td>
<td>src</td>
<td>The input image of type VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>in</td>
<td>templateImage</td>
<td>Searched template of type VX_DF_IMAGE_U8.</td>
</tr>
<tr>
<td>in</td>
<td>matchingMethod</td>
<td>attribute specifying the comparison method vx_comp_metric_e. This function support only VX_COMPARE_CCORR_NORM and VX_COMPARE_L2.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>Map of comparison results. The output is an image of type VX_DF_IMAGE_S16</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.52 LBP

3.52.1 Detailed Description

Extracts LBP image from an input image. The output image dimensions should be the same as the dimensions of the input image.

The function calculates one of the following LBP descriptors: Local Binary Pattern, Modified Local Binary Pattern, or Uniform Local Binary Pattern.

Local binary pattern is defined as: Each pixel \((y,x)\) generate an 8 bit value describing the local binary pattern around the pixel, by comparing the pixel value with its 8 neighbours (selected neighbours of the 3x3 or 5x5 window).

We will define the pixels for the 3x3 neighbourhood as:

\[
\begin{align*}
    g_0 &= \text{SrcImg}[y-1,x-1] \\
    g_1 &= \text{SrcImg}[y-1,x] \\
    g_2 &= \text{SrcImg}[y-1,x+1] \\
    g_3 &= \text{SrcImg}[y,x+1] \\
    g_4 &= \text{SrcImg}[y+1,x+1] \\
    g_5 &= \text{SrcImg}[y+1,x] \\
    g_6 &= \text{SrcImg}[y+1,x-1] \\
    g_7 &= \text{SrcImg}[y,x-1] \\
    g_c &= \text{SrcImg}[y,x]
\end{align*}
\] (3.35)

and the pixels in a 5x5 neighbourhood as:

\[
\begin{align*}
    g_0 &= \text{SrcImg}[y-1,x-1] \\
    g_1 &= \text{SrcImg}[y-2,x] \\
    g_2 &= \text{SrcImg}[y-1,x+1] \\
    g_3 &= \text{SrcImg}[y,x+1] \\
    g_4 &= \text{SrcImg}[y+1,x+1] \\
    g_5 &= \text{SrcImg}[y+2,x] \\
    g_6 &= \text{SrcImg}[y+1,x-1] \\
    g_7 &= \text{SrcImg}[y,x-2] \\
    g_c &= \text{SrcImg}[y,x]
\end{align*}
\] (3.36)

We also define the sign difference function:

\[
    s(x) = \begin{cases} 
        1 & x \geq 0 \\
        0 & x < 0 
    \end{cases}
\] (3.37)

Using the above definitions. The LBP image is defined in the following equation:

\[
    \text{DstImg}[y,x] = \sum_{p=0}^{7} s(g_p - g_c)2^p
\]

For modified local binary pattern. Each pixel \((y,x)\) generate an 8 bit value describing the modified local binary pattern around the pixel, by comparing the average of 8 neighbour pixels with its 8 neighbours (5x5 window).

\[
\begin{align*}
    \text{Avg}[y,x] &= \frac{(\text{SrcImg}[y-2,x-2])}{8} \\
    &+ \frac{(\text{SrcImg}[y-2,x])}{8} \\
    &+ \frac{(\text{SrcImg}[y-2,x+2])}{8} \\
    &+ \frac{(\text{SrcImg}[y,x+2])}{8} \\
    &+ \frac{(\text{SrcImg}[y+2,x+2])}{8} \\
    &+ \frac{(\text{SrcImg}[y+2,x])}{8} \\
    &+ \frac{(\text{SrcImg}[y+2,x-2])}{8} \\
    &+ \frac{(\text{SrcImg}[y,x-2])}{8} + 1
\end{align*}
\] (3.38)
\[ DstImg[y, x] = \begin{cases} 
\sum_{p=0}^{7} s(g_p - g_c)2^p & U \leq 2 \\
9 & \text{otherwise} 
\end{cases} \]

The uniform LBP patterns refer to the patterns which have limited transition or discontinuities (smaller than 2 or equal) in the circular binary presentation.

For each pixel \((y, x)\) a value is generated, describing the transition around the pixel (If there are up to 2 transitions between 0 to 1 or 1 to 0). And an additional value for all other local binary pattern values. We can define the function that measure transition as:

\[ U = |s(g_7 - g_c) - s(g_0 - g_c)| + \sum_{p=1}^{7} |s(g_p - g_c) - s(g_{p-1} - g_c)| \]

With the above definitions, the unified LBP equation is defined as.

\[ DstImg[y, x] = \begin{cases} 
\sum_{p=0}^{7} s(g_p - g_c)2^p & U \leq 2 \\
9 & \text{otherwise} 
\end{cases} \]

Enumerations

- \text{enum vx\_lbp\_format\_e} {
  \text{VX\_LBP} = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_ENUM\_LBP\_FORMAT << 12)) + 0x0,
  \text{VX\_MLBP} = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_ENUM\_LBP\_FORMAT << 12)) + 0x1,
  \text{VX\_ULBP} = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_ENUM\_LBP\_FORMAT << 12)) + 0x2
}

Local binary pattern supported.

Functions

- \text{vx\_node VX\_API\_CALL vxLBPNode (vx\_graph graph, vx\_image in, vx\_enum format, vx\_int8 kernel\_size, vx\_image out)}
  
  [Graph] Creates a node that extracts LBP image from an input image

- \text{vx\_status VX\_API\_CALL vxuLBP (vx\_context context, vx\_image in, vx\_enum format, vx\_int8 kernel\_size, vx\_image out)}

  [Immediate] The function extracts LBP image from an input image

3.52.2 Enumeration Type Documentation

\text{enum vx\_lbp\_format\_e}

Local binary pattern supported.

Enumerator

- \text{VX\_LBP} local binary pattern
- \text{VX\_MLBP} Modified Local Binary Patterns.
- \text{VX\_ULBP} Uniform local binary pattern.

Definition at line 1428 of file \text{vx\_types.h}. 
3.52.3 Function Documentation

vx_node VX_API_CALL vxLBPNode ( vx_graph graph, vx_image in, vx_enum format, vx_int8 kernel_size, vx_image out )

[Graph] Creates a node that extracts LBP image from an input image

Parameters

| in   | graph          | The reference to the graph.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>in</td>
<td>An input image in vx_image. Or SrcImg in the equations. the image is of type VX_DF_IMAGE_U8</td>
</tr>
<tr>
<td>in</td>
<td>format</td>
<td>A variation of LBP like original LBP and mLBP. see vx_lbp_format_e</td>
</tr>
<tr>
<td>in</td>
<td>kernel_size</td>
<td>Kernel size. Only size of 3 and 5 are supported</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>An output image in vx_image. Or DstImg in the equations. the image is of type VX_DF_IMAGE_U8 with the same dimensions as the input image.</td>
</tr>
</tbody>
</table>

Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

vx_status VX_API_CALL vxuLBP ( vx_context context, vx_image in, vx_enum format, vx_int8 kernel_size, vx_image out )

[Immediate] The function extracts LBP image from an input image

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>in</td>
<td>An input image in vx_image. Or SrcImg in the equations. the image is of type VX_DF_IMAGE_U8</td>
</tr>
<tr>
<td>in</td>
<td>format</td>
<td>A variation of LBP like original LBP and mLBP. see vx_lbp_format_e</td>
</tr>
<tr>
<td>in</td>
<td>kernel_size</td>
<td>Kernel size. Only size of 3 and 5 are supported</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>An output image in vx_image. Or DstImg in the equations. the image is of type VX_DF_IMAGE_U8</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.53 HOG

3.53.1 Detailed Description

Extracts Histogram of Oriented Gradients features from the input grayscale image.

The Histogram of Oriented Gradients (HOG) vision function is split into two nodes `vxHOGCellsNode` and `vxHOGFeaturesNode`. The specification of these nodes cover a subset of possible HOG implementations. The `vxHOGCellsNode` calculates the gradient orientation histograms and average gradient magnitudes for each of the cells. The `vxHOGFeaturesNode` uses the cell histograms and optionally the average gradient magnitude of the cells to produce a HOG feature vector. This involves grouping up the cell histograms into blocks which are then normalized. A moving window is applied to the input image and for each location the block data associated with the window is concatenated to the HOG feature vector.

Data Structures

- `struct vx_hog_t`
  
  The HOG descriptor structure. More...

Functions

- `vx_node VX_API_CALL vxHOGCellsNode (vx_graph graph, vx_image input, vx_int32 cell_width, vx_int32 cell_height, vx_int32 num_bins, vx_tensor magnitudes, vx_tensor bins)`
  
  [Graph] Performs cell calculations for the average gradient magnitude and gradient orientation histograms.

- `vx_node VX_API_CALL vxHOGFeaturesNode (vx_graph graph, vx_image input, vx_tensor magnitudes, vx_tensor bins, const vx_hog_t *params, vx_size hog_param_size, vx_tensor features)`
  
  [Graph] The node produces HOG features for the W1xW2 window in a sliding window fashion over the whole input image. Each position produces a HOG feature vector.

- `vx_status VX_API_CALL vxuHOGCells (vx_context context, vx_image input, vx_int32 cell_width, vx_int32 cell_height, vx_int32 num_bins, vx_tensor magnitudes, vx_tensor bins)`
  
  [Immediate] Performs cell calculations for the average gradient magnitude and gradient orientation histograms.

- `vx_status VX_API_CALL vxuHOGFeatures (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)`
  
  [Immediate] Computes Histogram of Oriented Gradients features for the W1xW2 window in a sliding window fashion over the whole input image.

3.53.2 Data Structure Documentation

`struct vx_hog_t`

The HOG descriptor structure.

Definition at line 1699 of file `vx_types.h`.

Data Fields

<table>
<thead>
<tr>
<th>Data Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vx_int32</code> cell_width</td>
<td>The histogram cell width of type <code>VX_TYPE_INT32</code>.</td>
</tr>
<tr>
<td><code>vx_int32</code> cell_height</td>
<td>The histogram cell height of type <code>VX_TYPE_INT32</code>.</td>
</tr>
<tr>
<td><code>vx_int32</code> block_width</td>
<td>The histogram block width of type <code>VX_TYPE_INT32</code>. Must be divisible by cell_width.</td>
</tr>
<tr>
<td><code>vx_int32</code> block_height</td>
<td>The histogram block height of type <code>VX_TYPE_INT32</code>. Must be divisible by cell_height.</td>
</tr>
<tr>
<td><code>vx_int32</code> block_stride</td>
<td>The histogram block stride within the window of type <code>VX_TYPE_INT32</code>. Must be an integral number of cell_width and cell_height.</td>
</tr>
<tr>
<td><code>vx_int32</code> num_bins</td>
<td>The histogram size of type <code>VX_TYPE_INT32</code>.</td>
</tr>
<tr>
<td><code>vx_int32</code> window_width</td>
<td>The feature descriptor window width of type <code>VX_TYPE_INT32</code>.</td>
</tr>
<tr>
<td><code>vx_int32</code> window_height</td>
<td>The feature descriptor window height of type <code>VX_TYPE_INT32</code>.</td>
</tr>
<tr>
<td><code>vx_int32</code> window_stride</td>
<td>The feature descriptor window stride of type <code>VX_TYPE_INT32</code>.</td>
</tr>
</tbody>
</table>
3.53.3 Function Documentation

\texttt{vx_node VX\_APL\_CALL vxHOGCellsNode ( vx\_graph graph, vx\_image input, vx\_int32 cell\_width, vx\_int32 cell\_height, vx\_int32 num\_bins, vx\_tensor magnitudes, vx\_tensor bins )}

[Graph] Performs cell calculations for the average gradient magnitude and gradient orientation histograms. Firstly, the gradient magnitude and gradient orientation are computed for each pixel in the input image. Two 1-D centred, point discrete derivative masks are applied to the input image in the horizontal and vertical directions.

\[
M_h = [-1,0,1]
\]

and

\[
M_v = [-1,0,1]^T
\]

\(G_v\) is the result of applying mask \(M_v\) to the input image, and \(G_h\) is the result of applying mask \(M_h\) to the input image. The border mode used for the gradient calculation is implementation dependent. Its behavior should be similar to \texttt{VX\_BORDER\_UNDEFINED}. The gradient magnitudes and gradient orientations for each pixel are then calculated in the following manner.

\[
G(x,y) = \sqrt{G_v(x,y)^2 + G_h(x,y)^2}
\]

\[
\theta(x,y) = \arctan(G_v(x,y), G_h(x,y))
\]

where \(\arctan(v,h)\) is \(\tan^{-1}(v/h)\) when \(h! = 0\),

\[\begin{align*}
-\pi/2 & \text{ if } v < 0 \text{ and } h == 0, \\
\pi/2 & \text{ if } v > 0 \text{ and } h == 0, \\
0 & \text{ if } v == 0 \text{ and } h == 0
\end{align*}\]

Secondly, the gradient magnitudes and orientations are used to compute the bins output tensor and optional magnitudes output tensor. These tensors are computed on a cell level where the cells are rectangular in shape. The magnitudes tensor contains the average gradient magnitude for each cell.

\[
magnitudes(c) = \frac{1}{(\text{cell\_width} \times \text{cell\_height})} \sum_{w=0}^{\text{cell\_width}} \sum_{h=0}^{\text{cell\_height}} G_c(w,h)
\]

where \(G_c\) is the gradient magnitudes related to cell \(c\). The bins tensor contains histograms of gradient orientations for each cell. The gradient orientations at each pixel range from 0 to 360 degrees. These are quantised into a set of histogram bins based on the num\_bins parameter. Each pixel votes for a specific cell histogram bin based on its gradient orientation. The vote itself is the pixel’s gradient magnitude.

\[
bins(c,n) = \sum_{w=0}^{\text{cell\_width}} \sum_{h=0}^{\text{cell\_height}} G_c(w,h) \times 1[B_c(w,h,num\_bins) == n]
\]

where \(B_c\) produces the histogram bin number based on the gradient orientation of the pixel at location \((w,h)\) in cell \(c\) based on the num\_bins and

\[1[B_c(w,h,num\_bins) == n]\]

is a delta-function with value 1 when \(B_c(w,h,num\_bins) == n\) or 0 otherwise.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{in graph}</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>\texttt{in input}</td>
<td>The input image of type \texttt{VX_DF_IMAGE_U8}.</td>
</tr>
<tr>
<td>\texttt{in cell_width}</td>
<td>The histogram cell width of type \texttt{VX_TYPE_INT32}.</td>
</tr>
<tr>
<td>\texttt{in cell_height}</td>
<td>The histogram cell height of type \texttt{VX_TYPE_INT32}.</td>
</tr>
<tr>
<td>\texttt{in num_bins}</td>
<td>The histogram size of type \texttt{VX_TYPE_INT32}.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>magnitudes</th>
<th>(Optional) The output average gradient magnitudes per cell of <code>vx_tensor</code> of type <code>VX_TYPE_INT16</code> of size <code>[floor(image_width/cell_width), floor(image_height/cell_height)]</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bins</td>
<td>The output gradient orientation histograms per cell of <code>vx_tensor</code> of type <code>VX_TYPE_INT16</code> of size <code>[floor(image_width/cell_width), floor(image_height/cell_height), num_bins]</code>.</td>
</tr>
</tbody>
</table>

Returns

`vx_node`

Return values

| `0` | Node could not be created. |
| `*` | Node handle. |

`vx_node VX_API_CALL vxHOGFeaturesNode ( vx_graph graph, vx_image input, vx_tensor magnitudes, vx_tensor bins, const vx_hog_t *params, vx_size hog_param_size, vx_tensor features )`

[Graph] The node produces HOG features for the W1xW2 window in a sliding window fashion over the whole input image. Each position produces a HOG feature vector. Firstly if a magnitudes tensor is provided the cell histograms in the bins tensor are normalised by the average cell gradient magnitudes.

\[
\text{bins}(c,n) = \frac{\text{bins}(c,n)}{\text{magnitudes}(c)}
\]

To account for changes in illumination and contrast the cell histograms must be locally normalized which requires grouping the cell histograms together into larger spatially connected blocks. Blocks are rectangular grids represented by three parameters: the number of cells per block, the number of pixels per cell, and the number of bins per cell histogram. These blocks typically overlap, meaning that each cell histogram contributes more than once to the final descriptor. To normalize a block its cell histograms \( h \) are grouped together to form a vector \( v = [h_1, h_2, h_3, \ldots, h_n] \). This vector is normalised using L2-Hys which means performing L2-norm on this vector; clipping the result (by limiting the maximum values of \( v \) to be threshold) and renormalizing again. If the threshold is equal to zero then L2-Hys normalization is not performed.

\[
L2\text{norm}(v) = \frac{v}{\sqrt{\|v\|_2^2 + \epsilon^2}}
\]

where \( \|v\|_k \) be its k-norm for \( k = 1, 2 \), and \( \epsilon \) be a small constant. For a specific window its HOG descriptor is then the concatenated vector of the components of the normalized cell histograms from all of the block regions contained in the window. The W1xW2 window starting position is at coordinates 0x0. If the input image has dimensions that are not an integer multiple of W1xW2 blocks with the specified stride, then the last positions that contain only a partial W1xW2 window will be calculated with the remaining part of the W1xW2 window padded with zeroes. The Window W1xW2 must also have a size so that it contains an integer number of cells, otherwise the node is not well-defined. The final output tensor will contain HOG descriptors equal to the number of windows in the input image. The output features tensor has 3 dimensions, given by:

\[
((\text{floor}(\text{ceil}_x) + 1) \times \text{floor}(\text{ceil}_y) + 1) \times \text{num_bins}
\]

See `vxCreateTensor` and `vxCreateVirtualTensor`. We recommend the output tensors always be virtual objects, with this node connected directly to the classifier. The output tensor will be very large, and using non-virtual tensors will result in a poorly optimized implementation. Merging of this node with a classifier node such as that described in the classifier extension will result in better performance. Notice that this node creation function has
more parameters than the corresponding kernel. Numbering of kernel parameters (required if you create this node using the generic interface) is explicitly specified here.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in graph</code></td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td><code>in input</code></td>
<td>The input image of type <code>VX_DF_IMAGE_U8</code>. (Kernel parameter #0)</td>
</tr>
<tr>
<td><code>in magnitudes</code></td>
<td>(Optional) The gradient magnitudes per cell of <code>vx_tensor</code> of type <code>VX_TYPE_INT16</code>. It is the output of <code>vxHOGCellsNode</code>. (Kernel parameter #1)</td>
</tr>
<tr>
<td><code>in bins</code></td>
<td>The gradient orientation histograms per cell of <code>vx_tensor</code> of type <code>VX_TYPE_INT16</code>. It is the output of <code>vxHOGCellsNode</code>. (Kernel parameter #2)</td>
</tr>
<tr>
<td><code>in params</code></td>
<td>The parameters of type <code>vx_hog_t</code>. (Kernel parameter #3)</td>
</tr>
<tr>
<td><code>in hog_param_size</code></td>
<td>Size of <code>vx_hog_t</code> in bytes. Note that this parameter is not counted as one of the kernel parameters.</td>
</tr>
<tr>
<td><code>out features</code></td>
<td>The output HOG features of <code>vx_tensor</code> of type <code>VX_TYPE_INT16</code>. (Kernel parameter #4)</td>
</tr>
</tbody>
</table>

Returns

`vx_node`.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Node could not be created.</td>
</tr>
<tr>
<td>∗</td>
<td>Node handle.</td>
</tr>
</tbody>
</table>

`vx_status VX_API_CALL vxuHOGCells ( vx_context context, vx_image input, vx_int32 cell_width, vx_int32 cell_height, vx_int32 num_bins, vx_tensor magnitudes, vx_tensor bins )`  

[Immediate] Performs cell calculations for the average gradient magnitude and gradient orientation histograms.

Firstly, the gradient magnitude and gradient orientation are computed for each pixel in the input image. Two 1-D centred, point discrete derivative masks are applied to the input image in the horizontal and vertical directions.

\[
M_h = [-1, 0, 1] \\
M_v = [-1, 0, 1]^T 
\]

\(G_v\) is the result of applying mask \(M_v\) to the input image, and \(G_h\) is the result of applying mask \(M_h\) to the input image. The border mode used for the gradient calculation is implementation dependent. Its behavior should be similar to `VX_BORDER_UNDEFINED`. The gradient magnitudes and gradient orientations for each pixel are then calculated in the following manner.

\[
G(x,y) = \sqrt{G_v(x,y)^2 + G_h(x,y)^2} \\
θ(x,y) = \arctan(G_v(x,y), G_h(x,y)) 
\]

where \(\arctan(v,h)\) is \(\tan^{-1}(v/h)\) when \(h! = 0\),

\(-\pi/2\) if \(v < 0\) and \(h == 0\),
\(\pi/2\) if \(v > 0\) and \(h == 0\)
and \(0\) if \(v == 0\) and \(h == 0\).

Secondly, the gradient magnitudes and orientations are used to compute the bins output tensor and optional magnitudes output tensor. These tensors are computed on a cell level where the cells are rectangular in shape. The magnitudes tensor contains the average gradient magnitude for each cell.

\[
magnitudes(c) = \frac{1}{(cell\_width*cell\_height)} \sum_{w=0}^{cell\_width} \sum_{h=0}^{cell\_height} G_c(w,h) 
\]
where $G_c$ is the gradient magnitudes related to cell $c$. The bins tensor contains histograms of gradient orientations for each cell. The gradient orientations at each pixel range from 0 to 360 degrees. These are quantised into a set of histogram bins based on the num_bins parameter. Each pixel votes for a specific cell histogram bin based on its gradient orientation. The vote itself is the pixel's gradient magnitude.

$$\text{bins}(c, n) = \sum_{w=0}^{\text{cell width}} \sum_{h=0}^{\text{cell height}} G_c(w, h) \cdot 1[B_c(w,h,\text{numBins}) == n]$$

where $B_c$ produces the histogram bin number based on the gradient orientation of the pixel at location $(w, h)$ in cell $c$ based on the num_bins and

$$1[B_c(w,h,\text{numBins}) == n]$$

is a delta-function with value 1 when $B_c(w,h,\text{numBins}) == n$ or 0 otherwise.

Parameters

| in context | The reference to the overall context. |
| in input  | The input image of type VX_DF_IMAGE_U8. |
| in cell_width | The histogram cell width of type VX_TYPE_INT32. |
| in cell_height | The histogram cell height of type VX_TYPE_INT32. |
| in num_bins | The histogram size of type VX_TYPE_INT32. |
| out magnitudes | The output average gradient magnitudes per cell of vx_tensor of type VX_TYPE_INT16 of size $[\text{floor}(\text{image width}/\text{cell width}), \text{floor}(\text{image height}/\text{cell height})]$. |
| out bins | The output gradient orientation histograms per cell of vx_tensor of type VX_TYPE_INT16 of size $[\text{floor}(\text{image width}/\text{cell width}), \text{floor}(\text{image height}/\text{cell height}), \text{numBins}]$. |

Returns

A vx_status_e enumeration.

Return values

| VX_SUCCESS | Success |
| * An error occurred. See vx_status_e. |

vx_status VX_API_CALL vxuHOGFeatures ( 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 )

[Immediate] Computes Histogram of Oriented Gradients features for the W1xW2 window in a sliding window fashion over the whole input image.

Firstly if a magnitudes tensor is provided the cell histograms in the bins tensor are normalised by the average cell gradient magnitudes.

$$\text{bins}(c, n) = \frac{\text{bins}(c, n)}{\text{magnitudes}(c)}$$

To account for changes in illumination and contrast the cell histograms must be locally normalized which requires grouping the cell histograms together into larger spatially connected blocks. Blocks are rectangular grids represented by three parameters: the number of cells per block, the number of pixels per cell, and the number of bins per cell histogram. These blocks typically overlap, meaning that each cell histogram contributes more than once to the final descriptor. To normalize a block its cell histograms $h$ are grouped together to form a vector $v = [h_1, h_2, h_3, \ldots, h_n]$. This vector is normalised using L2-Hys which means performing L2-norm on this vector; clipping the result (by limiting the maximum values of $v$ to be threshold) and renormalizing again. If the threshold is equal to zero then L2-Hys normalization is not performed.

$$L2norm(v) = \frac{v}{\sqrt{\|v\|^2 + \varepsilon^2}}$$
where $\|v\|_k$ be its k-norm for k=1, 2, and $\varepsilon$ be a small constant. For a specific window its HOG descriptor is then the concatenated vector of the components of the normalized cell histograms from all of the block regions contained in the window. The $W_1 \times W_2$ window starting position is at coordinates 0x0. If the input image has dimensions that are not an integer multiple of $W_1 \times W_2$ blocks with the specified stride, then the last positions that contain only a partial $W_1 \times W_2$ window will be calculated with the remaining part of the $W_1 \times W_2$ window padded with zeroes. The Window $W_1 \times W_2$ must also have a size so that it contains an integer number of cells, otherwise the node is not well-defined. The final output tensor will contain HOG descriptors equal to the number of windows in the input image. The output features tensor has 3 dimensions, given by:

$$
\left(\floor{(\text{image width} - \text{window width})/\text{window stride}} + 1, \\
\floor{(\text{image height} - \text{window height})/\text{window stride}} + 1, \\
\floor{(\text{window width} - \text{block width})/\text{block stride} + 1} \times \floor{(\text{window height} - \text{block height})/\text{block stride} + 1} \times
\left(\floor{(\text{block width} \times \text{block height})/((\text{cell width} \times \text{cell height})) \times \text{num bins}}\right)\right)
$$

See vxCreateTensor and vxCreateVirtualTensor. The output tensor from this function may be very large. For this reason, it is not recommended that this "immediate mode" version of the function be used. The preferred method to perform this function is as graph node with a virtual tensor as the output.

**Parameters**

- **in context**: The reference to the overall context.
- **in input**: The input image of type VX_DF_IMAGE_U8.
- **in magnitudes**: The average gradient magnitudes per cell of vx_tensor of type VX_TYPE_INT16. It is the output of vxuHOGCells.
- **in bins**: The gradient orientation histogram per cell of vx_tensor of type VX_TYPE_INT16. It is the output of vxuHOGCells.
- **in params**: The parameters of type vx_hog_t.
- **in hog_param_size**: Size of vx_hog_t in bytes.
- **out features**: The output HOG features of vx_tensor of type VX_TYPE_INT16.

**Returns**

A vx_status_e enumeration.

**Return values**

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.54 HoughLinesP

3.54.1 Detailed Description

Finds the Probabilistic Hough Lines detected in the input binary image.

The node implements the Progressive Probabilistic Hough Transform described in Matas, J. and Galambos, C. and Kittler, J.V., Robust Detection of Lines Using the Progressive Probabilistic Hough Transform. CVIU 78 1, pp 119-137 (2000) The linear Hough transform algorithm uses a two-dimensional array, called an accumulator, to detect the existence of a line described by $r = x\cos\theta + y\sin\theta$. The dimension of the accumulator equals the number of unknown parameters, i.e., two, considering quantized values of $r$ and $\theta$ in the pair $(r, \theta)$. For each pixel at $(x,y)$ and its neighborhood, the Hough transform algorithm determines if there is enough evidence of a straight line at that pixel. If so, it will calculate the parameters $(r, \theta)$ of that line, and then look for the accumulator's bin that the parameters fall into, and increment the value of that bin.

Algorithm Outline:

1. Check the input image; if it is empty then finish.
2. Update the accumulator with a single pixel randomly selected from the input image.
3. Remove the selected pixel from input image.
4. Check if the highest peak in the accumulator that was modified by the new pixel is higher than threshold. If not then goto 1.
5. Look along a corridor specified by the peak in the accumulator, and find the longest segment that either is continuous or exhibits a gap not exceeding a given threshold.
6. Remove the pixels in the segment from input image.
7. “Unvote” from the accumulator all the pixels from the line that have previously voted.
8. If the line segment is longer than the minimum length add it into the output list.
9. Goto 1 each line is stored in vx_line2d_t struct. Such that start_x<=end_x.

Data Structures

- struct vx_hough_lines_p_t
  
  Hough lines probability parameters. More...

Functions

- vx_node VX_API_CALL vxHoughLinesPNode (vx_graph graph, vx_image input, const vx_hough_lines_p_t *params, vx_array lines_array, vx_scalar num_lines)
  
  [Graph] Finds the Probabilistic Hough Lines detected in the input binary image, each line is stored in the output array as a set of points $(x1, y1, x2, y2)$.

- vx_status VX_API_CALL vxuHoughLinesP (vx_context context, vx_image input, const vx_hough_lines_p_t *params, vx_array lines_array, vx_scalar num_lines)
  
  [Immediate] Finds the Probabilistic Hough Lines detected in the input binary image, each line is stored in the output array as a set of points $(x1, y1, x2, y2)$.

3.54.2 Data Structure Documentation

struct vx_hough_lines_p_t

Hough lines probability parameters.

Definition at line 1552 of file vx_types.h.

Data Fields

<table>
<thead>
<tr>
<th>Data Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_float32</td>
<td>rho</td>
</tr>
<tr>
<td>vx_float32</td>
<td>theta</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

Data Fields

<table>
<thead>
<tr>
<th>vx_int32</th>
<th>threshold</th>
<th>The minimum number of intersections to detect a line.</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_int32</td>
<td>line_length</td>
<td>The minimum number of points that can form a line. Line segments shorter than that are rejected.</td>
</tr>
<tr>
<td>vx_int32</td>
<td>line_gap</td>
<td>The maximum allowed gap between points on the same line to link them.</td>
</tr>
<tr>
<td>vx_float32</td>
<td>theta_max</td>
<td>Optional restriction on theta. The max allowed value.</td>
</tr>
<tr>
<td>vx_float32</td>
<td>theta_min</td>
<td>Optional restriction on theta. The min allowed value.</td>
</tr>
</tbody>
</table>

3.54.3 Function Documentation

vx_node VX_API_CALL vxHoughLinesPNode ( vx_graph graph, vx_image input, const vx_hough_lines_p_t *params, vx_array lines_array, vx_scalar num_lines )

[Graph] Finds the Probabilistic Hough Lines detected in the input binary image, each line is stored in the output array as a set of points (x1, y1, x2, y2).

Some implementations of the algorithm may have a random or non-deterministic element. If the target application is in a safety-critical environment this should be borne in mind and steps taken in the implementation, the application or both to achieve the level of determinism required by the system design.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>graph handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>8 bit, single channel binary source image</td>
</tr>
<tr>
<td>in</td>
<td>params</td>
<td>parameters of the struct vx_hough_lines_p_t</td>
</tr>
<tr>
<td>out</td>
<td>lines_array</td>
<td>lines_array contains array of lines, see vx_line2d_t The order of lines in implementation dependent</td>
</tr>
<tr>
<td>out</td>
<td>num_lines</td>
<td>[optional] The total number of detected lines in image. Use a VX_TYPE_SIZE scalar</td>
</tr>
</tbody>
</table>

Returns

vx_node.

Return values

| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using vxGetStatus |

vx_status VX_API_CALL vxuHoughLinesP ( vx_context context, vx_image input, const vx_hough_lines_p_t *params, vx_array lines_array, vx_scalar num_lines )

[Immediate] Finds the Probabilistic Hough Lines detected in the input binary image, each line is stored in the output array as a set of points (x1, y1, x2, y2).

Some implementations of the algorithm may have a random or non-deterministic element. If the target application is in a safety-critical environment this should be borne in mind and steps taken in the implementation, the application or both to achieve the level of determinism required by the system design.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>8 bit, single channel binary source image</td>
</tr>
<tr>
<td>in</td>
<td>params</td>
<td>parameters of the struct vx_hough_lines_p_t</td>
</tr>
<tr>
<td>out</td>
<td>lines_array</td>
<td>lines_array contains array of lines, see vx_line2d_t The order of lines in implementation dependent</td>
</tr>
</tbody>
</table>
Parameters

| out  | num_lines | [optional] The total number of detected lines in image. Use a VX_TYPE_SIZE scalar |

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
### 3.55 Tensor Multiply

#### 3.55.1 Detailed Description

Performs element wise multiplications on element values in the input tensor data with a scale. Pixel-wise multiplication is performed between the pixel values in two tensors and a scalar floating-point number \( \text{scale} \). The scale with a value of \( 1/2^n \), where \( n \) is an integer and \( 0 \leq n \leq 15 \), and \( 1/255 \) (0x1.010102p-8 C99 float hex) must be supported. The support for other values of scale is not prohibited. Furthermore, for scale with a value of \( 1/255 \) the rounding policy of \( \text{VX\_ROUND\_POLICY\_TO\_NEAREST\_EVEN} \) must be supported whereas for the scale with value of \( 1/2^n \) the rounding policy of \( \text{VX\_ROUND\_POLICY\_TO\_ZERO} \) must be supported. The support of other rounding modes for any values of scale is not prohibited.

#### Functions

- \( \text{vx\_node VX\_API\_CALL vxTensorMultiplyNode (vx\_graph graph, vx\_tensor input1, vx\_tensor input2, vx\_scalar scale, vx\_enum overflow\_policy, vx\_enum rounding\_policy, vx\_tensor output) } \)
  - [Graph] Performs element wise multiplications on element values in the input tensor data with a scale.

- \( \text{vx\_status VX\_API\_CALL vxuTensorMultiply (vx\_context context, vx\_tensor input1, vx\_tensor input2, vx\_scalar scale, vx\_enum overflow\_policy, vx\_enum rounding\_policy, vx\_tensor output) } \)
  - [Immediate] Performs element wise multiplications on element values in the input tensor data with a scale.

#### 3.55.2 Function Documentation

\( \text{vx\_node VX\_API\_CALL vxTensorMultiplyNode ( vx\_graph graph, vx\_tensor input1, vx\_tensor input2, vx\_scalar scale, vx\_enum overflow\_policy, vx\_enum rounding\_policy, vx\_tensor output ) } \)

[Graph] Performs element wise multiplications on element values in the input tensor data with a scale.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The handle to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input1</td>
<td>Input tensor data.</td>
</tr>
<tr>
<td></td>
<td>Input tensor data.</td>
<td>Implementations must support input tensor data type VX_TYPE_INT16 with fixed_point_position 8, and tensor data types VX_TYPE_UINT8 and VX_TYPE_INT8, with fixed_point_position 0.</td>
</tr>
<tr>
<td>in</td>
<td>input2</td>
<td>Input tensor data.</td>
</tr>
<tr>
<td></td>
<td>Input tensor data.</td>
<td>The dimensions and sizes of input2 match those of input1, unless the vx_tensor of one or more dimensions in input2 is 1. In this case, those dimensions are treated as if this tensor was expanded to match the size of the corresponding dimension of input1, and data was duplicated on all terms in that dimension. After this expansion, the dimensions will be equal. The data type must match the data type of input1.</td>
</tr>
<tr>
<td>in</td>
<td>scale</td>
<td>A non-negative VX_TYPE_FLOAT32 multiplied to each product before overflow handling.</td>
</tr>
<tr>
<td>in</td>
<td>overflow_policy</td>
<td>A vx_convert_policy_e enumeration.</td>
</tr>
<tr>
<td>in</td>
<td>rounding_policy</td>
<td>A vx_round_policy_e enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output tensor data with the same dimensions as the input tensor data.</td>
</tr>
</tbody>
</table>

**Returns**

- \( \text{vx\_node} \)
  - A node reference \( \text{vx\_node} \). Any possible errors preventing a successful creation should be checked using \( \text{vxAxAGetStatus} \).

\( \text{vx\_status VX\_API\_CALL vxuTensorMultiply ( vx\_context context, vx\_tensor input1, vx\_tensor input2, vx\_scalar scale, vx\_enum overflow\_policy, vx\_enum rounding\_policy, vx\_tensor output ) } \)

[Immediate] Performs element wise multiplications on element values in the input tensor data with a scale.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input1</td>
<td>Input tensor data. Implementations must support input tensor data type VX_TYPE_INT16 with fixed_point_position 8, and tensor data types VX_TYPE_UINT8 and VX_TYPE_INT8, with fixed_point_position 0.</td>
</tr>
<tr>
<td>in</td>
<td>input2</td>
<td>Input tensor data. The dimensions and sizes of input2 match those of input1, unless the vx_tensor of one or more dimensions in input2 is 1. In this case, those dimensions are treated as if this tensor was expanded to match the size of the corresponding dimension of input1, and data was duplicated on all terms in that dimension. After this expansion, the dimensions will be equal. The data type must match the data type of input1.</td>
</tr>
<tr>
<td>in</td>
<td>scale</td>
<td>A non-negative VX_TYPE_FLOAT32 multiplied to each product before overflow handling.</td>
</tr>
<tr>
<td>in</td>
<td>overflow_policy</td>
<td>A vx_convert_policy_e enumeration.</td>
</tr>
<tr>
<td>in</td>
<td>rounding_policy</td>
<td>A vx_round_policy_e enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output tensor data with the same dimensions as the input tensor data.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.56 Tensor Add

3.56.1 Detailed Description
Performs arithmetic addition on element values in the input tensor data.

Functions

- **vx_node** VX_API_CALL vxTensorAddNode (vx_graph graph, vx_tensor input1, vx_tensor input2, vx_enum policy, vx_tensor output)

  [Graph] Performs arithmetic addition on element values in the input tensor data.

- **vx_status** VX_API_CALL vxuTensorAdd (vx_context context, vx_tensor input1, vx_tensor input2, vx_enum policy, vx_tensor output)

  [Immediate] Performs arithmetic addition on element values in the input tensor data.

3.56.2 Function Documentation

vx_node VX_API_CALL vxTensorAddNode (vx_graph graph, vx_tensor input1, vx_tensor input2, vx_enum policy, vx_tensor output)

[Graph] Performs arithmetic addition on element values in the input tensor data.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The handle to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input1</td>
<td>Input tensor data. Implementations must support input tensor data type VX_TYPE_INT16 with fixed_point_position 8, and tensor data types VX_TYPE_UINT8 and VX_TYPE_INT8, with fixed_point_position 0.</td>
</tr>
<tr>
<td>in</td>
<td>input2</td>
<td>Input tensor data. The dimensions and sizes of input2 match those of input1, unless the vx_tensor of one or more dimensions in input2 is 1. In this case, those dimensions are treated as if this tensor was expanded to match the size of the corresponding dimension of input1, and data was duplicated on all terms in that dimension. After this expansion, the dimensions will be equal. The data type must match the data type of Input1.</td>
</tr>
<tr>
<td>in</td>
<td>policy</td>
<td>A vx_convert_policy_e enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output tensor data with the same dimensions as the input tensor data.</td>
</tr>
</tbody>
</table>

Returns

vx_node.
A node reference vx_node. Any possible errors preventing a successful creation should be checked using vxGetStatus.

vx_status VX_API_CALL vxuTensorAdd (vx_context context, vx_tensor input1, vx_tensor input2, vx_enum policy, vx_tensor output)

[Immediate] Performs arithmetic addition on element values in the input tensor data.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input1</td>
<td>Input tensor data. Implementations must support input tensor data type VX_TYPE_INT16 with fixed_point_position 8, and tensor data types VX_TYPE_UINT8 and VX_TYPE_INT8, with fixed_point_position 0.</td>
</tr>
<tr>
<td>in</td>
<td>input2</td>
<td>Input tensor data. The dimensions and sizes of input2 match those of input1, unless the vx_tensor of one or more dimensions in input2 is 1. In this case, those dimensions are treated as if this tensor was expanded to match the size of the corresponding dimension of input1, and data was duplicated on all terms in that dimension. After this expansion, the dimensions will be equal. The data type must match the data type of Input1.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>policy</th>
<th>A vx_convert_policy_e enumeration.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>output</td>
<td>The output tensor data with the same dimensions as the input tensor data.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.57 Tensor Subtract

3.57.1 Detailed Description
Performs arithmetic subtraction on element values in the input tensor data.

Functions
- **vx_node VX_API_CALL vxTensorSubtractNode (vx_graph graph, vx_tensor input1, vx_tensor input2, vx_enum policy, vx_tensor output)**
  [Graph] Performs arithmetic subtraction on element values in the input tensor data.
- **vx_status VX_API_CALL vxuTensorSubtract (vx_context context, vx_tensor input1, vx_tensor input2, vx_enum policy, vx_tensor output)**
  [Immediate] Performs arithmetic subtraction on element values in the input tensor data.

3.57.2 Function Documentation

**vx_node VX_API_CALL vxTensorSubtractNode (vx_graph graph, vx_tensor input1, vx_tensor input2, vx_enum policy, vx_tensor output)**
[Graph] Performs arithmetic subtraction on element values in the input tensor data.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The handle to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input1</td>
<td>Input tensor data. Implementations must support input tensor data type VX_TYPE_INT16 with fixed_point_position 8, and tensor data types VX_TYPE_UINT8 and VX_TYPE_INT8, with fixed_point_position 0.</td>
</tr>
<tr>
<td>in</td>
<td>input2</td>
<td>Input tensor data. The dimensions and sizes of input2 match those of input1, unless the vx_tensor of one or more dimensions in input2 is 1. In this case, those dimensions are treated as if this tensor was expanded to match the size of the corresponding dimension of input1, and data was duplicated on all terms in that dimension. After this expansion, the dimensions will be equal. The data type must match the data type of Input1.</td>
</tr>
<tr>
<td>in</td>
<td>policy</td>
<td>A vx_convert_policy_e enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output tensor data with the same dimensions as the input tensor data.</td>
</tr>
</tbody>
</table>

Returns
**vx_node.**
A node reference vx_node. Any possible errors preventing a successful creation should be checked using vxGetStatus.

**vx_status VX_API_CALL vxuTensorSubtract (vx_context context, vx_tensor input1, vx_tensor input2, vx_enum policy, vx_tensor output)**
[Immediate] Performs arithmetic subtraction on element values in the input tensor data.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input1</td>
<td>Input tensor data. Implementations must support input tensor data type VX_TYPE_INT16 with fixed_point_position 8, and tensor data types VX_TYPE_UINT8 and VX_TYPE_INT8, with fixed_point_position 0.</td>
</tr>
<tr>
<td>in</td>
<td>input2</td>
<td>Input tensor data. The dimensions and sizes of input2 match those of input1, unless the vx_tensor of one or more dimensions in input2 is 1. In this case, those dimensions are treated as if this tensor was expanded to match the size of the corresponding dimension of input1, and data was duplicated on all terms in that dimension. After this expansion, the dimensions will be equal. The data type must match the data type of Input1.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>policy</th>
<th>A vx_convert_policy_e enumeration.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>output</td>
<td>The output tensor data with the same dimensions as the input tensor data.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.58 Tensor TableLookUp

3.58.1 Detailed Description

Performs LUT on element values in the input tensor data.

This kernel uses each element in a tensor to index into a LUT and put the indexed LUT value into the output tensor. The tensor types supported are VX_TYPE_UINT8 and VX_TYPE_INT16. Signed inputs are cast to unsigned before used as input indexes to the LUT.

Functions

- **vx_node VX_API_CALL vxTensorTableLookupNode (vx_graph graph, vx_tensor input1, vx_lut lut, vx_tensor output)**
  
  [Graph] Performs LUT on element values in the input tensor data.

- **vx_status VX_API_CALL vxuTensorTableLookup (vx_context context, vx_tensor input1, vx_lut lut, vx_tensor output)**
  
  [Immediate] Performs LUT on element values in the input tensor data.

3.58.2 Function Documentation

**vx_node VX_API_CALL vxTensorTableLookupNode (vx_graph graph, vx_tensor input1, vx_lut lut, vx_tensor output)**

[Graph] Performs LUT on element values in the input tensor data.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> graph</td>
<td>The handle to the graph.</td>
</tr>
<tr>
<td><strong>in</strong> input1</td>
<td>Input tensor data. Implementations must support input tensor data type VX_TYPE_INT16 with fixed_point_position 8, and tensor data types VX_TYPE_UINT8, with fixed_point_position 0.</td>
</tr>
<tr>
<td><strong>in</strong> lut</td>
<td>The look-up table to use, of type vx_lut. The elements of input1 are treated as unsigned integers to determine an index into the look-up table. The data type of the items in the look-up table must match that of the output tensor.</td>
</tr>
<tr>
<td><strong>out</strong> output</td>
<td>The output tensor data with the same dimensions as the input tensor data.</td>
</tr>
</tbody>
</table>

**Returns**

vx_node.

A node reference vx_node. Any possible errors preventing a successful creation should be checked using vxGetStatus.

**vx_status VX_API_CALL vxuTensorTableLookup (vx_context context, vx_tensor input1, vx_lut lut, vx_tensor output)**

[Immediate] Performs LUT on element values in the input tensor data.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td><strong>in</strong> input1</td>
<td>Input tensor data. Implementations must support input tensor data type VX_TYPE_INT16 with fixed_point_position 8, and tensor data types VX_TYPE_UINT8, with fixed_point_position 0.</td>
</tr>
<tr>
<td><strong>in</strong> lut</td>
<td>The look-up table to use, of type vx_lut. The elements of input1 are treated as unsigned integers to determine an index into the look-up table. The data type of the items in the look-up table must match that of the output tensor.</td>
</tr>
<tr>
<td><strong>out</strong> output</td>
<td>The output tensor data with the same dimensions as the input tensor data.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th><code>VX_SUCCESS</code></th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.59 Tensor Transpose

3.59.1 Detailed Description
Performs transpose on the input tensor.

Functions

- **vx_node VX_API_CALL vxTensorTransposeNode (vx_graph graph, vx_tensor input, vx_tensor output, vx_\text{size} dimension1, vx_\text{size} dimension2)**
  
  [Graph] Performs transpose on the input tensor. The node transpose the tensor according to a specified 2 indexes in the tensor (0-based indexing)

- **vx_status VX_API_CALL vxuTensorTranspose (vx_context context, vx_tensor input, vx_tensor output, vx_\text{size} dimension1, vx_\text{size} dimension2)**
  
  [Immediate] Performs transpose on the input tensor. The tensor is transposed according to a specified 2 indexes in the tensor (0-based indexing)

3.59.2 Function Documentation

**vx_node VX_API_CALL vxTensorTransposeNode (vx_graph graph, vx_tensor input, vx_tensor output, vx_\text{size} dimension1, vx_\text{size} dimension2)**

[Graph] Performs transpose on the input tensor. The node transpose the tensor according to a specified 2 indexes in the tensor (0-based indexing)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in graph</strong></td>
<td>The handle to the graph.</td>
</tr>
<tr>
<td><strong>in input</strong></td>
<td>Input tensor data. Implementations must support input tensor data type VX_TYPE_INT16 with fixed_point_position 8, and tensor data types VX_TYPE_UINT8 and VX_TYPE_INT8, with fixed_point_position 0.</td>
</tr>
<tr>
<td><strong>out output</strong></td>
<td>Output tensor data.</td>
</tr>
<tr>
<td><strong>in dimension1</strong></td>
<td>Dimension index that is transposed with dim 2.</td>
</tr>
<tr>
<td><strong>in dimension2</strong></td>
<td>Dimension index that is transposed with dim 1.</td>
</tr>
</tbody>
</table>

Returns

**vx_node**.

A node reference **vx_node**. Any possible errors preventing a successful creation should be checked using vxGetStatus.

**vx_status VX_API_CALL vxuTensorTranspose (vx_context context, vx_tensor input, vx_tensor output, vx_\text{size} dimension1, vx_\text{size} dimension2)**

[Immediate] Performs transpose on the input tensor. The tensor is transposed according to a specified 2 indexes in the tensor (0-based indexing)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in context</strong></td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td><strong>in input</strong></td>
<td>Input tensor data. Implementations must support input tensor data type VX_TYPE_INT16 with fixed_point_position 8, and tensor data types VX_TYPE_UINT8 and VX_TYPE_INT8, with fixed_point_position 0.</td>
</tr>
<tr>
<td><strong>out output</strong></td>
<td>Output tensor data.</td>
</tr>
<tr>
<td><strong>in dimension1</strong></td>
<td>Dimension index that is transposed with dim 2.</td>
</tr>
<tr>
<td><strong>in dimension2</strong></td>
<td>Dimension index that is transposed with dim 1.</td>
</tr>
</tbody>
</table>
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.60 Tensor Convert Bit-Depth

3.60.1 Detailed Description

Creates a bit-depth conversion node.

Convert tensor from a specific data type and fixed point position to another data type and fixed point position.

The equation for the conversion is as follows:

\[ \text{output} = \left( \frac{\text{input} - \text{offset}}{\text{norm}} \right) \times 2^{\text{output\_fixed\_point\_position}} \]

Where offset and norm are the input parameters in \texttt{vx\_float32}. \texttt{input\_fixed\_point\_position} and \texttt{output\_fixed\_point\_position} are the fixed point positions of the input and output respectively. In case input or output tensors are of \texttt{VX\_TYPE\_FLOAT32} fixed point position 0 is used.

Functions

- \texttt{vx\_node VX\_API\_CALL vxTensorConvertDepthNode (vx\_graph graph, vx\_tensor input, vx\_enum policy, vx\_scalar norm, vx\_scalar offset, vx\_tensor output)}

  [Graph] Creates a bit-depth conversion node.

- \texttt{vx\_status VX\_API\_CALL vxuTensorConvertDepth (vx\_context context, vx\_tensor input, vx\_enum policy, vx\_scalar norm, vx\_scalar offset, vx\_tensor output)}

  [Immediate] Performs a bit-depth conversion.

3.60.2 Function Documentation

\texttt{vx\_node VX\_API\_CALL vxTensorConvertDepthNode (vx\_graph graph, vx\_tensor input, vx\_enum policy, vx\_scalar norm, vx\_scalar offset, vx\_tensor output)}

[Graph] Creates a bit-depth conversion node.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{graph}</td>
<td>The reference to the graph.</td>
<td></td>
</tr>
<tr>
<td>\texttt{input}</td>
<td>The input tensor. Implementations must support input tensor data type \texttt{VX_TYPE_INT16} with fixed_point_position 8, and tensor data types \texttt{VX_TYPE_UINT8} and \texttt{VX_TYPE_INT8}, with fixed_point_position 0.</td>
<td></td>
</tr>
<tr>
<td>\texttt{policy}</td>
<td>A \texttt{VX_TYPE_ENUM} of the \texttt{vx_convert_policy_e} enumeration.</td>
<td></td>
</tr>
<tr>
<td>\texttt{norm}</td>
<td>A scalar containing a \texttt{VX_TYPE_FLOAT32} of the normalization value.</td>
<td></td>
</tr>
<tr>
<td>\texttt{offset}</td>
<td>A scalar containing a \texttt{VX_TYPE_FLOAT32} of the offset value subtracted before normalization.</td>
<td></td>
</tr>
<tr>
<td>\texttt{output}</td>
<td>The output tensor. Implementations must support input tensor data type \texttt{VX_TYPE_INT16} with fixed_point_position 8. And \texttt{VX_TYPE_UINT8} with fixed_point_position 0.</td>
<td></td>
</tr>
</tbody>
</table>

Returns

\texttt{vx\_node}.

Return values

| \texttt{vx\_node} | A node reference. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus}. |
vx_status VX_API_CALL vxuTensorConvertDepth ( vx_context context, vx_tensor input, vx_enum policy, vx_scalar norm, vx_scalar offset, vx_tensor output )

[Immediate] Performs a bit-depth conversion.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input tensor. Implementations must support input tensor data type VX_TYPE_INT16 with fixed_point_position 8, and tensor data types VX_TYPE_UINT8 and VX_TYPE_INT8, with fixed_point_position 0.</td>
</tr>
<tr>
<td>in</td>
<td>policy</td>
<td>A VX_TYPE_ENUM of the vx_convert_policy_e enumeration.</td>
</tr>
<tr>
<td>in</td>
<td>norm</td>
<td>A scalar containing a VX_TYPE_FLOAT32 of the normalization value.</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>A scalar containing a VX_TYPE_FLOAT32 of the offset value subtracted before normalization.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output tensor. Implementations must support input tensor data type VX_TYPE_INT16 with fixed_point_position 8. And VX_TYPE_UINT8 with fixed_point_position 0.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.61 Tensor Matrix Multiply

3.61.1 Detailed Description

Creates a generalized matrix multiplication node.

Performs:

\[ \text{output} = T1(\text{input}1) * T2(\text{input}2) + T3(\text{input}3) \]

Where matrix multiplication is defined as:

\[
C[i * L + j] = \text{saturate}(\text{truncate}(\text{round}(\sum_{k=1}^{M} (C[i * L + j] + ((\text{int})A[i * M + k]) * ((\text{int})B[k * L + j])))))
\]

where i,j are indexes from 1 to N,L respectively. C matrix is of size NxL. A matrix is of size NxM and B matrix is of size MxL. For signed integers, a fixed point calculation is performed with round, truncate and saturate according to the number of accumulator bits. round: rounding to nearest on the fractional part. truncate: at every multiplication result of 32bit is truncated after rounding. saturate: a saturation if performed on the accumulation and after the truncation, meaning no saturation is performed on the multiplication result.

Data Structures

- struct vx_tensor_matrix_multiply_params_t
  
  Matrix Multiply Parameters. More...

Functions

- vx_node VX_API_CALL vxTensorMatrixMultiplyNode (vx_graph graph, vx_tensor input1, vx_tensor input2, vx_tensor input3, const vx_tensor_matrix_multiply_params_t *matrix_multiply_params, vx_tensor output)
  
  [Graph] Creates a generalized matrix multiplication node.

- vx_status VX_API_CALL vxuTensorMatrixMultiply (vx_context context, vx_tensor input1, vx_tensor input2, vx_tensor input3, const vx_tensor_matrix_multiply_params_t *matrix_multiply_params, vx_tensor output)
  
  [Immediate] Performs a generalized matrix multiplication.

3.61.2 Data Structure Documentation

struct vx_tensor_matrix_multiply_params_t

Matrix Multiply Parameters.

- transpose_input1/input2/input3 : if True the matrix is transposed before the operation, otherwise the matrix is used as is.

Definition at line 1590 of file vx_types.h.

Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_bool</td>
<td>transpose_input1</td>
<td>if True the matrix is transposed before the operation, otherwise the matrix is used as is</td>
</tr>
<tr>
<td>vx_bool</td>
<td>transpose_input2</td>
<td>if True the matrix is transposed before the operation, otherwise the matrix is used as is</td>
</tr>
<tr>
<td>vx_bool</td>
<td>transpose_input3</td>
<td>if True the matrix is transposed before the operation, otherwise the matrix is used as is</td>
</tr>
</tbody>
</table>
### 3.61.3 Function Documentation

**vx_node VX_API_CALL vxTensorMatrixMultiplyNode ( vx_graph graph, vx_tensor input1, vx_tensor input2, vx_tensor input3, const vx_tensor_matrix_multiply_params_t * matrix_multiply_params, vx_tensor output )**

[Graph] Creates a generalized matrix multiplication node.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in input1</td>
<td>The first input 2D tensor of type VX_TYPE_INT16 with fixed_point_pos 8,</td>
</tr>
<tr>
<td></td>
<td>or tensor data types VX_TYPE_UINT8 or VX_TYPE_INT8, with fixed_point_pos 0.</td>
</tr>
<tr>
<td>in input2</td>
<td>The second 2D tensor. Must be in the same data type as input1.</td>
</tr>
<tr>
<td>in input3</td>
<td>The third 2D tensor. Must be in the same data type as input1. [optional].</td>
</tr>
<tr>
<td>in matrix_multiply_params</td>
<td>Matrix multiply parameters, see vx_tensor_matrix_multiply_params_t.</td>
</tr>
<tr>
<td>out output</td>
<td>The output 2D tensor. Must be in the same data type as input1. Output dimension must agree the</td>
</tr>
<tr>
<td></td>
<td>formula in the description.</td>
</tr>
</tbody>
</table>

**Returns**

*vx_node.*

A node reference *vx_node*. Any possible errors preventing a successful creation should be checked using vxGetStatus.

**vx_status VX_API_CALL vxuTensorMatrixMultiply ( vx_context context, vx_tensor input1, vx_tensor input2, vx_tensor input3, const vx_tensor_matrix_multiply_params_t * matrix_multiply_params, vx_tensor output )**

[Immediate] Performs a generalized matrix multiplication.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>in input1</td>
<td>The first input 2D tensor of type VX_TYPE_INT16 with fixed_point_pos 8,</td>
</tr>
<tr>
<td></td>
<td>or tensor data types VX_TYPE_UINT8 or VX_TYPE_INT8, with fixed_point_pos 0.</td>
</tr>
<tr>
<td>in input2</td>
<td>The second 2D tensor. Must be in the same data type as input1.</td>
</tr>
<tr>
<td>in input3</td>
<td>The third 2D tensor. Must be in the same data type as input1. [optional].</td>
</tr>
<tr>
<td>in matrix_multiply_params</td>
<td>Matrix multiply parameters, see vx_tensor_matrix_multiply_params_t.</td>
</tr>
<tr>
<td>out output</td>
<td>The output 2D tensor. Must be in the same data type as input1. Output dimension must agree the</td>
</tr>
<tr>
<td></td>
<td>formula in the description.</td>
</tr>
</tbody>
</table>

**Returns**

A *vx_status_e* enumeration.

**Return values**

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See vx_status_e.</td>
</tr>
</tbody>
</table>
3.62 Basic Features

3.62.1 Detailed Description

The basic parts of OpenVX needed for computation.

Types in OpenVX intended to be derived from the C99 Section 7.18 standard definition of fixed width types.

Modules

- **Objects**
  
  Defines the basic objects within OpenVX.

Data Structures

- **struct vx_coordinates2d_t**
  
  The 2D Coordinates structure. More...

- **struct vx_coordinates2df_t**
  
  The floating-point 2D Coordinates structure. More...

- **struct vx_coordinates3d_t**
  
  The 3D Coordinates structure. More...

- **struct vx_keypoint_t**
  
  The keypoint data structure. More...

- **struct vx_line2d_t**
  
  line struct More...

- **struct vx_rectangle_t**
  
  The rectangle data structure that is shared with the users. The area of the rectangle can be computed as (end_x−start_x)∗(end_y−start_y). More...

Macros

- **#define VX_API_CALL**
  
  Defines calling convention for OpenVX API.

- **#define VX_ATTRIBUTE_BASE(vendor, object) (((vendor) << 20) | (object << 8))**
  
  Defines the manner in which to combine the Vendor and Object IDs to get the base value of the enumeration.

- **#define VX_ATTRIBUTE_ID_MASK (0x000000FF)**
  
  An object’s attribute ID is within the range of [0, 2^8 − 1] (inclusive).

- **#define VX_CALLBACK**
  
  Defines calling convention for user callbacks.

- **#define VX_DF_IMAGE(a, b, c, d) ((a) | (b << 8) | (c << 16) | (d << 24))**
  
  Converts a set of four chars into a uint32_t container of a VX_DF_IMAGE code.

- **#define VX_ENUM_BASE(vendor, id) (((vendor) << 20) | (id << 12))**
  
  Defines the manner in which to combine the Vendor and Object IDs to get the base value of the enumeration.

- **#define VX_ENUM_MASK (0x00000FFF)**
  
  A generic enumeration list can have values between [0, 2^{12} − 1] (inclusive).

- **#define VX_ENUM_TYPE(e) (((vx_uint32)e & VX_ENUM_TYPE_MASK) >> 12)**
  
  A macro to extract the enum type from an enumerated value.

- **#define VX_ENUM_TYPE_MASK (0x000FF000)**
  
  A type of enumeration. The valid range is between [0, 2^8 − 1] (inclusive).

- **#define VX_FMT_REF "%p"**
  
  Use to aid in debugging values in OpenVX.

- **#define VX_FMT_SIZE "%zu"**
  
  Use to aid in debugging values in OpenVX.

- **#define VX_KERNEL_BASE(vendor, lib) (((vendor) << 20) | (lib << 12))**
Defines the manner in which to combine the Vendor and Library IDs to get the base value of the enumeration.

- \#define VX_KERNEL_MASK (0x00000FFF)
  An individual kernel in a library has its own unique ID within $[0, 2^{12} - 1]$ (inclusive).
- \#define VX_LIBRARY(e) (((vx_uint32)e & VX_LIBRARY_MASK) >> 12)
  A macro to extract the kernel library enumeration from a enumerated kernel value.
- \#define VX_LIBRARY_MASK (0x000FF000)
  A library is a set of vision kernels with its own ID supplied by a vendor. The vendor defines the library ID. The range is $[0, 2^{8} - 1]$ inclusive.
- \#define VX_MAX_LOG_MESSAGE_LEN (1024)
  Defines the length of a message buffer to copy from the log, including the trailing zero.
- \#define VX_SCALE_UNITY (1024u)
  Use to indicate the 1:1 ratio in Q22.10 format.
- \#define VX_TYPE(e) (((vx_uint32)e & VX_TYPE_MASK) >> 8)
  A macro to extract the type from an enumerated attribute value.
- \#define VX_TYPE_MASK (0x000FFF00)
  A type mask removes the scalar/object type from the attribute. It is 3 nibbles in size and is contained between the third and second byte.
- \#define VX_VENDOR(e) (((vx_uint32)e & VX_VENDOR_MASK) >> 20)
  A macro to extract the vendor ID from the enumerated value.
- \#define VX_VENDOR_MASK (0xFFF00000)
  Vendor IDs are 2 nibbles in size and are located in the upper byte of the 4 bytes of an enumeration.
- \#define VX_VERSION VX_VERSION_1_2
  Defines the OpenVX Version Number.
- \#define VX_VERSION_1_0 (VX_VERSION_MAJOR(1) | VX_VERSION_MINOR(0))
  Defines the predefined version number for 1.0.
- \#define VX_VERSION_1_1 (VX_VERSION_MAJOR(1) | VX_VERSION_MINOR(1))
  Defines the predefined version number for 1.1.
- \#define VX_VERSION_1_2 (VX_VERSION_MAJOR(1) | VX_VERSION_MINOR(2))
  Defines the predefined version number for 1.2.
- \#define VX_VERSION_MAJOR(x) ((x & 0xFF) << 8)
  Defines the major version number macro.
- \#define VX_VERSION_MINOR(x) ((x & 0xFF) << 0)
  Defines the minor version number macro.

**Typedefs**

- typedef vx_enum vx_bool
  A formal boolean type with known fixed size.
- typedef char vx_char
  An 8 bit ASCII character.
- typedef uint32_t vx_df_image
  Used to hold a VX_DF_IMAGE code to describe the pixel format and color space.
- typedef int32_t vx_enum
  Sets the standard enumeration type size to be a fixed quantity.
- typedef float vx_float32
  A 32-bit float value.
- typedef double vx_float64
  A 64-bit float value (aka double).
- typedef int16_t vx_int16
  A 16-bit signed value.
A 32-bit signed value.
- typedef int64_t vx_int64
  A 64-bit signed value.
- typedef int8_t vx_int8
  An 8-bit signed value.
- typedef size_t vx_size
  A wrapper of size_t to keep the naming convention uniform.
- typedef vx_enum vx_status
  A formal status type with known fixed size.
- typedef uint16_t vx_uint16
  A 16-bit unsigned value.
- typedef uint32_t vx_uint32
  A 32-bit unsigned value.
- typedef uint64_t vx_uint64
  A 64-bit unsigned value.
- typedef uint8_t vx_uint8
  An 8-bit unsigned value.

Enumerations
- enum vx_bool_e {
  vx_false_e = 0,
  vx_true_e }
  A Boolean value. This allows 0 to be FALSE, as it is in C, and any non-zero to be TRUE.
- enum vx_channel_e {
  VX_CHANNEL_0 = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_CHANNEL << 12)) + 0x0,
  VX_CHANNEL_1 = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_CHANNEL << 12)) + 0x1,
  VX_CHANNEL_2 = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_CHANNEL << 12)) + 0x2,
  VX_CHANNEL_3 = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_CHANNEL << 12)) + 0x3,
  VX_CHANNEL_R = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_CHANNEL << 12)) + 0x10,
  VX_CHANNEL_G = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_CHANNEL << 12)) + 0x11,
  VX_CHANNEL_B = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_CHANNEL << 12)) + 0x12,
  VX_CHANNEL_A = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_CHANNEL << 12)) + 0x13,
  VX_CHANNEL_Y = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_CHANNEL << 12)) + 0x14,
  VX_CHANNEL_U = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_CHANNEL << 12)) + 0x15,
  VX_CHANNEL_V = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_CHANNEL << 12)) + 0x16 }
  The channel enumerations for channel extractions.
- enum vx_convert_policy_e {
  VX_CONVERT_POLICY_WRAP = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_CONVERT_POLICY << 12)) + 0x0,
  VX_CONVERT_POLICY_SATURATE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_CONVERT_POLICY << 12)) + 0x1 }
  The Conversion Policy Enumeration.
CHAPTER 3. MODULE DOCUMENTATION

172

VX_DF_IMAGE_VIRT = (( 'V' ) | ( 'I' << 8 ) | ( 'R' << 16 ) | ( 'T' << 24 ) ),
VX_DF_IMAGE_RGB = (( 'R' ) | ( 'G' << 8 ) | ( 'B' << 16 ) | ( 'A' << 24 ) ),
VX_DF_IMAGE_RGBX = (( 'R' ) | ( 'G' << 8 ) | ( 'B' << 16 ) | ( 'A' << 24 ) ),
VX_DF_IMAGE_NV12 = (( 'N' ) | ( 'V' << 8 ) | ( '1' << 16 ) | ( '2' << 24 ) ),
VX_DF_IMAGE_NV21 = (( 'N' ) | ( 'V' << 8 ) | ( '2' << 16 ) | ( '1' << 24 ) ),
VX_DF_IMAGE_UYVY = (( 'U' ) | ( 'Y' << 8 ) | ( 'V' << 16 ) | ( 'Y' << 24 ) ),
VX_DF_IMAGE_YUYV = (( 'Y' ) | ( 'U' << 8 ) | ( 'Y' << 16 ) | ( 'V' << 24 ) ),
VX_DF_IMAGE_IYUV = (( 'I' ) | ( 'Y' << 8 ) | ( 'U' << 16 ) | ( 'V' << 24 ) ),
VX_DF_IMAGE_YUV4 = (( 'Y' ) | ( 'U' << 8 ) | ( 'V' << 16 ) | ( '4' << 24 ) ),
VX_DF_IMAGE_U8 = (( 'U' ) | ( '0' << 8 ) | ( '0' << 16 ) | ( '8' << 24 ) ),
VX_DF_IMAGE_U16 = (( 'U' ) | ( '0' << 8 ) | ( '1' << 16 ) | ( '6' << 24 ) ),
VX_DF_IMAGE_S16 = (( 'S' ) | ( '0' << 8 ) | ( '1' << 16 ) | ( '6' << 24 ) ),
VX_DF_IMAGE_U32 = (( 'U' ) | ( '0' << 8 ) | ( '3' << 16 ) | ( '2' << 24 ) ),
VX_DF_IMAGE_S32 = (( 'S' ) | ( '0' << 8 ) | ( '3' << 16 ) | ( '2' << 24 ) )

Based on the VX_DF_IMAGE definition.

• enum vx_enum_e {
    VX_ENUM_DIRECTION = 0x00,
    VX_ENUM_ACTION = 0x01,
    VX_ENUM_HINT = 0x02,
    VX_ENUM_INTERPOLATION = 0x04,
    VX_ENUM_OVERFLOW = 0x05,
    VX_ENUM_COLOR_SPACE = 0x06,
    VX_ENUM_COLOR_RANGE = 0x07,
    VX_ENUM_PARAMETER_STATE = 0x08,
    VX_ENUM_CHANNEL = 0x09,
    VX_ENUM_CONVERT_POLICY = 0x0A,
    VX_ENUM_THRESHOLD_TYPE = 0x0B,
    VX_ENUM_BORDER = 0x0C,
    VX_ENUM_COMPARISON = 0x0D,
    VX_ENUM_MEMORY_TYPE = 0x0E,
    VX_ENUM_TERM_CRITERIA = 0x0F,
    VX_ENUM_NORM_TYPE = 0x10,
    VX_ENUM_ACCESSOR = 0x11,
    VX_ENUM_ROUND_POLICY = 0x12,
    VX_ENUM_BORDER_POLICY = 0x13,
    VX_ENUM_GRAPH_STATE = 0x15,
    VX_ENUM_NONLINEAR = 0x16,
    VX_ENUM_PATTERN = 0x17,
    VX_ENUM_LBP_FORMAT = 0x18,
    VX_ENUM_COMP_METRIC = 0x19,
    VX_ENUM_SCALAR_OPERATION = 0x20
}  

The set of supported enumerations in OpenVX.

• enum vx_interpolation_type_e {
    VX_INTERPOLATION_NEAREST_NEIGHBOR = (( ( VX_ID_KHRONOS ) << 20 ) | ( VX_ENUM_INTERPOLATION << 12 ) ) + 0x0,
    VX_INTERPOLATION_BILINEAR = (( ( VX_ID_KHRONOS ) << 20 ) | ( VX_ENUM_INTERPOLATION << 12 ) ) + 0x1,
    VX_INTERPOLATION_AREA = (( ( VX_ID_KHRONOS ) << 20 ) | ( VX_ENUM_INTERPOLATION << 12 ) ) + 0x2
}  

The image reconstruction filters supported by image resampling operations.

• enum vx_non_linear_filter_e {
    VX_NONLINEAR_FILTER_MEDIAN = (( ( VX_ID_KHRONOS ) << 20 ) | ( VX_ENUM_NONLINEAR << 12 ) ) + 0x0,
    VX_NONLINEAR_FILTER_MIN = (( ( VX_ID_KHRONOS ) << 20 ) | ( VX_ENUM_NONLINEAR << 12 ) ) + 0x1,
    VX_NONLINEAR_FILTER_MAX = (( ( VX_ID_KHRONOS ) << 20 ) | ( VX_ENUM_NONLINEAR << 12 ) ) + 0x2
}  

The set of supported non-linear filters supported by image processing operations.
0x2 }

An enumeration of non-linear filter functions.

• enum vx_pattern_e {
  VX_PATTERN_BOX = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_PATTERN << 12)) + 0x0,
  VX_PATTERN_CROSS = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_PATTERN << 12)) + 0x1,
  VX_PATTERN_DISK = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_PATTERN << 12)) + 0x2,
  VX_PATTERN_OTHER = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_PATTERN << 12)) + 0x3
}

An enumeration of matrix patterns. See vxCreateMatrixFromPattern and vxCreateMatrixFromPatternAndOrigin

• enum vx_status_e {
  VX_STATUS_MIN = -25,
  VX_ERROR_REFERENCE_NONZERO = -24,
  VX_ERROR_MULTIPLE_WRITERS = -23,
  VX_ERROR_GRAPH_ABANDONED = -22,
  VX_ERROR_GRAPH_SCHEDULED = -21,
  VX_ERROR_INVALID_SCOPE = -20,
  VX_ERROR_INVALID_NODE = -19,
  VX_ERROR_INVALID_GRAPH = -18,
  VX_ERROR_INVALID_TYPE = -17,
  VX_ERROR_INVALID_VALUE = -16,
  VX_ERROR_INVALID_DIMENSION = -15,
  VX_ERROR_INVALID_FORMAT = -14,
  VX_ERROR_INVALID_LINK = -13,
  VX_ERROR_INVALID_REFERENCE = -12,
  VX_ERROR_INVALID_MODULE = -11,
  VX_ERROR_INVALID_PARAMETERS = -10,
  VX_ERROR_OPTIMIZED_AWAY = -9,
  VX_ERROR_NO_MEMORY = -8,
  VX_ERROR_NO_RESOURCES = -7,
  VX_ERROR_NOT_COMPATIBLE = -6,
  VX_ERROR_NOT_ALLOCATED = -5,
  VX_ERROR_NOT_SUFFICIENT = -4,
  VX_ERROR_NOT_SUPPORTED = -3,
  VX_ERROR_NOT_IMPLEMENTED = -2,
  VX_FAILURE = -1,
  VX_SUCCESS = 0
}

The enumeration of all status codes.

• enum vx_target_e {
  VX_TARGET_ANY = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_TARGET << 12)) + 0x0000,
  VX_TARGET_STRING = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_TARGET << 12)) + 0x0001,
  VX_TARGET_VENDOR_BEGIN = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_TARGET << 12)) + 0x1000
}

The Target Enumeration.

• enum vx_type_e {

The type enumeration lists all the known types in OpenVX.

• enum vx_vendor_id_e {

VX_TYPE_INVALID = 0x000,
VX_TYPE_CHAR = 0x001,
VX_TYPE_INT8 = 0x002,
VX_TYPE_UINT8 = 0x003,
VX_TYPE_INT16 = 0x004,
VX_TYPE_UINT16 = 0x005,
VX_TYPE_INT32 = 0x006,
VX_TYPE_UINT32 = 0x007,
VX_TYPE_INT64 = 0x008,
VX_TYPE_UINT64 = 0x009,
VX_TYPE_FLOAT32 = 0x00A,
VX_TYPE_FLOAT64 = 0x00B,
VX_TYPE_ENUM = 0x00C,
VX_TYPE_SIZE = 0x00D,
VX_TYPE_DF_IMAGE = 0x00E,
VX_TYPE_BOOL = 0x010,
VX_TYPE_RECTANGLE = 0x020,
VX_TYPE_KEYPOINT = 0x021,
VX_TYPE_COORDINATES2D = 0x022,
VX_TYPE_COORDINATES3D = 0x023,
VX_TYPE_COORDINATES2DF = 0x024,
VX_TYPE_HOG_PARAMS = 0x028,
VX_TYPE_HOUGH_LINES_PARAMS = 0x029,
VX_TYPE_LINE_2D = 0x02A,
VX_TYPE_TENSOR_MATRIX_MULTIPLY_PARAMS = 0x02B,
VX_TYPE_USER_STRUCT_START = 0x100,
VX_TYPE_VENDOR_STRUCT_START = 0x400,
VX_TYPE_KHRONOS_OBJECT_START = 0x800,
VX_TYPE_VENDOR_OBJECT_START = 0xC00,
VX_TYPE_KHRONOS_STRUCT_MAX = VX_TYPE_USER_STRUCT_START - 1,
VX_TYPE_USER_STRUCT_END = VX_TYPE_VENDOR_STRUCT_START - 1,
VX_TYPE_VENDOR_STRUCT_END = VX_TYPE_KHRONOS_OBJECT_START - 1,
VX_TYPE_VENDOR_OBJECT_END = 0xFFF,
VX_TYPE_REFERENCE = 0x800,
VX_TYPE_CONTEXT = 0x801,
VX_TYPE_GRAPH = 0x802,
VX_TYPE_NODE = 0x803,
VX_TYPE_KERNEL = 0x804,
VX_TYPE_PARAMETER = 0x805,
VX_TYPE_DELAY = 0x806,
VX_TYPE_LUT = 0x807,
VX_TYPE_DISTRIBUTION = 0x808,
VX_TYPE_PYRAMID = 0x809,
VX_TYPE_THRESHOLD = 0x80A,
VX_TYPE_MATRIX = 0x80B,
VX_TYPE_CONVOLUTION = 0x80C,
VX_TYPE_SCALAR = 0x80D,
VX_TYPE_ARRAY = 0x80E,
VX_TYPE_IMAGE = 0x80F,
VX_TYPE_REMAP = 0x810,
VX_TYPE_ERROR = 0x811,
VX_TYPE_META_FORMAT = 0x812,
VX_TYPE_OBJECT_ARRAY = 0x813,
VX_TYPE_TENSOR = 0x815}

The type enumeration lists all the known types in OpenVX.
VX_ID_KHRONOS = 0x000,
VX_ID_TI = 0x001,
VX_ID.Qualcomm = 0x002,
VX_ID_NVIDIA = 0x003,
VX_ID_ARM = 0x004,
VX_ID_BDTI = 0x005,
VX_ID_RENESAS = 0x006,
VX_ID_VIVANTE = 0x007,
VX_ID_XILINX = 0x008,
VX_ID_AXIS = 0x009,
VX_ID_MOVIDIUS = 0x00A,
VX_ID_SAMSUNG = 0x00B,
VX_ID_FREESCALE = 0x00C,
VX_ID_AMD = 0x00D,
VX_ID_BROADCOM = 0x00E,
VX_ID_INTEL = 0x00F,
VX_ID_MARVELL = 0x010,
VX_ID_MEDIATEK = 0x011,
VX_ID_ST = 0x012,
VX_ID_CEVA = 0x013,
VX_ID_ITSEEZ = 0x014,
VX_ID_IMAGINATION = 0x015,
VX_ID_NXP = 0x016,
VX_ID_VIDEANTIS = 0x017,
VX_ID_SYNOPSYS = 0x018,
VX_ID_CADENCE = 0x019,
VX_ID_HUAWEI = 0x01A,
VX_ID_SOCIONEXT = 0x01B,
VX_ID_USER = 0xFFE,
VX_ID_MAX = 0xFFF,
VX_ID_DEFAULT = VX_ID_MAX

The Vendor ID of the Implementation. As new vendors submit their implementations, this enumeration will grow.

Functions

- **vx_status VX_API_CALL vxGetStatus (vx_reference reference)**

  Provides a generic API to return status values from Object constructors if they fail.

3.62.2 Data Structure Documentation

**struct vx_coordinates2d_t**

The 2D Coordinates structure.

Definition at line 1658 of file vx_types.h.

**Data Fields**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_uint32</td>
<td>x: The X coordinate.</td>
</tr>
<tr>
<td>vx_uint32</td>
<td>y: The Y coordinate.</td>
</tr>
</tbody>
</table>

**struct vx_coordinates2df_t**

The floating-point 2D Coordinates structure.

Definition at line 1666 of file vx_types.h.

**Data Fields**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_float32</td>
<td>x: The X coordinate.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

Data Fields

| vx_float32 | y | The Y coordinate. |

struct vx_coordinates3d_t

The 3D Coordinates structure.

Definition at line 1674 of file vx_types.h.

Data Fields

| vx_uint32 | x | The X coordinate. |
| vx_uint32 | y | The Y coordinate. |
| vx_uint32 | z | The Z coordinate. |

struct vx_keypoint_t

The keypoint data structure.

Definition at line 1635 of file vx_types.h.

Data Fields

| vx_int32  | x | The x coordinate. |
| vx_int32  | y | The y coordinate. |
| vx_float32| strength | The strength of the keypoint. Its definition is specific to the corner detector. |
| vx_float32| scale | Initialized to 0 by corner detectors. |
| vx_float32| orientation | Initialized to 0 by corner detectors. |
| vx_int32  | tracking_status | A zero indicates a lost point. Initialized to 1 by corner detectors. |
| vx_float32| error | A tracking method specific error. Initialized to 0 by corner detectors. |

struct vx_line2d_t

line struct

Definition at line 1573 of file vx_types.h.

Data Fields

| vx_float32 | start_x | x index of line start |
| vx_float32 | start_y | y index of line start |
| vx_float32 | end_x | x index of line end |
| vx_float32 | end_y | y index of line end |

struct vx_rectangle_t

The rectangle data structure that is shared with the users. The area of the rectangle can be computed as (end_x-start_x)∗(end_y-start_y).

Definition at line 1648 of file vx_types.h.
Data Fields

<table>
<thead>
<tr>
<th>vx_uint32</th>
<th>start←_x</th>
<th>The Start X coordinate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_uint32</td>
<td>start←_y</td>
<td>The Start Y coordinate.</td>
</tr>
<tr>
<td>vx_uint32</td>
<td>end←_x</td>
<td>The End X coordinate.</td>
</tr>
<tr>
<td>vx_uint32</td>
<td>end←_y</td>
<td>The End Y coordinate.</td>
</tr>
</tbody>
</table>

3.62.3 Macro Definition Documentation

#define VX_TYPE_MASK (0x000FFF00)

A type mask removes the scalar/object type from the attribute. It is 3 nibbles in size and is contained between the third and second byte.

See also vx_type_e

Definition at line 473 of file vx_types.h.

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

Converts a set of four chars into a uint32_t container of a VX_DF_IMAGE code.

Note

Use a vx_df_image variable to hold the value.

Definition at line 526 of file vx_types.h.

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

Defines the manner in which to combine the Vendor and Object IDs to get the base value of the enumeration.

From any enumerated value (with exceptions), the vendor, and enumeration type should be extractable. Those types that are exceptions are vx_vendor_id_e, vx_type_e, vx_enum_e, vx_df_image_e, and vx←bool.

Definition at line 550 of file vx_types.h.

3.62.4 Typedef Documentation

typedef int32_t vx_enum

Sets the standard enumeration type size to be a fixed quantity.

All enumerable fields must use this type as the container to enforce enumeration ranges and sizeof() operations.

Definition at line 149 of file vx_types.h.

typedef vx_enum vx_bool

A formal boolean type with known fixed size.

See also vx_bool_e

Definition at line 309 of file vx_types.h.
typedef vx_enum vx_status
A formal status type with known fixed size.

See also

vx_status_e

Definition at line 445 of file vx_types.h.

3.62.5 Enumeration Type Documentation

enum vx_bool_e
A Boolean value. This allows 0 to be FALSE, as it is in C, and any non-zero to be TRUE.

```
00001 vx_bool ret = vx_true_e;
00002 if (ret) printf("true!\n");
00003 ret = vx_false_e;
00004 if (!ret) printf("false!\n");
```

This would print both strings.

See also

vx_bool

Enumerator

vx_false_e  The "false" value.
vx_true_e  The "true" value.

Definition at line 298 of file vx_types.h.

enum vx_type_e
The type enumeration lists all the known types in OpenVX.

Enumerator

VX_TYPE_INVALID  An invalid type value. When passed an error must be returned.
VX_TYPE_CHAR  A vx_char.
VX_TYPE_INT8  A vx_int8.
VX_TYPE_UINT8  A vx_uint8.
VX_TYPE_INT16  A vx_int16.
VX_TYPE_UINT16  A vx_uint16.
VX_TYPE_INT32  A vx_int32.
VX_TYPE_UINT32  A vx_uint32.
VX_TYPE_INT64  A vx_int64.
VX_TYPE_UINT64  A vx_uint64.
VX_TYPE_FLOAT32  A vx_float32.
VX_TYPE_FLOAT64  A vx_float64.
VX_TYPE_ENUM  A vx_enum. Equivalent in size to a vx_int32.
VX_TYPE_SIZE  A vx_size.
VX_TYPE_DF_IMAGE  A vx_df_image.
VX_TYPE_BOOL  A vx_bool.
VX_TYPE_RECTANGLE  A vx_rectangle_t.
VX_TYPE_KEYPOINT  A vx_keypoint_t.
VX_TYPE_COORDINATES2D  A vx_coordinates2d_t.
CHAPTER 3. MODULE DOCUMENTATION

VX_TYPE_COORDINATES3D  A `vx_coordinates3d_t`.
VX_TYPE_COORDINATES2DF  A `vx_coordinates2df_t`.
VX_TYPE_HOG_PARAMS  A `vx_hog_t`.
VX_TYPE_HOUGH_LINES_PARAMS  A `vx_hough_lines_p_t`.
VX_TYPE_LINE_2D  A `vx_line2d_t`.
VX_TYPE_TENSOR_MATRIX_MULTIPLY_PARAMS  A `vx_tensor_matrix_multiply_params_t`.
VX_TYPE_USER_STRUCT_START  A user-defined struct base index.
VX_TYPE_VENDOR_STRUCT_START  A vendor-defined struct base index.
VX_TYPE_KHRONOS_OBJECT_START  A Khronos defined object base index.
VX_TYPE_VENDOR_OBJECT_START  A vendor defined object base index.
VX_TYPE_KHRONOS_OBJECT_END  A value for comparison between Khronos defined structs and user structs.
VX_TYPE_USER_STRUCT_END  A value for comparison between user structs and vendor structs.
VX_TYPE_VENDOR_STRUCT_END  A value for comparison between vendor structs and Khronos defined objects.
VX_TYPE_KHRONOS_OBJECT_END  A value for comparison between Khronos defined objects and vendor structs.
VX_TYPE_VENDOR_OBJECT_END  A value used for bound checking of vendor objects.
VX_TYPE_REFERENCE  A `vx_reference`.
VX_TYPE_CONTEXT  A `vx_context`.
VX_TYPE_GRAPH  A `vx_graph`.
VX_TYPE_NODE  A `vx_node`.
VX_TYPE_KERNEL  A `vx_kernel`.
VX_TYPE_PARAMETER  A `vx_parameter`.
VX_TYPE_DELAY  A `vx_delay`.
VX_TYPE_LUT  A `vx_lut`.
VX_TYPE_DISTRIBUTION  A `vx_distribution`.
VX_TYPE_PYRAMID  A `vx_pyramid`.
VX_TYPE_THRESHOLD  A `vx_threshold`.
VX_TYPE_MATRIX  A `vx_matrix`.
VX_TYPE_CONVOLUTION  A `vx_convolution`.
VX_TYPE_SCALAR  A `vx_scalar` when needed to be completely generic for kernel validation.
VX_TYPE_ARRAY  A `vx_array`.
VX_TYPE_IMAGE  A `vx_image`.
VX_TYPE_REMAP  A `vx_remap`.
VX_TYPE_ERROR  An error object which has no type.
VX_TYPE_META_FORMAT  A `vx_meta_format`.
VX_TYPE_OBJECT_ARRAY  A `vx_object_array`.
VX_TYPE_TENSOR  A `vx_tensor`.

Definition at line 325 of file `vx_types.h`. 
enum vx_status_e

The enumeration of all status codes.

See also

vx_status.

Enumerator

VX_STATUS_MIN  Indicates the lower bound of status codes in VX. Used for bounds checks only.

VX_ERROR_REFERENCE_NONZERO  Indicates that an operation did not complete due to a reference count being non-zero.

VX_ERROR_MULTIPLE_WRITERS  Indicates that the graph has more than one node outputting to the same data object. This is an invalid graph structure.

VX_ERROR_GRAPH_ABANDONED  Indicates that the graph is stopped due to an error or a callback that abandoned execution.

VX_ERROR_GRAPH_SCHEDULED  Indicates that the supplied graph already has been scheduled and may be currently executing.

VX_ERROR_INVALID_SCOPE  Indicates that the supplied parameter is from another scope and cannot be used in the current scope.

VX_ERROR_INVALID_MODULE  Indicates that the supplied node could not be created.

VX_ERROR_INVALID_GRAPH  Indicates that the supplied graph has invalid connections (cycles).

VX_ERROR_INVALID_TYPE  Indicates that the supplied type parameter is incorrect.

VX_ERROR_INVALID_VALUE  Indicates that the supplied parameter has an incorrect value.

VX_ERROR_INVALID_DIMENSION  Indicates that the supplied parameter is too big or too small in dimension.

VX_ERROR_INVALID_FORMAT  Indicates that the supplied parameter is in an invalid format.

VX_ERROR_INVALID_LINK  Indicates that the link is not possible as specified. The parameters are incompatible.

VX_ERROR_INVALID_REFERENCE  Indicates that the reference provided is not valid.

VX_ERROR_INVALID_MODULE  This is returned from vxLoadKernels when the module does not contain the entry point.

VX_ERROR_INVALID_PARAMETERS  Indicates that the supplied parameter information does not match the kernel contract.

VX_ERROR_OPTIMIZED_AWAY  Indicates that the object refered to has been optimized out of existence.

VX_ERROR_NO_MEMORY  Indicates that an internal or implicit allocation failed. Typically catastrophic. After detection, deconstruct the context.

See also

vxVerifyGraph.

VX_ERROR_NO_RESOURCES  Indicates that an internal or implicit resource can not be acquired (not memory). This is typically catastrophic. After detection, deconstruct the context.

See also

vxVerifyGraph.

VX_ERROR_NOT_COMPATIBLE  Indicates that the attempt to link two parameters together failed due to type incompatibility.

VX_ERROR_NOT_ALLOCATED  Indicates to the system that the parameter must be allocated by the system.

VX_ERROR_NOT_SUFFICIENT  Indicates that the given graph has failed verification due to an insufficient number of required parameters, which cannot be automatically created. Typically this indicates required atomic parameters.
See also vxVerifyGraph.

**VX_ERROR_NOT_SUPPORTED** Indicates that the requested set of parameters produce a configuration that cannot be supported. Refer to the supplied documentation on the configured kernels.

See also vx_kernel_e. This is also returned if a function to set an attribute is called on a Read-only attribute.

**VX_ERROR_NOT_IMPLEMENTED** Indicates that the requested kernel is missing.

See also vx_kernel_e vxGetKernelByName.

**VX_FAILURE** Indicates a generic error code, used when no other describes the error.

**VX_SUCCESS** No error.

Definition at line 411 of file vx_types.h.

```
enum vx_enum_e
```

The set of supported enumerations in OpenVX.

These can be extracted from enumerated values using **VX_ENUM_TYPE**.

**Enumerator**

- **VX_ENUM_DIRECTION** Parameter Direction.
- **VX_ENUM_ACTION** Action Codes.
- **VX_ENUM_HINT** Hint Values.
- **VX_ENUM_DIRECTIVE** Directive Values.
- **VX_ENUM_INTERPOLATION** Interpolation Types.
- **VX_ENUM_OVERFLOW** Overflow Policies.
- **VX_ENUM_COLOR_SPACE** Color Space.
- **VX_ENUM_COLOR_RANGE** Color Space Range.
- **VX_ENUM_PARAMETER_STATE** Parameter State.
- **VX_ENUM_CHANNEL** Channel Name.
- **VX_ENUM_CONVERT_POLICY** Convert Policy.
- **VX_ENUM_THRESHOLD_TYPE** Threshold Type List.
- **VX_ENUM_BORDER** Border Mode List.
- **VX_ENUM_COMPARISON** Comparison Values.
- **VX_ENUM_MEMORY_TYPE** The memory type enumeration.
- **VX_ENUM_TERM_CRITERIA** A termination criteria.
- **VX_ENUM_NORM_TYPE** A norm type.
- **VX_ENUM_ACCESSOR** An accessor flag type.
- **VX_ENUM_ROUND_POLICY** Rounding Policy.
- **VX_ENUM_TARGET** Target.
- **VX_ENUM_BORDER_POLICY** Unsupported Border Mode Policy List.
- **VX_ENUM_GRAPH_STATE** Graph attribute states.
- **VX_ENUM_NONLINEAR** Non-linear function list.
- **VX_ENUM_PATTERN** Matrix pattern enumeration.
- **VX_ENUM_LBP_FORMAT** Lbp format.
- **VX_ENUM_COMP_METRIC** Compare metric.
- **VX_ENUM_SCALAR_OPERATION** Scalar operation list.

Definition at line 556 of file vx_types.h.
enum vx_convert_policy_e

The Conversion Policy Enumeration.

Enumerator

- **VX_CONVERT_POLICY_WRAP** Results are the least significant bits of the output operand, as if stored in two's complement binary format in the size of its bit-depth.

- **VX_CONVERT_POLICY_SATURATE** Results are saturated to the bit depth of the output operand.

Definition at line 694 of file vx_types.h.

enum vx_df_image_e

Based on the VX_DF_IMAGE definition.

Note

Use `vx_df_image` to contain these values.

Enumerator

- **VX_DF_IMAGE_VIRT** A virtual image of no defined type.

- **VX_DF_IMAGE_RGB** A single plane of 24-bit pixel as 3 interleaved 8-bit units of R then G then B data. This uses the BT709 full range by default.

- **VX_DF_IMAGE_RGBX** A single plane of 32-bit pixel as 4 interleaved 8-bit units of R then G then B data, then a don't care byte. This uses the BT709 full range by default.

- **VX_DF_IMAGE_NV12** A 2-plane YUV format of Luma (Y) and interleaved UV data at 4:2:0 sampling. This uses the BT709 full range by default.

- **VX_DF_IMAGE_NV21** A 2-plane YUV format of Luma (Y) and interleaved VU data at 4:2:0 sampling. This uses the BT709 full range by default.

- **VX_DF_IMAGE_UYVY** A single plane of 32-bit macro pixel of U0, Y0, V0, Y1 bytes. This uses the BT709 full range by default.

- **VX_DF_IMAGE_YUV4** A 3 plane of 8-bit 4:4:4 sampled Y, U, V planes. This uses the BT709 full range by default.

- **VX_DF_IMAGE_YUV4** A 3 plane of 8 bit 4:4:4 sampled Y, U, V planes. This uses the BT709 full range by default.

- **VX_DF_IMAGE_U8** A single plane of unsigned 8-bit data. The range of data is not specified, as it may be extracted from a YUV or generated.

- **VX_DF_IMAGE_U16** A single plane of unsigned 16-bit data. The range of data is not specified, as it may be extracted from a YUV or generated.

- **VX_DF_IMAGE_S16** A single plane of signed 16-bit data. The range of data is not specified, as it may be extracted from a YUV or generated.

- **VX_DF_IMAGE_U32** A single plane of unsigned 32-bit data. The range of data is not specified, as it may be extracted from a YUV or generated.

- **VX_DF_IMAGE_S32** A single plane of unsigned 32-bit data. The range of data is not specified, as it may be extracted from a YUV or generated.

Definition at line 707 of file vx_types.h.

enum vx_target_e

The Target Enumeration.

Enumerator

- **VX_TARGET_ANY** Any available target. An OpenVX implementation must support at least one target associated with this value.
CHAPTER 3. MODULE DOCUMENTATION

**VX_TARGET_STRING** Target, explicitly specified by its (case-insensitive) name string.

**VX_TARGET_VENDOR_BEGIN** Start of Vendor specific target enumerates.

Definition at line 773 of file `vx_types.h`.

```c
enum vx_channel_e
```

The channel enumerations for channel extractions.

See also

- `vxChannelExtractNode`
- `vxuChannelExtract`
- `VX_KERNEL_CHANNEL_EXTRACT`

**Enumerators**

- **VX_CHANNEL_0** Used by formats with unknown channel types.
- **VX_CHANNEL_1** Used by formats with unknown channel types.
- **VX_CHANNEL_2** Used by formats with unknown channel types.
- **VX_CHANNEL_3** Used by formats with unknown channel types.
- **VX_CHANNEL_R** Use to extract the RED channel, no matter the byte or packing order.
- **VX_CHANNEL_G** Use to extract the GREEN channel, no matter the byte or packing order.
- **VX_CHANNEL_B** Use to extract the BLUE channel, no matter the byte or packing order.
- **VX_CHANNEL_A** Use to extract the ALPHA channel, no matter the byte or packing order.
- **VX_CHANNEL_Y** Use to extract the LUMA channel, no matter the byte or packing order.
- **VX_CHANNEL_U** Use to extract the Cr/U/Value channel, no matter the byte or packing order.
- **VX_CHANNEL_V** Use to extract the Cr/V/Value channel, no matter the byte or packing order.

Definition at line 1194 of file `vx_types.h`.

```c
enum vx_interpolation_type_e
```

The image reconstruction filters supported by image resampling operations.

The edge of a pixel is interpreted as being aligned to the edge of the image. The value for an output pixel is evaluated at the center of that pixel.

This means, for example, that an even enlargement of a factor of two in nearest-neighbor interpolation will replicate every source pixel into a 2x2 quad in the destination, and that an even shrink by a factor of two in bilinear interpolation will create each destination pixel by average a 2x2 quad of source pixels.

Samples that cross the boundary of the source image have values determined by the border mode - see `vx←_border_e` and `VX_NODE_BORDER`.

See also

- `vxuScaleImage`
- `vxScaleImageNode`
- `VX_KERNEL_SCALE_IMAGE`
- `vxuWarpAffine`
- `vxWarpAffineNode`
- `VX_KERNEL_WARP_AFFINE`
- `vxuWarpPerspective`
- `vxWarpPerspectiveNode`
- `VX_KERNEL_WARP_PERSPECTIVE`

**Enumerators**

- **VX_INTERPOLATION_NEAREST_NEIGHBOR** Output values are defined to match the source pixel whose center is nearest to the sample position.
**VX_INTERPOLATION_BILINEAR**  Output values are defined by bilinear interpolation between the pixels whose centers are closest to the sample position, weighted linearly by the distance of the sample from the pixel centers.

**VX_INTERPOLATION_AREA**  Output values are determined by averaging the source pixels whose areas fall under the area of the destination pixel, projected onto the source image.

Definition at line 1254 of file `vx_types.h`.

**enum vx_non_linear_filter_e**

An enumeration of non-linear filter functions.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_NONLINEAR_FILTER_MEDIAN</td>
<td>Nonlinear median filter.</td>
</tr>
<tr>
<td>VX_NONLINEAR_FILTER_MIN</td>
<td>Nonlinear Erode.</td>
</tr>
<tr>
<td>VX_NONLINEAR_FILTER_MAX</td>
<td>Nonlinear Dilate.</td>
</tr>
</tbody>
</table>

Definition at line 1268 of file `vx_types.h`.

**enum vx_pattern_e**

An enumeration of matrix patterns. See `vxCreateMatrixFromPattern` and `vxCreateMatrixFromPatternAndOrigin`.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_PATTERN_BOX</td>
<td>Box pattern matrix.</td>
</tr>
<tr>
<td>VX_PATTERN_CROSS</td>
<td>Cross pattern matrix.</td>
</tr>
<tr>
<td>VX_PATTERN_DISK</td>
<td>A square matrix (rows = columns = size)</td>
</tr>
<tr>
<td>VX_PATTERN_OTHER</td>
<td>Matrix with any pattern other than above.</td>
</tr>
</tbody>
</table>

Definition at line 1281 of file `vx_types.h`.

**enum vx_vendor_id_e**

The Vendor ID of the Implementation. As new vendors submit their implementations, this enumeration will grow.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ID_KHRONOS</td>
<td>The Khronos Group.</td>
</tr>
<tr>
<td>VX_ID_TI</td>
<td>Texas Instruments, Inc.</td>
</tr>
<tr>
<td>VX_ID_QUALCOMM</td>
<td>Qualcomm, Inc.</td>
</tr>
<tr>
<td>VX_ID_NVIDIA</td>
<td>NVIDIA Corporation.</td>
</tr>
<tr>
<td>VX_ID_ARM</td>
<td>ARM Ltd.</td>
</tr>
<tr>
<td>VX_ID_BDTI</td>
<td>Berkley Design Technology, Inc.</td>
</tr>
<tr>
<td>VX_ID_RENESAS</td>
<td>Renasas Electronics.</td>
</tr>
<tr>
<td>VX_ID_VIVANTE</td>
<td>Vivante Corporation.</td>
</tr>
<tr>
<td>VX_ID_XILINX</td>
<td>Xilinx Inc.</td>
</tr>
<tr>
<td>VX_ID_AXIS</td>
<td>Axis Communications.</td>
</tr>
<tr>
<td>VX_ID_MOVIDIUS</td>
<td>Movidius Ltd.</td>
</tr>
<tr>
<td>VX_ID_SAMSUNG</td>
<td>Samsung Electronics.</td>
</tr>
<tr>
<td>VX_ID_FREESCALE</td>
<td>Freescale Semiconductor.</td>
</tr>
<tr>
<td>VX_ID_AMD</td>
<td>Advanced Micro Devices.</td>
</tr>
<tr>
<td>VX_ID_BROADCOM</td>
<td>Broadcom Corporation.</td>
</tr>
<tr>
<td>VX_ID_INTEL</td>
<td>Intel Corporation.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

**VX_ID_MARVELL**  Marvell Technology Group Ltd.

**VX_ID_MEDIATEK**  MediaTek, Inc.

**VX_ID_ST**  STMicroelectronics.

**VX_ID_CEVA**  CEVA DSP.

**VX_ID_ITSEEZ**  Itseez, Inc.

**VX_ID_IMAGINATION**  Imagination Technologies.

**VX_ID_NXP**  NXP Semiconductors.

**VX_ID_VIDEANTIS**  Videantis.

**VX_ID_SYNOPSYS**  Synopsys.

**VX_ID_CADENCE**  Cadence.

**VX_ID_HUAWEI**  Huawei.

**VX_ID_SOCIONEXT**  Socionext.

**VX_ID_USER**  For use by vxAllocateUserKernelId and vxAllocateUserKernelLibraryId.

**VX_ID_DEFAULT**  For use by all Kernel authors until they can obtain an assigned ID.

Definition at line 30 of file vx_vendors.h.

### 3.62.6 Function Documentation

**vx_status VX_API_CALL vxGetStatus ( vx_reference reference )**

Provides a generic API to return status values from Object constructors if they fail.

**Note**

Users do not need to strictly check every object creator as the errors should properly propagate and be detected during verification time or run-time.

```
00001 vx_image img = vxCreateImage(context, 639, 480, VX_DF_IMAGE_UYVY);
00002 vx_status status = vxGetStatus((vx_reference)img);
00003 // status == VX_ERROR_INVALID_DIMENSIONS
00004 vxReleaseImage(img);
```

**Precondition**

Appropriate Object Creator function.

**Postcondition**

Appropriate Object Release function.

**Parameters**

| in | reference | The reference to check for construction errors. |

**Returns**

A **vx_status_e** enumeration.

**Return values**

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Some error occurred, please check enumeration list and constructor.</td>
</tr>
</tbody>
</table>
3.63 Objects

3.63.1 Detailed Description

Defines the basic objects within OpenVX.

All objects in OpenVX derive from a \texttt{vx_reference} and contain a reference to the \texttt{vx_context} from which they were made, except the \texttt{vx_context} itself.

Modules

\begin{itemize}
\item **Object: Reference**
  Defines the Reference Object interface.
\item **Object: Context**
  Defines the Context Object Interface.
\item **Object: Graph**
  Defines the Graph Object interface.
\item **Object: Node**
  Defines the Node Object interface.
\item **Object: Array**
  Defines the Array Object Interface.
\item **Object: Convolution**
  Defines the Image Convolution Object interface.
\item **Object: Distribution**
  Defines the Distribution Object Interface.
\item **Object: Image**
  Defines the Image Object interface.
\item **Object: LUT**
  Defines the Look-Up Table Interface.
\item **Object: Matrix**
  Defines the Matrix Object Interface.
\item **Object: Pyramid**
  Defines the Image Pyramid Object Interface.
\item **Object: Remap**
  Defines the Remap Object Interface.
\item **Object: Scalar**
  Defines the Scalar Object interface.
\item **Object: Threshold**
  Defines the Threshold Object Interface.
\item **Object: ObjectArray**
  An opaque array object that could be an array of any data-object (not data-type) of OpenVX except Delay and ObjectArray Array objects.
\item **Object: Tensor**
  Defines The Tensor Object Interface.
\end{itemize}
3.64 Object: Reference

3.64.1 Detailed Description

Defines the Reference Object interface.

All objects in OpenVX are derived (in the object-oriented sense) from `vx_reference`. All objects shall be able to be cast back to this type safely.

### Macros

- `#define VX_MAX_REFERENCE_NAME (64)`
  
  Defines the length of the reference name string, including the trailing zero.

### Typedefs

- `typedef struct _vx_reference ∗ vx_reference`
  
  A generic opaque reference to any object within OpenVX.

### Enumerations

- `enum vx_reference_attribute_e {
  VX_REFERENCE_COUNT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_REFERENCE << 8)) + 0x0,
  VX_REFERENCE_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_REFERENCE << 8)) + 0x1,
  VX_REFERENCE_NAME = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_REFERENCE << 8)) + 0x2 }
  
  The reference attributes list.

### Functions

- `vx_status VX_API_CALL vxQueryReference (vx_reference ref, vx_enum attribute, void ∗ptr, vx_size size)`
  
  Queries any reference type for some basic information like count or type.

- `vx_status VX_API_CALL vxReleaseReference (vx_reference ∗ref_ptr)`
  
  Releases a reference. The reference may potentially refer to multiple OpenVX objects of different types. This function can be used instead of calling a specific release function for each individual object type (e.g. `vxRelease<object>`). The object will not be destroyed until its total reference count is zero.

- `vx_status VX_API_CALL vxRetainReference (vx_reference ref)`
  
  Increments the reference counter of an object This function is used to express the fact that the OpenVX object is referenced multiple times by an application. Each time this function is called for an object, the application will need to release the object one additional time before it can be destructed.

- `vx_status VX_API_CALL vxSetReferenceName (vx_reference ref, const vx_char ∗name)`
  
  Name a reference
  
  This function is used to associate a name to a referenced object. This name can be used by the OpenVX implementation in log messages and any other reporting mechanisms.

### 3.64.2 Macro Definition Documentation

`#define VX_MAX_REFERENCE_NAME (64)`

Defines the length of the reference name string, including the trailing zero.

See also

- `vxSetReferenceName`

  Definition at line 45 of file `vx.h`. 
3.64.3 Typedef Documentation

typedef struct _vx_reference *vx_reference

A generic opaque reference to any object within OpenVX.

A user of OpenVX should not assume that this can be cast directly to anything; however, any object in OpenVX
can be cast back to this for the purposes of querying attributes of the object or for passing the object as a parameter
to functions that take a vx_reference type. If the API does not take that specific type but may take others, an
error may be returned from the API.

Definition at line 142 of file vx_types.h.

3.64.4 Enumeration Type Documentation

enum vx_reference_attribute_e

The reference attributes list.

Enumerator

- **VX_REFERENCE_COUNT** Returns the reference count of the object. Read-only. Use a vx_uint32 parameter.
- **VX_REFERENCE_TYPE** Returns the vx_type_e of the reference. Read-only. Use a vx_enum parameter.
- **VX_REFERENCE_NAME** Used to query the reference for its name. Read-write. Use a *vx_char parameter.

Definition at line 785 of file vx_types.h.

3.64.5 Function Documentation

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

Queries any reference type for some basic information like count or type.

Parameters

| In | ref  | The reference to query. |
| In | attribute | The value for which to query. Use vx_reference_attribute_e. |
| Out | ptr | The location at which to store the resulting value. |
| In | size | The size in bytes of the container to which ptr points. |

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>ref is not a valid vx_reference reference.</td>
</tr>
</tbody>
</table>

vx_status VX_API_CALL vxReleaseReference ( vx_reference *ref_ptr )

Releases a reference. The reference may potentially refer to multiple OpenVX objects of different types. This
function can be used instead of calling a specific release function for each individual object type (e.g. vx←
Release<object>). The object will not be destroyed until its total reference count is zero.
Note

After returning from this function the reference is zeroed.

Parameters

| in | ref_ptr | The pointer to the reference of the object to release. |

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>ref_ptr is not a valid <code>vx_reference</code> reference.</td>
</tr>
</tbody>
</table>

`vx_status VX_API_CALL vxRetainReference (vx_reference ref)`

Increments the reference counter of an object. This function is used to express the fact that the OpenVX object is referenced multiple times by an application. Each time this function is called for an object, the application will need to release the object one additional time before it can be destructed.

Parameters

| in | ref | The reference to retain. |

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>ref is not a valid <code>vx_reference</code> reference.</td>
</tr>
</tbody>
</table>

`vx_status VX_API_CALL vxSetReferenceName (vx_reference ref, const vx_char *name)`

Name a reference

This function is used to associate a name to a referenced object. This name can be used by the OpenVX implementation in log messages and any other reporting mechanisms.

The OpenVX implementation will not check if the name is unique in the reference scope (context or graph). Several references can then have the same name.

Parameters

| in | ref | The reference to the object to be named. |

| in | name | Pointer to the ’0’ terminated string that identifies the referenced object. The string is copied by the function so that it stays the property of the caller. NULL means that the reference is not named. The length of the string shall be lower than VX_MAX_REFERENCE_NAME bytes. |
Returns

A \texttt{vx\_status\_e} enumeration.

Return values

<table>
<thead>
<tr>
<th>\texttt{VX_SUCCESS}</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{VX_ERROR_INVALID_REFERENCE}</td>
<td>ref is not a valid \texttt{vx_reference} reference.</td>
</tr>
</tbody>
</table>
3.65 Object: Context

3.65.1 Detailed Description

Defines the Context Object Interface.

The OpenVX context is the object domain for all OpenVX objects. All data objects live in the context as well as all framework objects. The OpenVX context keeps reference counts on all objects and must do garbage collection during its deconstruction to free lost references. While multiple clients may connect to the OpenVX context, all data are private in that the references referring to data objects are given only to the creating party.

Macros

- #define VX_MAX_IMPLEMENTATION_NAME (64)
  Defines the length of the implementation name string, including the trailing zero.

Typedefs

- typedef struct _vx_context *vx_context
  An opaque reference to the implementation context.

Enumerations

- enum vx_accessor_e {
  VX_READ_ONLY = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_ACCESSOR << 12)) + 0x1,
  VX_WRITE_ONLY = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_ACCESSOR << 12)) + 0x2,
  VX_READ_AND_WRITE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_ACCESSOR << 12)) + 0x3
}
  The memory accessor hint flags. These enumeration values are used to indicate desired system behavior, not the User intent. For example: these can be interpreted as hints to the system about cache operations or marshalling operations.

- enum vx_context_attribute_e {
  VX_CONTEXT_VENDOR_ID = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x0,
  VX_CONTEXT_VERSION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x1,
  VX_CONTEXT_UNIQUE_KERNELS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x2,
  VX_CONTEXT_MODULES = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x3,
  VX_CONTEXT_REFERENCES = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x4,
  VX_CONTEXT_IMPLEMENTATION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x5,
  VX_CONTEXT_EXTENSIONS_SIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x6,
  VX_CONTEXT_EXTENSIONS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x7,
  VX_CONTEXT_CONVOLUTION_MAX_DIMENSION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x8,
  VX_CONTEXT_OPTICAL_FLOW_MAX_WINDOW_DIMENSION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x9,
  VX_CONTEXT_IMMEDIATE_BORDER = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0xA,
  VX_CONTEXT_NONLINEAR_MAX_DIMENSION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0xB,
  VX_CONTEXT_MAX_TENSOR_DIMS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0xC
}
  A list of context attributes.

CHAPTER 3. MODULE DOCUMENTATION

• enum vx_memory_type_e {
  VX_MEMORY_TYPE_NONE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_MEMORY_TYPE << 12)) + 0x0,
  VX_MEMORY_TYPE_HOST = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_MEMORY_TYPE << 12)) + 0x1 }

An enumeration of memory import types.

• enum vx_round_policy_e {
  VX_ROUND_POLICY_TO_ZERO = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_ROUND_POLICY << 12)) + 0x1,
  VX_ROUND_POLICY_TO_NEAREST_EVEN = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_ROUND_POLICY << 12)) + 0x2 }

The Round Policy Enumeration.

• enum vx_termination_criteria_e {
  VX_TERM_CRITERIA_ITERATIONS = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_TERM_CRITERIA << 12)) + 0x0,
  VX_TERM_CRITERIA_EPSILON = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_TERM_CRITERIA << 12)) + 0x1,
  VX_TERM_CRITERIA_BOTH = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_TERM_CRITERIA << 12)) + 0x2 }

The termination criteria list.

Functions

• vx_context VX_API_CALL vxCreateContext (void)

  Creates a vx_context.

• vx_context VX_API_CALL vxGetContext (vx_reference reference)

  Retrieves the context from any reference from within a context.

• vx_status VX_API_CALL vxQueryContext (vx_context context, vx_enum attribute, void *ptr, vx_size size)

  Queries the context for some specific information.

• vx_status VX_API_CALL vxReleaseContext (vx_context +context)

  Releases the OpenVX object context.

• vx_status VX_API_CALL vxSetContextAttribute (vx_context context, vx_enum attribute, const void *ptr, vx_size size)

  Sets an attribute on the context.

• vx_status VX_API_CALL vxSetImmediateModeTarget (vx_context context, vx_enum target_enum, const char *target_string)

  Sets the default target of the immediate mode. Upon successful execution of this function any future execution of immediate mode function is attempted on the new default target of the context.

3.65.2 Typedef Documentation

typedef struct _vx_context* vx_context

An opaque reference to the implementation context.

See also

  vxCreateContext

  Definition at line 215 of file vx_types.h.

3.65.3 Enumeration Type Documentation

enum vx_context_attribute_e

A list of context attributes.
**CHAPTER 3. MODULE DOCUMENTATION**

Enumerator

- **VX_CONTEXT_VENDOR_ID**  Queries the unique vendor ID. Read-only. Use a `vx_uint16`.
- **VX_CONTEXT_VERSION**  Queries the OpenVX Version Number. Read-only. Use a `vx_uint16`.
- **VX_CONTEXT_UNIQUE_KERNELS**  Queries the context for the number of *unique* kernels. Read-only. Use a `vx_uint32` parameter.
- **VX_CONTEXT_MODULES**  Queries the context for the number of active modules. Read-only. Use a `vx_uint32` parameter.
- **VX_CONTEXT_REFERENCES**  Queries the context for the number of active references. Read-only. Use a `vx_uint32` parameter.
- **VX_CONTEXT_IMPLEMENTATION**  Queries the context for it's implementation name. Read-only. Use a `vx_char[VX_MAX_IMPLEMENTATION_NAME]` array.
- **VX_CONTEXT_EXTENSIONS_SIZE**  Queries the number of bytes in the extensions string. Read-only. Use a `vx_size` parameter.
- **VX_CONTEXT_EXTENSIONS**  Retrieves the extensions string. Read-only. This is a space-separated string of extension names. Each OpenVX official extension has a unique identifier, comprised of capital letters, numbers and the underscore character, prefixed with "KHR", for example "KHR_NEW_FEATURE". Use a `vx_char` pointer allocated to the size returned from **VX_CONTEXT_EXTENSIONS_SIZE**.
- **VX_CONTEXT_CONVOLUTION_MAX_DIMENSION**  The maximum width or height of a convolution matrix. Read-only. Use a `vx_size` parameter. Each vendor must support centered kernels of size w X h, where both w and h are odd numbers, 3 <= w <= n and 3 <= h <= n, where n is the value of the **VX_CONTEXT_CONVOLUTION_MAX_DIMENSION** attribute. n is an odd number that should not be smaller than 9. w and h may or may not be equal to each other. All combinations of w and h meeting the conditions above must be supported. The behavior of **vxCreateConvolution** is undefined for values larger than the value returned by this attribute.
- **VX_CONTEXT_OPTICAL_FLOW_MAX_WINDOW_DIMENSION**  The maximum window dimension of the OpticalFlowPyrLK kernel. The value of this attribute shall be equal to or greater than '9'. See also

  - **VX_KERNEL_OPTICAL_FLOW_Pyr_LK**. Read-only. Use a `vx_size` parameter.
- **VX_CONTEXT_IMMEDIATE_BORDER**  The border mode for immediate mode functions. Graph mode functions are unaffected by this attribute. Read-write. Use a pointer to a `vx_border_t` structure as parameter.

  - Note  The assumed default value for immediate mode functions is **VX_BORDER_UNDEFINED**.
- **VX_CONTEXT_UNIQUE_KERNEL_TABLE**  Returns the table of all unique the kernels that exist in the context. Read-only. Use a `vx_kernel_info_t` array.

  - Precondition  You must call **vxQueryContext** with **VX_CONTEXT_UNIQUE_KERNELS** to compute the necessary size of the array.
- **VX_CONTEXT_IMMEDIATE_BORDER_POLICY**  The unsupported border mode policy for immediate mode functions. Read-Write. Graph mode functions are unaffected by this attribute. Use a `vx_enum` as parameter. Will contain a `vx_border_policy_e`.

  - Note  The assumed default value for immediate mode functions is **VX_BORDER_POLICY_DEFAULT_TO_UNDEFINED**. Users should refer to the documentation of their implementation to determine what border modes are supported by each kernel.
- **VX_CONTEXT_NONLINEAR_MAX_DIMENSION**  The dimension of the largest nonlinear filter supported. See **vxNonLinearFilterNode**. The implementation must support all dimensions (height or width, not necessarily the same) up to the value of this attribute. The lowest value that must be supported for this attribute is 9. Read-only. Use a `vx_size` parameter.
- **VX_CONTEXT_MAX_TENSOR_DIMS**  tensor Data maximal number of dimensions supported by the implementation.

Definition at line 797 of file `vx_types.h`.
**enum vx_memory_type_e**

An enumeration of memory import types.

**Enumerator**

- **VX_MEMORY_TYPE_NONE**  For memory allocated through OpenVX, this is the import type.
- **VX_MEMORY_TYPE_HOST**  The default memory type to import from the Host.

Definition at line 1223 of file vx_types.h.

**enum vx_termination_criteria_e**

The termination criteria list.

See also

Optical Flow Pyramid (LK)

**Enumerator**

- **VX_TERM_CRITERIA_ITERATIONS**  Indicates a termination after a set number of iterations.
- **VX_TERM_CRITERIA_EPSILON**  Indicates a termination after matching against the value of epsilon provided to the function.
- **VX_TERM_CRITERIA_BOTH**  Indicates that both an iterations and epsilon method are employed. Whichever one matches first causes the termination.

Definition at line 1363 of file vx_types.h.

**enum vx_accessor_e**

The memory accessor hint flags. These enumeration values are used to indicate desired system behavior, not the User intent. For example: these can be interpreted as hints to the system about cache operations or marshalling operations.

**Enumerator**

- **VX_READ_ONLY**  The memory shall be treated by the system as if it were read-only. If the User writes to this memory, the results are implementation defined.
- **VX_WRITE_ONLY**  The memory shall be treated by the system as if it were write-only. If the User reads from this memory, the results are implementation defined.
- **VX_READ_AND_WRITE**  The memory shall be treated by the system as if it were readable and writeable.

Definition at line 1401 of file vx_types.h.

**enum vx_round_policy_e**

The Round Policy Enumeration.

**Enumerator**

- **VX_ROUND_POLICY_TO_ZERO**  When scaling, this truncates the least significant values that are lost in operations.
- **VX_ROUND_POLICY_TO_NEAREST_EVEN**  When scaling, this rounds to nearest even output value.

Definition at line 1418 of file vx_types.h.
### 3.65.4 Function Documentation

**vx_context VX_API_CALL vxCreateContext ( void )**

Creates a `vx_context`. This creates a top-level object context for OpenVX.

**Note**

This is required to do anything else.

**Returns**

The reference to the implementation context `vx_context`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**Postcondition**

`vxReleaseContext`

**vx_status VX_API_CALL vxReleaseContext ( vx_context * context )**

Releases the OpenVX object context.

All reference counted objects are garbage-collected by the return of this call. No calls are possible using the parameter context after the context has been released until a new reference from `vxCreateContext` is returned. All outstanding references to OpenVX objects from this context are invalid after this call.

**Parameters**

| in  | context | The pointer to the reference to the context. |

**Postcondition**

After returning from this function the reference is zeroed.

**Returns**

A `vx_status_e` enumeration.

**Return values**

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>context is not a valid <code>vx_context</code> reference.</td>
</tr>
</tbody>
</table>

**Precondition**

`vxCreateContext`

**vx_context VX_API_CALL vxGetContext ( vx_reference reference )**

Retrieves the context from any reference from within a context.

**Parameters**

| in  | reference | The reference from which to extract the context. |
Returns

The overall context that created the particular reference. Any possible errors preventing a successful completion of this function should be checked using \texttt{vxGetStatus}.

\begin{verbatim}
vx_status VX_API_CALL vxQueryContext ( vx_context context, vx_enum attribute, void * ptr, vx_size size )

describes the context for some specific information.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>The reference to the context.</td>
</tr>
<tr>
<td>attribute</td>
<td>The attribute to query. Use a \texttt{vx_context_attribute_e}.</td>
</tr>
<tr>
<td>ptr</td>
<td>The location at which to store the resulting value.</td>
</tr>
<tr>
<td>size</td>
<td>The size in bytes of the container to which \texttt{ptr} points.</td>
</tr>
</tbody>
</table>

Returns

A \texttt{vx_status_e} enumeration.

Return values

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>context is not a valid \texttt{vx_context} reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>If any of the other parameters are incorrect.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUPPORTED</td>
<td>If the attribute is not supported on this implementation.</td>
</tr>
</tbody>
</table>

\begin{verbatim}
vx_status VX_API_CALL vxSetContextAttribute ( vx_context context, vx_enum attribute, const void * ptr, vx_size size )

describes setting an attribute on the context.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>The handle to the overall context.</td>
</tr>
<tr>
<td>attribute</td>
<td>The attribute to set from \texttt{vx_context_attribute_e}.</td>
</tr>
<tr>
<td>ptr</td>
<td>The pointer to the data to which to set the attribute.</td>
</tr>
<tr>
<td>size</td>
<td>The size in bytes of the data to which \texttt{ptr} points.</td>
</tr>
</tbody>
</table>

Returns

A \texttt{vx_status_e} enumeration.

Return values

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>context is not a valid \texttt{vx_context} reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>If any of the other parameters are incorrect.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUPPORTED</td>
<td>If the attribute is not settable.</td>
</tr>
</tbody>
</table>
**vx_status VX_API_CALL vxSetImmediateModeTarget ( vx_context context, vx_enum target_enum, const char * target_string )**

Sets the default target of the immediate mode. Upon successful execution of this function any future execution of immediate mode function is attempted on the new default target of the context.

**Parameters**

| in | context | The reference to the implementation context. |
| in | target_enum | The default immediate mode target enum to be set to the vx_context object. Use a vx_target_e. |
| in | target_string | The target name ASCII string. This contains a valid value when target_enum is set to VX_TARGET_STRING, otherwise it is ignored. |

**Returns**

A vx_status_e enumeration.

**Return values**

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Default target set; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>If the context is not a valid vx_context reference.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUPPORTED</td>
<td>If the specified target is not supported in this context.</td>
</tr>
</tbody>
</table>
3.66 Object: Graph

3.66.1 Detailed Description

Defines the Graph Object interface.

A set of nodes connected in a directed (only goes one-way) acyclic (does not loop back) fashion. A Graph may have sets of Nodes that are unconnected to other sets of Nodes within the same Graph. See Graph Formalisms. Figure below shows the Graph state transition diagram. Also see vx_graph_state_e.

Figure 3.1: Graph State Transition

Typedefs

- typedef struct _vx_graph * vx_graph
  
  An opaque reference to a graph.

Enumerations

- enum vx_graph_attribute_e {
  
  VX_GRAPH_NUMNODES = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_GRAPH << 8)) + 0x0,
  VX_GRAPH_PERFORMANCE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_GRAPH << 8)) + 0x2,
  VX_GRAPH_NUMPARAMETERS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_GRAPH << 8)) + 0x3,
  VX_GRAPH_STATE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_GRAPH << 8)) + 0x4
  }

  The graph attributes list.

- enum vx_graph_state_e {
  
  VX_GRAPH_STATE_UNVERIFIED = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_GRAPH_STATE << 12)) + 0x0,
  VX_GRAPH_STATE_VERIFIED = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_GRAPH_STATE << 12)) + 0x1,
  VX_GRAPH_STATE_RUNNING = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_GRAPH_STATE << 12)) + 0x2,
  VX_GRAPH_STATE_ABANDONED = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_GRAPH_STATE << 12)) + 0x3,
  VX_GRAPH_STATE_COMPLETED = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_GRAPH_STATE << 12)) + 0x4
  }

  The Graph State Enumeration.
CHAPTER 3. MODULE DOCUMENTATION

Functions

- **vx_graph VX_API_CALL vxCreateGraph (vx_context context)**
  Creates an empty graph.

- **vx_bool VX_API_CALL vxIsGraphVerified (vx_graph graph)**
  Returns a Boolean to indicate the state of graph verification.

- **vx_status VX_API_CALL vxProcessGraph (vx_graph graph)**
  This function causes the synchronous processing of a graph. If the graph has not been verified, then the implementation verifies the graph immediately. If verification fails this function returns a status identical to what `vxVerifyGraph` would return. After the graph verifies successfully then processing occurs. If the graph was previously verified via `vxVerifyGraph` or `vxProcessGraph` then the graph is processed. This function blocks until the graph is completed.

- **vx_status VX_API_CALL vxQueryGraph (vx_graph graph, vx_enum attribute, void *ptr, vx_size size)**
  Allows the user to query attributes of the Graph.

- **vx_status VX_API_CALL vxRegisterAutoAging (vx_graph graph, vx_delay delay)**
  Register a delay for auto-aging.

- **vx_status VX_API_CALL vxReleaseGraph (vx_graph *graph)**
  Releases a reference to a graph. The object may not be garbage collected until its total reference count is zero. Once the reference count is zero, all node references in the graph are automatically released as well. Releasing the graph will only release the nodes if the nodes were not previously released by the application. Data referenced by those nodes may not be released as the user may still have references to the data.

- **vx_status VX_API_CALL vxScheduleGraph (vx_graph graph)**
  Schedules a graph for future execution. If the graph has not been verified, then the implementation verifies the graph immediately. If verification fails this function returns a status identical to what `vxVerifyGraph` would return. After the graph verifies successfully then processing occurs. If the graph was previously verified via `vxVerifyGraph` or `vxProcessGraph` then the graph is processed.

- **vx_status VX_API_CALL vxSetGraphAttribute (vx_graph graph, vx_enum attribute, const void *ptr, vx_size size)**
  Allows the attributes of the Graph to be set to the provided value.

- **vx_status VX_API_CALL vxVerifyGraph (vx_graph graph)**
  Verifies the state of the graph before it is executed. This is useful to catch programmer errors and contract errors. If not verified, the graph verifies before being processed.

- **vx_status VX_API_CALL vxWaitGraph (vx_graph graph)**
  Waits for a specific graph to complete. If the graph has been scheduled multiple times since the last call to `vxWaitGraph`, then `vxWaitGraph` returns only when the last scheduled execution completes.

3.66.2 Typedef Documentation

typedef struct _vx_graph * vx_graph

An opaque reference to a graph.

See also `vxCreateGraph`

Definition at line 208 of file `vx_types.h`.

3.66.3 Enumeration Type Documentation

enum vx_graph_state_e

The Graph State Enumeration.

Enumertor

- **VX_GRAPH_STATE_UNVERIFIED** The graph should be verified before execution.
- **VX_GRAPH_STATE VERIFIED** The graph has been verified and has not been executed or scheduled for execution yet.
**VX_GRAPH_STATE_RUNNING**  The graph either has been scheduled and not completed, or is being executed.

**VX_GRAPH_STATE_ABANDONED**  The graph execution was abandoned.

**VX_GRAPH_STATE_COMPLETED**  The graph execution is completed and the graph is not scheduled for execution.

Definition at line 661 of file `vx_types.h`.

**enum vx_graph_attribute_e**
The graph attributes list.

**Enumerator**

**VX_GRAPH_NUMNODES**  Returns the number of nodes in a graph. Read-only. Use a `vx_uint32` parameter.

**VX_GRAPH_PERFORMANCE**  Returns the overall performance of the graph. Read-only. Use a `vx_perf_t` parameter. The accuracy of timing information is platform dependent.

- **Note**
  Performance tracking must have been enabled. See `vx_directive_e`

**VX_GRAPH_NUMPARAMETERS**  Returns the number of explicitly declared parameters on the graph. Read-only. Use a `vx_uint32` parameter.

**VX_GRAPH_STATE**  Returns the state of the graph. See `vx_graph_state_e` enum.

Definition at line 677 of file `vx_types.h`.

### 3.66.4 Function Documentation

**vx_graph VX_API_CALL vxCreateGraph ( vx_context context )**

Creates an empty graph.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td>context</td>
<td>The reference to the implementation context.</td>
</tr>
</tbody>
</table>

**Returns**

- A graph reference `vx_graph`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx_status VX_API_CALL vxReleaseGraph ( vx_graph * graph )**

Releases a reference to a graph. The object may not be garbage collected until its total reference count is zero. Once the reference count is zero, all node references in the graph are automatically released as well. Releasing the graph will only release the nodes if the nodes were not previously released by the application. Data referenced by those nodes may not be released as the user may still have references to the data.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td>graph</td>
<td>The pointer to the graph to release.</td>
</tr>
</tbody>
</table>

**Postcondition**

- After returning from this function the reference is zeroed.
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>graph is not a valid <code>vx_graph</code> reference.</td>
</tr>
</tbody>
</table>

`vx_status VX_API_CALL vxVerifyGraph ( vx_graph graph )`

Verifies the state of the graph before it is executed. This is useful to catch programmer errors and contract errors. If not verified, the graph verifies before being processed.

Precondition

Memory for data objects is not guaranteed to exist before this call.

Postcondition

After this call data objects exist unless the implementation optimized them out.

Parameters

`in graph` The reference to the graph to verify.

Returns

A status code for graphs with more than one error; it is undefined which error will be returned. Register a log callback using `vxRegisterLogCallback` to receive each specific error in the graph. A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>graph is not a valid <code>vx_graph</code> reference.</td>
</tr>
<tr>
<td>VX_ERROR_MULTIPLE_WRITERS</td>
<td>If the graph contains more than one writer to any data object.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_NODE</td>
<td>If a node in the graph is invalid or failed to be created.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_GRAPH</td>
<td>If the graph contains cycles or some other invalid topology.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_TYPE</td>
<td>If any parameter on a node is given the wrong type.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_VALUE</td>
<td>If any value of any parameter is out of bounds of specification.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_FORMAT</td>
<td>If the image format is not compatible.</td>
</tr>
</tbody>
</table>

See also

`vxProcessGraph`

`vx_status VX_API_CALL vxProcessGraph ( vx_graph graph )`

This function causes the synchronous processing of a graph. If the graph has not been verified, then the implementation verifies the graph immediately. If verification fails this function returns a status identical to what `vxVerifyGraph` would return. After the graph verifies successfully then processing occurs. If the graph was previously verified via `vxVerifyGraph` or `vxProcessGraph` then the graph is processed. This function blocks until the graph is completed.
Parameters

**in** `graph` The graph to execute.

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Graph has been processed; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Graph is not a valid <code>vx_graph</code> reference.</td>
</tr>
<tr>
<td>VX_FAILURE</td>
<td>A catastrophic error occurred during processing.</td>
</tr>
</tbody>
</table>

**vx_status VX_API_CALL vxScheduleGraph ( vx_graph graph )**

Schedules a graph for future execution. If the graph has not been verified, then the implementation verifies the graph immediately. If verification fails this function returns a status identical to what `vxVerifyGraph` would return. After the graph verifies successfully then processing occurs. If the graph was previously verified via `vxVerifyGraph` or `vxProcessGraph` then the graph is processed.

Parameters

**in** `graph` The graph to schedule.

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>The graph has been scheduled; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Graph is not a valid <code>vx_graph</code> reference.</td>
</tr>
<tr>
<td>VX_ERROR_NO_RESOURCES</td>
<td>The graph cannot be scheduled now.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUFFICIENT</td>
<td>The graph is not verified and has failed forced verification.</td>
</tr>
</tbody>
</table>

**vx_status VX_API_CALL vxWaitGraph ( vx_graph graph )**

Waits for a specific graph to complete. If the graph has been scheduled multiple times since the last call to `vxWaitGraph`, then `vxWaitGraph` returns only when the last scheduled execution completes.

Parameters

**in** `graph` The graph to wait on.

Returns

A `vx_status_e` enumeration.
Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VX_SUCCESS</strong></td>
<td>The graph has successfully completed execution and its outputs are the valid results of the most recent execution; any other value indicates failure.</td>
</tr>
<tr>
<td><strong>VX_ERROR_INVALID_REFERENCE</strong></td>
<td>The graph is not a valid <code>vx_graph</code> reference.</td>
</tr>
<tr>
<td><strong>VX_FAILURE</strong></td>
<td>An error occurred or the graph was never scheduled. Output data of the graph is undefined.</td>
</tr>
</tbody>
</table>

Precondition

`vxScheduleGraph`

`vx_status VX_API_CALL vxQueryGraph ( vx_graph graph, vx_enum attribute, void * ptr, vx_size size )`

Allows the user to query attributes of the Graph.

Parameters

- **in graph** The reference to the created graph.
- **in attribute** The `vx_graph_attribute_e` type needed.
- **out ptr** The location at which to store the resulting value.
- **in size** The size in bytes of the container to which `ptr` points.

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VX_SUCCESS</strong></td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td><strong>VX_ERROR_INVALID_REFERENCE</strong></td>
<td>The graph is not a valid <code>vx_graph</code> reference.</td>
</tr>
</tbody>
</table>

`vx_status VX_API_CALL vxSetGraphAttribute ( vx_graph graph, vx_enum attribute, const void * ptr, vx_size size )`

Allows the attributes of the Graph to be set to the provided value.

Parameters

- **in graph** The reference to the graph.
- **in attribute** The `vx_graph_attribute_e` type needed.
- **in ptr** The location from which to read the value.
- **in size** The size in bytes of the container to which `ptr` points.

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VX_SUCCESS</strong></td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td><strong>VX_ERROR_INVALID_REFERENCE</strong></td>
<td>The graph is not a valid <code>vx_graph</code> reference.</td>
</tr>
</tbody>
</table>
vx_bool VX_API_CALL vxIsGraphVerified ( vx_graph graph )
Returns a Boolean to indicate the state of graph verification.

Parameters

| in | graph | The reference to the graph to check. |

Returns

A vx_bool value.

Return values

| vx_true_e | The graph is verified. |
| vx_false_e | The graph is not verified. It must be verified before execution either through vxVerifyGraph or automatically through vxProcessGraph or vxScheduleGraph. |

vx_status VX_API_CALL vxRegisterAutoAging ( vx_graph graph, vx_delay delay )
Register a delay for auto-aging.
This function registers a delay object to be auto-aged by the graph. This delay object will be automatically aged after each successful completion of this graph. Aging of a delay object cannot be called during graph execution. A graph abandoned due to a node callback will trigger an auto-aging.

If a delay is registered for auto-aging multiple times in a same graph, the delay will be only aged a single time at each graph completion. If a delay is registered for auto-aging in multiple graphs, this delay will aged automatically after each successful completion of any of these graphs.

Parameters

| in | graph | The graph to which the delay is registered for auto-aging. |
| in | delay | The delay to automatically age. |

Returns

A vx_status_e enumeration.

Return values

| VX_SUCCESS | No errors; any other value indicates failure. |
| VX_ERROR_INVALID_REFERENCE | graph is not a valid vx_graph reference, or delay is not a valid vx_delay reference. |
3.67 Object: Node

3.67.1 Detailed Description

Defines the Node Object interface.

A node is an instance of a kernel that will be paired with a specific set of references (the parameters). Nodes are created from and associated with a single graph only. When a `vx_parameter` is extracted from a Node, an additional attribute can be accessed:

- **Reference** - The `vx_reference` assigned to this parameter index from the Node creation function (e.g., `vxSobel3x3Node`).

**Typedefs**

- `typedef struct _vx_node *vx_node`  
  An opaque reference to a kernel node.

**Enumerations**

- `enum vx_node_attribute_e {
  VX_NODE_STATUS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x0,
  VX_NODE_PERFORMANCE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x1,
  VX_NODE_BORDER = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x2,
  VX_NODE_LOCAL_DATA_SIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x3,
  VX_NODE_LOCAL_DATA_PTR = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x4,
  VX_NODE_PARAMETERS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x5,
  VX_NODE_IS_REPLICATED = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x6,
  VX_NODE_REPLICATE_FLAGS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x7,
  VX_NODE_VALID_RECT_RESET = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x8
}

The node attributes list.

**Functions**

- `vx_status VX_API_CALL vxQueryNode (vx_node node, vx_enum attribute, void *ptr, vx_size size)`  
  Allows a user to query information out of a node.

- `vx_status VX_API_CALL vxReleaseNode (vx_node *node)`  
  Releases a reference to a Node object. The object may not be garbage collected until its total reference count is zero.

- `vx_status VX_API_CALL vxRemoveNode (vx_node *node)`  
  Removes a Node from its parent Graph and releases it.

- `vx_status VX_API_CALL vxReplicateNode (vx_graph graph, vx_node first_node, vx_bool replicate[], vx_uint32 number_of_parameters)`  
  Creates replicas of the same node first_node to process a set of objects stored in `vx_pyramid` or `vx_object_array`. first_node needs to have as parameter levels 0 of a `vx_pyramid` or the index 0 of a `vx_object_array`. Replica nodes are not accessible by the application through any means. An application request for removal of first_node from the graph will result in removal of all replicas. Any change of parameter or attribute of first_node will be propagated to the replicas. `vxVerifyGraph` shall enforce consistency of parameters and attributes in the replicas.

- `vx_status VX_API_CALL vxSetNodeAttribute (vx_node node, vx_enum attribute, const void *ptr, vx_size size)`  
  Allows a user to set attribute of a node before Graph Validation.

- `vx_status VX_API_CALL vxSetNodeTarget (vx_node node, vx_enum target_enum, const char *target_string)`  
  Sets the node target to the provided value. A success invalidates the graph that the node belongs to (`vxVerifyGraph` must be called before the next execution).
3.67.2 Typedef Documentation

typedef struct _vx_node* vx_node

An opaque reference to a kernel node.

See also

vxCreateGenericNode

Definition at line 201 of file vx_types.h.

3.67.3 Enumeration Type Documentation

enum vx_node_attribute_e

The node attributes list.

Enumerator

- **VX_NODE_STATUS**: Queries the status of node execution. Read-only. Use a `vx_status` parameter.
- **VX_NODE_PERFORMANCE**: Queries the performance of the node execution. The accuracy of timing information is platform dependent and also depends on the graph optimizations. Read-only.

  Note

  Performance tracking must have been enabled. See `vx_directive_e`.

- **VX_NODE_BORDER**: Gets or sets the border mode of the node. Read-write. Use a `vx_border_t` structure with a default value of `VX_BORDER_UNDEFINED`.

- **VX_NODE_LOCAL_DATA_SIZE**: Indicates the size of the kernel local memory area. Read-only. Can be written only at user-node (de)initialization if `VX_KERNEL_LOCAL_DATA_SIZE==0`. Use a `vx_size` parameter.

- **VX_NODE_LOCAL_DATA_PTR**: Indicates the pointer kernel local memory area. Read-Write. Can be written only at user-node (de)initialization if `VX_KERNEL_LOCAL_DATA_SIZE==0`. Use a `void*` parameter.

- **VX_NODE_PARAMETERS**: Indicates the number of node parameters, including optional parameters that are not passed. Read-only. Use a `vx_uint32` parameter.

- **VX_NODE_IS_REPLICATED**: Indicates whether the node is replicated. Read-only. Use a `vx_bool` parameter.

- **VX_NODE_REPLICATE_FLAGS**: Indicates the replicated parameters. Read-only. Use a `vx_bool*` parameter.

- **VX_NODE_VALID_RECT_RESET**: Indicates the behavior with respect to the valid rectangle. Read-only. Use a `vx_bool` parameter.

Definition at line 886 of file vx_types.h.

3.67.4 Function Documentation

vx_status VX_API_CALL vxQueryNode ( vx_node node, vx_enum attribute, void* ptr, vx_size size )

Allows a user to query information out of a node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>node</th>
<th>The reference to the node to query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>attribute</td>
<td>Use <code>vx_node_attribute_e</code> value to query for information.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The location at which to store the resulting value.</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>The size in bytes of the container to which <code>ptr</code> points.</td>
</tr>
</tbody>
</table>
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Node is not a valid <code>vx_node</code> reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>The type or size is incorrect.</td>
</tr>
</tbody>
</table>

`vx_status VX_API_CALL vxSetNodeAttribute ( vx_node node, vx_enum attribute, const void *ptr, vx_size size )`

Allows a user to set attribute of a node before Graph Validation.

Parameters

| in | node | The reference to the node to set. |
| in | attribute | Use `vx_node_attribute_e` value to set the desired attribute. |
| in | ptr | The pointer to the desired value of the attribute. |
| in | size | The size in bytes of the objects to which `ptr` points. |

Note

Some attributes are inherited from the `vx_kernel`, which was used to create the node. Some of these can be overridden using this API, notably `VX_NODE_LOCAL_DATA_SIZE` and `VX_NODE_LOCAL_DATA_PTR`.

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>The attribute was set; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Node is not a valid <code>vx_node</code> reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>Size is not correct for the type needed.</td>
</tr>
</tbody>
</table>

`vx_status VX_API_CALL vxReleaseNode ( vx_node *node )`

Releases a reference to a Node object. The object may not be garbage collected until its total reference count is zero.

Parameters

| in | node | The pointer to the reference of the node to release. |

Postcondition

After returning from this function the reference is zeroed.

Returns

A `vx_status_e` enumeration.
Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>node is not a valid vx_node reference.</td>
</tr>
</tbody>
</table>

**vx_status VX_API_CALL vxRemoveNode ( vx_node * node )**

Removes a Node from its parent Graph and releases it.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>node</td>
<td>The pointer to the node to remove and release.</td>
</tr>
</tbody>
</table>

**Postcondition**

After returning from this function the reference is zeroed.

**Returns**

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Node target set; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>node is not a valid vx_node reference.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUPPORTED</td>
<td>If the node kernel is not supported by the specified target.</td>
</tr>
</tbody>
</table>

**vx_status VX_API_CALL vxSetNodeTarget ( vx_node node, vx_enum target_enum, const char * target_string )**

Sets the node target to the provided value. A success invalidates the graph that the node belongs to (vxVerifyGraph must be called before the next execution)

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>node</td>
<td>The reference to the vx_node object.</td>
</tr>
<tr>
<td>in</td>
<td>target_enum</td>
<td>The target enum to be set to the vx_node object. Use a vx_target_e.</td>
</tr>
<tr>
<td>in</td>
<td>target_string</td>
<td>The target name ASCII string. This contains a valid value when target_enum is set to VX_TARGET_STRING, otherwise it is ignored.</td>
</tr>
</tbody>
</table>

**Returns**

A vx_status_e enumeration.
\texttt{vx\_status VX\_API\_CALL vxReplicateNode ( vx\_graph graph, vx\_node first\_node, vx\_bool replicate[], vx\_uint32 number\_of\_parameters )}

Creates replicas of the same node \texttt{first\_node} to process a set of objects stored in \texttt{vx\_pyramid} or \texttt{vx\_object\_array}. \texttt{first\_node} needs to have as parameter levels 0 of a \texttt{vx\_pyramid} or the index 0 of a \texttt{vx\_object\_array}. Replica nodes are not accessible by the application through any means. An application request for removal of \texttt{first\_node} from the graph will result in removal of all replicas. Any change of parameter or attribute of \texttt{first\_node} will be propagated to the replicas. \texttt{vxVerifyGraph} shall enforce consistency of parameters and attributes in the replicas.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>first_node</td>
<td>The reference to the node in the graph that will be replicated.</td>
</tr>
<tr>
<td>in</td>
<td>replicate</td>
<td>an array of size equal to the number of node parameters, \texttt{vx_true_e} for the parameters that should be iterated over (should be a reference to a \texttt{vx_pyramid} or a \texttt{vx_object_array}), \texttt{vx_false_e} for the parameters that should be the same across replicated nodes and for optional parameters that are not used. Should be \texttt{vx_true_e} for all output and bidirectional parameters.</td>
</tr>
<tr>
<td>in</td>
<td>number_of_parameters</td>
<td>number of elements in the replicate array</td>
</tr>
</tbody>
</table>

**Returns**

A \texttt{vx\_status\_e} enumeration.

**Return values**

<table>
<thead>
<tr>
<th>\texttt{VX_SUCCESS}</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{VX_ERROR_INVALID_REFERENCE}</td>
<td>graph is not a valid \texttt{vx_graph} reference, or \texttt{first_node} is not a valid \texttt{vx_node} reference.</td>
</tr>
<tr>
<td>\texttt{VX_ERROR_NOT_COMPATIBLE}</td>
<td>At least one of replicated parameters is not of level 0 of a pyramid or at index 0 of an object array.</td>
</tr>
<tr>
<td>\texttt{VX_FAILURE}</td>
<td>If the node does not belong to the graph, or the number of objects in the parent objects of inputs and output are not the same.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.68 Object: Array

3.68.1 Detailed Description

Defines the Array Object Interface.

Array is a strongly-typed container, which provides random access by index to its elements in constant time. It uses value semantics for its own elements and holds copies of data. This is an example for loop over an Array:

```c
vx_size i, stride = sizeof(vx_size);
void *base = NULL;
vx_map_id map_id; /* access entire array at once */
vxMapArrayRange(array, 0, num_items, &map_id, &stride, &base,
VX_READ_AND_WRITE, VX_MEMORY_TYPE_HOST, 0);
for (i = 0; i < num_items; i++)
{   
  vxArrayItem(mystruct, base, i, stride).some_uint += i;
  vxArrayItem(mystruct, base, i, stride).some_double = 3.14f;
}
vxUnmapArrayRange(array, map_id);
```

Macros

- `#define vxArrayItem(type, ptr, index, stride) (∗(type ∗)(vxFormatArrayPointer((ptr), (index), (stride))))`
  Allows access to an array item as a typcast pointer deference.

- `#define vxFormatArrayPointer(ptr, index, stride) (&(((vx_uint8 ∗)(ptr))[(index)∗(stride)]))`
  Accesses a specific indexed element in an array.

Typedefs

- `typedef struct _vx_array ∗vx_array`
  The Array Object. Array is a strongly-typed container for other data structures.

Enumerations

- `enum vx_array_attribute_e {
  VX_ARRAY_ITEMTYPE = ((( VX_ID_KHRONOS )<<20) | ( VX_TYPE_ARRAY << 8)) + 0x0,
  VX_ARRAY_NUMITEMS = ((( VX_ID_KHRONOS )<<20) | ( VX_TYPE_ARRAY << 8)) + 0x1,
  VX_ARRAY_CAPACITY = ((( VX_ID_KHRONOS )<<20) | ( VX_TYPE_ARRAY << 8)) + 0x2,
  VX_ARRAY_ITEMSIZE = ((( VX_ID_KHRONOS )<<20) | ( VX_TYPE_ARRAY << 8)) + 0x3 }
  The array object attributes.

Functions

- `vx_status VX_API_CALL vxAddArrayItems (vx_array arr, vx_size count, const void ∗ptr, vx_size stride)`
  Adds items to the Array.

- `vx_status VX_API_CALL vxCopyArrayRange (vx_array array, vx_size range_start, vx_size range_end, vx_size user_stride, void ∗user_ptr, vx_enum usage, vx_enum user_mem_type)`
  Allows the application to copy a range from/into an array object.

- `vx_array VX_API_CALL vxCreateArray (vx_context context, vx_enum item_type, vx_size capacity)`
  Creates a reference to an Array object.

- `vx_array VX_API_CALL vxCreateVirtualArray (vx_graph graph, vx_enum item_type, vx_size capacity)`
  Creates an opaque reference to a virtual Array with no direct user access.

- `vx_status VX_API_CALL vxMapArrayRange (vx_array array, vx_size range_start, vx_size range_end, vx_map_id ∗map_id, vx_size ∗stride, void ∗∗ptr, vx_enum usage, vx_enum mem_type, vx_uint32 flags)`
  Allows the application to get direct access to a range of an array object.

- `vx_status VX_API_CALL vxQueryArray (vx_array arr, vx_enum attribute, void ∗∗ptr, vx_size size)`
  Queries the Array for some specific information.

- `vx_status VX_API_CALL vxReleaseArray (vx_array ∗arr)`
Releases a reference of an Array object. The object may not be garbage collected until its total reference count is zero. After returning from this function the reference is zeroed.

- `vx_status VX_API_CALL vxTruncateArray (vx_array arr, vx_size new_num_items)`
  Truncates an Array (remove items from the end).

- `vx_status VX_API_CALL vxUnmapArrayRange (vx_array array, vx_map_id map_id)`
  Unmap and commit potential changes to an array object range that was previously mapped. Unmapping an array range invalidates the memory location from which the range could be accessed by the application. Accessing this memory location after the unmap function completes has an undefined behavior.

### 3.68.2 Macro Definition Documentation

```c
#define vxFormatArrayPointer( ptr, index, stride ) (&(((vx_uint8 ∗)(ptr))[(index) ∗ (stride)]))
```

Accesses a specific indexed element in an array.

**Parameters**

- **ptr**: The base pointer for the array range.
- **index**: The index of the element, not byte, to access.
- **stride**: The ‘number of bytes’ between the beginning of two consecutive elements.

Definition at line 2876 of file vx_api.h.

```c
#define vxArrayItem( type, ptr, index, stride ) ((type ∗)(vxFormatArrayPointer((ptr), (index), (stride))))
```

Allows access to an array item as a typecast pointer dereference.

**Parameters**

- **type**: The type of the item to access.
- **ptr**: The base pointer for the array range.
- **index**: The index of the element, not byte, to access.
- **stride**: The ‘number of bytes’ between the beginning of two consecutive elements.

Definition at line 2887 of file vx_api.h.

### 3.68.3 Enumeration Type Documentation

```c
enum vx_array_attribute_e
```

The array object attributes.

**Enumerator**

- **VX_ARRAY_ITEMTYPE**: The type of the Array items. Read-only. Use a `vx_enum` parameter.
- **VX_ARRAY_NUMITEMS**: The number of items in the Array. Read-only. Use a `vx_size` parameter.
- **VX_ARRAY_CAPACITY**: The maximal number of items that the Array can hold. Read-only. Use a `vx_size` parameter.
- **VX_ARRAY_ITEMSIZE**: Queries an array item size. Read-only. Use a `vx_size` parameter.

Definition at line 1145 of file vx_types.h.

### 3.68.4 Function Documentation

```c
vx_array VX_API_CALL vxCreateArray ( vx_context context, vx_enum item_type, vx_size capacity )
```

Creates a reference to an Array object.

User must specify the Array capacity (i.e., the maximal number of items that the array can hold).
### CHAPTER 3. MODULE DOCUMENTATION

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall Context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>item_type</td>
<td>The type of data to hold. Must be greater than VX_TYPE_INVALID and less than or equal to VX_TYPE_VENDOR_STRUCT_END. Or must be a vx_enum returned from vxRegisterUserStruct.</td>
</tr>
<tr>
<td>in</td>
<td>capacity</td>
<td>The maximal number of items that the array can hold. This value must be greater than zero.</td>
</tr>
</tbody>
</table>

#### Returns

An array reference `vx_array`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx_array VX_API_CALL vxCreateVirtualArray ( vx_graph graph, vx_enum item_type, vx_size capacity )**

Creates an opaque reference to a virtual Array with no direct user access. Virtual Arrays are useful when item type or capacity are unknown ahead of time and the Array is used as internal graph edge. Virtual arrays are scoped within the parent graph only. All of the following constructions are allowed.

```c
00001 vx_context context = vxCreateContext();
00002 vx_graph graph = vxCreateGraph(context);
00003 vx_array virt[] = {
00004     vxCreateVirtualArray(graph, 0, 0), // totally unspecified
00005     vxCreateVirtualArray(graph, VX_TYPE_KEYPOINT, 0), // unspecified capacity
00006     vxCreateVirtualArray(graph, VX_TYPE_KEYPOINT, 1000), // no access
00007 }
```

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the parent graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>item_type</td>
<td>The type of data to hold. Must be greater than VX_TYPE_INVALID and less than or equal to VX_TYPE_VENDOR_STRUCT_END. Or must be a vx_enum returned from vxRegisterUserStruct. This may be set to zero to indicate an unspecified item type.</td>
</tr>
<tr>
<td>in</td>
<td>capacity</td>
<td>The maximal number of items that the array can hold. This may be set to zero to indicate an unspecified capacity.</td>
</tr>
</tbody>
</table>

#### See also

`vxCreateArray` for a type list.

#### Returns

A array reference `vx_array`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx_status VX_API_CALL vxReleaseArray ( vx_array * arr )**

Releases a reference of an Array object. The object may not be garbage collected until its total reference count is zero. After returning from this function the reference is zeroed.

#### Parameters

| in  | arr | The pointer to the Array to release. |
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>arr is not a valid <code>vx_array</code> reference.</td>
</tr>
</tbody>
</table>

`vx_status VX_API_CALL vxQueryArray ( vx_array arr, vx_enum attribute, void * ptr, vx_size size )`

Queries the Array for some specific information.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> arr</td>
<td>The reference to the Array.</td>
</tr>
<tr>
<td><strong>in</strong> attribute</td>
<td>The attribute to query. Use a <code>vx_array_attribute_e</code>.</td>
</tr>
<tr>
<td><strong>out</strong> ptr</td>
<td>The location at which to store the resulting value.</td>
</tr>
<tr>
<td><strong>in</strong> size</td>
<td>The size in bytes of the container to which <code>ptr</code> points.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>arr is not a valid <code>vx_array</code> reference.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUPPORTED</td>
<td>If the <code>attribute</code> is not a value supported on this implementation.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>If any of the other parameters are incorrect.</td>
</tr>
</tbody>
</table>

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

Adds items to the Array.

This function increases the container size.

By default, the function does not reallocate memory, so if the container is already full (number of elements is equal to capacity) or it doesn't have enough space, the function returns `VX_FAILURE` error code.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> arr</td>
<td>The reference to the Array.</td>
</tr>
<tr>
<td><strong>in</strong> count</td>
<td>The total number of elements to insert.</td>
</tr>
<tr>
<td><strong>in</strong> ptr</td>
<td>The location from which to read the input values.</td>
</tr>
<tr>
<td><strong>in</strong> stride</td>
<td>The number of bytes between the beginning of two consecutive elements.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>arr is not a valid vx_array reference.</td>
</tr>
<tr>
<td>VX_FAILURE</td>
<td>If the Array is full.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>If any of the other parameters are incorrect.</td>
</tr>
</tbody>
</table>

vx_status VX_API_CALL vxTruncateArray ( vx_array arr, vx_size new_num_items )

Truncates an Array (remove items from the end).

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>arr</td>
</tr>
<tr>
<td>in</td>
<td>new_num_items</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>arr is not a valid vx_array reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>The new_size is greater than the current size.</td>
</tr>
</tbody>
</table>

vx_status VX_API_CALL vxCopyArrayRange ( vx_array array, vx_size range_start, vx_size range_end, vx_size user_stride, void * user_ptr, vx_enum usage, vx_enum user_mem_type )

Allows the application to copy a range from/into an array object.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>array</td>
</tr>
<tr>
<td>in</td>
<td>range_start</td>
</tr>
<tr>
<td>in</td>
<td>range_end</td>
</tr>
<tr>
<td>in</td>
<td>user_stride</td>
</tr>
<tr>
<td>in</td>
<td>user_ptr</td>
</tr>
</tbody>
</table>

The address of the memory location where to store the requested data if the copy was requested in read mode, or from where to get the data to store into the array object if the copy was requested in write mode. The accessible memory must be large enough to contain the specified range with the specified stride: accessible memory in bytes >= (range_end range_start) * user_stride.
### Parameters

<table>
<thead>
<tr>
<th><strong>in</strong></th>
<th><strong>usage</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
</table>
|        | **vx_accessor_e** enumeration | This declares the effect of the copy with regard to the array object using the **vx_accessor_e** enumeration. Only **VX_READ_ONLY** and **VX_WRITE_ONLY** are supported:  
- **VX_READ_ONLY** means that data are copied from the array object into the user memory.  
- **VX_WRITE_ONLY** means that data are copied into the array object from the user memory. |

<table>
<thead>
<tr>
<th><strong>in</strong></th>
<th><strong>user_mem_type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>vx_memory_type_e</strong> enumeration</td>
<td>A <strong>vx_memory_type_e</strong> enumeration that specifies the memory type of the memory referenced by the user_addr.</td>
</tr>
</tbody>
</table>

### Returns

A **vx_status_e** enumeration.

### Return values

<table>
<thead>
<tr>
<th><strong>vx_status_e</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VX_SUCCESS</strong></td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td><strong>VX_ERROR_OPTIMIZED_AWAY</strong></td>
<td>This is a reference to a virtual array that cannot be accessed by the application.</td>
</tr>
<tr>
<td><strong>VX_ERROR_INVALID_REFERENCE</strong></td>
<td>Array is not a valid <strong>vx_array</strong> reference.</td>
</tr>
<tr>
<td><strong>VX_ERROR_INVALID_PARAMETERS</strong></td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

### vx_status VX_API_CALL vxMapArrayRange ( **vx_array** array, **vx_size** range_start, **vx_size** range_end, **vx_map_id** *map_id*, **vx_size** *stride*, void **ptr**, **vx_enum** usage, **vx_enum** mem_type, **vx_uint32** flags )

Allows the application to get direct access to a range of an array object.

### Parameters

<table>
<thead>
<tr>
<th><strong>in</strong></th>
<th><strong>array</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>The reference to the array object that contains the range to map.</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>in</strong></th>
<th><strong>range_start</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>The index of the first item of the array object to map.</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>in</strong></th>
<th><strong>range_end</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>The index of the item following the last item of the array object to map. (range_end - range_start) items are mapped, starting from index range_start included. The range must be within the bounds of the array: Must be 0 &lt;= range_start &lt; range_end &lt;= number of items.</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>out</strong></th>
<th><strong>map_id</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
</table>
|         | **The address of a vx_map_id variable where the function returns a map identifier.**  
- (**map_id**) must eventually be provided as the map_id parameter of a call to **vxUnmapArrayRange**. |

<table>
<thead>
<tr>
<th><strong>out</strong></th>
<th><strong>stride</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>The address of a vx_size variable where the function returns the memory layout of the mapped array range. The function sets (<strong>stride</strong>) to the number of bytes between the beginning of two consecutive items. The application must consult (<strong>stride</strong>) to access the array items starting from address (<strong>ptr</strong>). The layout of the mapped array follows an item major order: (<strong>stride</strong>) &gt;= item size in bytes.</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>out</strong></th>
<th><strong>ptr</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>The address of a pointer that the function sets to the address where the requested data can be accessed. The returned (<strong>ptr</strong>) address is only valid between the call to the function and the corresponding call to <strong>vxUnmapArrayRange</strong>.</strong></td>
<td></td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usage</th>
<th>This declares the access mode for the array range, using the vx_accessor_e enumeration.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• <strong>VX_READ_ONLY</strong>: after the function call, the content of the memory location pointed by ((\ast ptr)) contains the array range data. Writing into this memory location is forbidden and its behavior is undefined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>VX_READ_AND_WRITE</strong>: after the function call, the content of the memory location pointed by ((\ast ptr)) contains the array range data; writing into this memory is allowed only for the location of items and will result in a modification of the affected items in the array object once the range is unmapped. Writing into a gap between items (when ((\ast stride) &gt; \text{item size in bytes})) is forbidden and its behavior is undefined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>VX_WRITE_ONLY</strong>: after the function call, the memory location pointed by ((\ast ptr)) contains undefined data; writing each item of the range is required prior to unmapping. Items not written by the application before unmap will become undefined after unmap, even if they were well defined before map. Like for <strong>VX_READ_AND_WRITE</strong>, writing into a gap between items is forbidden and its behavior is undefined.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>in</th>
<th>mem_type</th>
<th>A vx_memory_type_e enumeration that specifies the type of the memory where the array range is requested to be mapped.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>flags</td>
<td>An integer that allows passing options to the map operation. Use the vx_map_flag_e enumeration.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_OPTIMIZED_AWAY</td>
<td>This is a reference to a virtual array that cannot be accessed by the application.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>array is not a valid vx_array reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

Postcondition

vxUnmapArrayRange with same \((\ast map\_id)\) value.

**vx_status VX_API_CALL vxUnmapArrayRange ( vx_array array, vx_map_id map_id )**

Unmap and commit potential changes to an array object range that was previously mapped. Unmapping an array range invalidates the memory location from which the range could be accessed by the application. Accessing this memory location after the unmap function completes has an undefined behavior.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>array</th>
<th>The reference to the array object to unmap.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>map_id</td>
<td>The unique map identifier that was returned when calling vxMapArrayRange.</td>
</tr>
</tbody>
</table>
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>array is not a valid <code>vx_array</code> reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

Precondition

`vxMapArrayRange` returning the same `map_id` value
3.69  Object: Convolution

3.69.1  Detailed Description

Defines the Image Convolution Object interface.

Typedefs

- typedef struct _vx_convolution ∗ vx_convolution
  The Convolution Object. A user-defined convolution kernel of MxM elements.

Enumerations

- enum vx_convolution_attribute_e {
  VX_CONVOLUTION_ROWS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONVOLUTION << 8)) + 0x0,
  VX_CONVOLUTION_COLUMNS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONVOLUTION << 8)) + 0x1,
  VX_CONVOLUTION_SCALE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONVOLUTION << 8)) + 0x2,
  VX_CONVOLUTION_SIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONVOLUTION << 8)) + 0x3
}
  The convolution attributes.

Functions

- vx_status VX_API_CALL vxCopyConvolutionCoefficients (vx_convolution conv, void ∗user_ptr, vx_enum usage, vx_enum user_mem_type)
  Allows the application to copy coefficients from/into a convolution object.

- vx_convolution VX_API_CALL vxCreateConvolution (vx_context context, vx_size columns, vx_size rows)
  Creates a reference to a convolution matrix object.

- vx_convolution VX_API_CALL vxCreateVirtualConvolution (vx_graph graph, vx_size columns, vx_size rows)
  Creates an opaque reference to a convolution matrix object without direct user access.

- vx_status VX_API_CALL vxQueryConvolution (vx_convolution conv, vx_enum attribute, void ∗ptr, vx_size size)
  Queries an attribute on the convolution matrix object.

- vx_status VX_API_CALL vxReleaseConvolution (vx_convolution ∗conv)
  Releases the reference to a convolution matrix. The object may not be garbage collected until its total reference count is zero.

- vx_status VX_API_CALL vxSetConvolutionAttribute (vx_convolution conv, vx_enum attribute, const void ∗ptr, vx_size size)
  Sets attributes on the convolution object.

3.69.2  Enumeration Type Documentation

enum vx_convolution_attribute_e

The convolution attributes.

Enumerator

  VX_CONVOLUTION_ROWS  The number of rows of the convolution matrix. Read-only. Use a vx_size parameter.

  VX_CONVOLUTION_COLUMNS  The number of columns of the convolution matrix. Read-only. Use a vx_size parameter.

  VX_CONVOLUTION_SCALE  The scale of the convolution matrix. Read-write. Use a vx_uint32 parameter.
Note
For 1.0, only powers of 2 are supported up to $2^{31}$.

\textit{VX\_CONVOLUTION\_SIZE}  The total size of the convolution matrix in bytes. Read-only. Use a \textit{vx\_size} parameter.

Definition at line 1097 of file \textit{vx\_types.h}.

### 3.69.3 Function Documentation

\texttt{vx\_convolution} \texttt{VX\_API\_CALL} \texttt{vxCreateConvolution} ( \texttt{vx\_context context}, \texttt{vx\_size columns}, \texttt{vx\_size rows} )

Creates a reference to a convolution matrix object.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>columns</td>
<td>The columns dimension of the convolution. Must be odd and greater than or equal to 3 and less than the value returned from \texttt{VX_CONTEXT_CONVOLUTION_MAX_DIMENSION}.</td>
</tr>
<tr>
<td>in</td>
<td>rows</td>
<td>The rows dimension of the convolution. Must be odd and greater than or equal to 3 and less than the value returned from \texttt{VX_CONTEXT_CONVOLUTION_MAX_DIMENSION}.</td>
</tr>
</tbody>
</table>

**Returns**

A convolution reference \texttt{vx\_convolution}. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus}.

\texttt{vx\_convolution} \texttt{VX\_API\_CALL} \texttt{vxCreateVirtualConvolution} ( \texttt{vx\_graph graph}, \texttt{vx\_size columns}, \texttt{vx\_size rows} )

Creates an opaque reference to a convolution matrix object without direct user access.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the parent graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>columns</td>
<td>The columns dimension of the convolution. Must be odd and greater than or equal to 3 and less than the value returned from \texttt{VX_CONTEXT_CONVOLUTION_MAX_DIMENSION}.</td>
</tr>
<tr>
<td>in</td>
<td>rows</td>
<td>The rows dimension of the convolution. Must be odd and greater than or equal to 3 and less than the value returned from \texttt{VX_CONTEXT_CONVOLUTION_MAX_DIMENSION}.</td>
</tr>
</tbody>
</table>

**See also**

\texttt{vxCreateConvolution}

**Returns**

A convolution reference \texttt{vx\_convolution}. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus}.

\texttt{vx\_status} \texttt{VX\_API\_CALL} \texttt{vxReleaseConvolution} ( \texttt{vx\_convolution \* conv} )

Releases the reference to a convolution matrix. The object may not be garbage collected until its total reference count is zero.

**Parameters**

| in     | conv | The pointer to the convolution matrix to release. |

Postcondition

After returning from this function the reference is zeroed.

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td><code>conv</code> is not a valid <code>vx_convolution</code> reference.</td>
</tr>
</tbody>
</table>

```c
vx_status VX_API_CALL vxQueryConvolution ( vx_convolution conv, vx_enum attribute, void * ptr, vx_size size )
```

Queries an attribute on the convolution matrix object.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>conv</td>
<td>The convolution matrix object to set.</td>
</tr>
<tr>
<td>in</td>
<td>attribute</td>
<td>The attribute to query. Use a <code>vx_convolution_attribute_e</code> enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The location at which to store the resulting value.</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>The size in bytes of the container to which <code>ptr</code> points.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td><code>conv</code> is not a valid <code>vx_convolution</code> reference.</td>
</tr>
</tbody>
</table>

```c
vx_status VX_API_CALL vxSetConvolutionAttribute ( vx_convolution conv, vx_enum attribute, const void * ptr, vx_size size )
```

Sets attributes on the convolution object.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>conv</td>
<td>The coordinates object to set.</td>
</tr>
<tr>
<td>in</td>
<td>attribute</td>
<td>The attribute to modify. Use a <code>vx_convolution_attribute_e</code> enumeration.</td>
</tr>
<tr>
<td>in</td>
<td>ptr</td>
<td>The pointer to the value to which to set the attribute.</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>The size in bytes of the data pointed to by <code>ptr</code>.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
</tbody>
</table>
Return values

| VX_ERROR_INVALID_REFERENCE | conv is not a valid vx_convolution reference. |

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

Allows the application to copy coefficients from/into a convolution object.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>conv</th>
<th>The reference to the convolution object that is the source or the destination of the copy.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>user_ptr</td>
<td>The address of the memory location where to store the requested coefficient data if the copy was requested in read mode, or from where to get the coefficient data to store into the convolution object if the copy was requested in write mode. In the user memory, the convolution coefficient data is structured as a row-major 2D array with elements of the type corresponding to VX_TYPE_CONVOLUTION, with a number of rows corresponding to VX_CONVOLUTION_ROWS and a number of columns corresponding to VX_CONVOLUTION_COLUMNS. The accessible memory must be large enough to contain this 2D array: accessible memory in bytes ( \geq ) ( \text{sizeof(data	extunderscore element)} \times \text{rows} \times \text{columns} ).</td>
</tr>
</tbody>
</table>
| in   | usage                 | This declares the effect of the copy with regard to the convolution object using the vx_accessor_e enumeration. Only VX_READ_ONLY and VX_WRITE_ONLY are supported:  
  - VX_READ_ONLY means that data are copied from the convolution object into the user memory.  
  - VX_WRITE_ONLY means that data are copied into the convolution object from the user memory. |
| in   | user_mem_type         | A vx_memory_type_e enumeration that specifies the memory type of the memory referenced by the user_addr. |

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>conv is not a valid vx_convolution reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>
3.70 Object: Distribution

3.70.1 Detailed Description

Defines the Distribution Object Interface.

Typedefs

- typedef struct _vx_distribution *vx_distribution

The Distribution object. This has a user-defined number of bins over a user-defined range (within a uint32_t range).

Enumerations

- enum vx_distribution_attribute_e {
  VX_DISTRIBUTION_DIMENSIONS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DISTRIBUTION << 8)) + 0x0,
  VX_DISTRIBUTION_OFFSET = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DISTRIBUTION << 8)) + 0x1,
  VX_DISTRIBUTION_RANGE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DISTRIBUTION << 8)) + 0x2,
  VX_DISTRIBUTION_BINS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DISTRIBUTION << 8)) + 0x3,
  VX_DISTRIBUTION_WINDOW = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DISTRIBUTION << 8)) + 0x4,
  VX_DISTRIBUTION_SIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DISTRIBUTION << 8)) + 0x5 }

The distribution attribute list.

Functions

- vx_status VX_API_CALL vxCopyDistribution (vx_distribution distribution, void *user_ptr, vx_enum usage, vx_enum user_mem_type)
  Allows the application to copy from/into a distribution object.

- vx_distribution VX_API_CALL vxCreateDistribution (vx_context context, vx_size numBins, vx_int32 offset, vx_uint32 range)
  Creates a reference to a 1D Distribution of a consecutive interval [offset, offset + range - 1] defined by a start offset and valid range, divided equally into numBins parts.

- vx_distribution VX_API_CALL vxCreateVirtualDistribution (vx_graph graph, vx_size numBins, vx_int32 offset, vx_uint32 range)
  Creates an opaque reference to a 1D Distribution object without direct user access.

- vx_status VX_API_CALL vxMapDistribution (vx_distribution distribution, vx_map_id *map_id, void **ptr, vx_enum usage, vx_enum mem_type, vx_bitfield flags)
  Allows the application to get direct access to distribution object.

- vx_status VX_API_CALL vxQueryDistribution (vx_distribution distribution, vx_enum attribute, void *ptr, vx_size size)
  Queries a Distribution object.

- vx_status VX_API_CALL vxReleaseDistribution (vx_distribution *distribution)
  Releases a reference to a distribution object. The object may not be garbage collected until its total reference count is zero.

- vx_status VX_API_CALL vxUnmapDistribution (vx_distribution distribution, vx_map_id map_id)
  Unmap and commit potential changes to distribution object that was previously mapped. Unmapping a distribution invalidates the memory location from which the distribution data could be accessed by the application. Accessing this memory location after the unmap function completes has an undefined behavior.

3.70.2 Enumeration Type Documentation

enum vx_distribution_attribute_e

The distribution attribute list.
### Enumerator

**VX_DISTRIBUTION_DIMENSIONS** Indicates the number of dimensions in the distribution. Read-only. Use a `vx_size` parameter.

**VX_DISTRIBUTION_OFFSET** Indicates the start of the values to use (inclusive). Read-only. Use a `vx_int32` parameter.

**VX_DISTRIBUTION_RANGE** Indicates the total number of the consecutive values of the distribution interval.

**VX_DISTRIBUTION_BINS** Indicates the number of bins. Read-only. Use a `vx_size` parameter.

**VX_DISTRIBUTION_WINDOW** Indicates the width of a bin. Equal to the range divided by the number of bins. If the range is not a multiple of the number of bins, it is not valid. Read-only. Use a `vx_uint32` parameter.

**VX_DISTRIBUTION_SIZE** Indicates the total size of the distribution in bytes. Read-only. Use a `vx_size` parameter.

Definition at line 1034 of file `vx_types.h`.

### 3.70.3 Function Documentation

#### `vx_distribution VX_API_CALL vxCreateDistribution ( vx_context context, vx_size numBins, vx_int32 offset, vx_uint32 range )`

Creates a reference to a 1D Distribution of a consecutive interval `[offset, offset + range - 1]` defined by a start offset and valid range, divided equally into numBins parts.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>numBins</td>
<td>The number of bins in the distribution.</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>The start offset into the range value that marks the beginning of the 1D Distribution.</td>
</tr>
<tr>
<td>in</td>
<td>range</td>
<td>The total number of the consecutive values of the distribution interval.</td>
</tr>
</tbody>
</table>

**Returns**

A distribution reference `vx_distribution`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

#### `vx_distribution VX_API_CALL vxCreateVirtualDistribution ( vx_graph graph, vx_size numBins, vx_int32 offset, vx_uint32 range )`

Creates an opaque reference to a 1D Distribution object without direct user access.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the parent graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>numBins</td>
<td>The number of bins in the distribution.</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>The start offset into the range value that marks the beginning of the 1D Distribution.</td>
</tr>
<tr>
<td>in</td>
<td>range</td>
<td>The total number of the consecutive values of the distribution interval.</td>
</tr>
</tbody>
</table>

**See also**

`vxCreateDistribution`

**Returns**

A distribution reference `vx_distribution`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.
vx_status VX_API_CALL vxReleaseDistribution ( vx_distribution * distribution )

Releases a reference to a distribution object. The object may not be garbage collected until its total reference count is zero.

Parameters

| in  | distribution | The reference to the distribution to release. |

Postcondition

After returning from this function the reference is zeroed.

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>distribution is not a valid vx_distribution reference.</td>
</tr>
</tbody>
</table>

vx_status VX_API_CALL vxQueryDistribution ( vx_distribution distribution, vx_enum attribute, void * ptr, vx_size size )

Queries a Distribution object.

Parameters

| in  | distribution | The reference to the distribution to query. |
| in  | attribute    | The attribute to query. Use a vx_distribution_attribute_e enumeration. |
| out | ptr          | The location at which to store the resulting value. |
| in  | size         | The size in bytes of the container to which ptr points. |

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>distribution is not a valid vx_distribution reference.</td>
</tr>
</tbody>
</table>

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

Allows the application to copy from/into a distribution object.

Parameters

| in  | distribution | The reference to the distribution object that is the source or the destination of the copy. |
### vxMapDistribution

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>user_ptr</td>
<td>The address of the memory location where to store the requested data if the copy was requested in read mode, or from where to get the data to store into the distribution object if the copy was requested in write mode. In the user memory, the distribution is represented as a <code>vx_uint32</code> array with a number of elements equal to the value returned via <strong>VX_DISTRIBUTION_BINS</strong>. The accessible memory must be large enough to contain this <code>vx_uint32</code> array: accessible memory in bytes (&gt;) (=) <code>sizeof(vx_uint32)</code> * num_bins.</td>
</tr>
</tbody>
</table>
| in   | usage | This declares the effect of the copy with regard to the distribution object using the **vx_accessor_e** enumeration. Only **VX_READ_ONLY** and **VX_WRITE_ONLY** are supported:  
  - **VX_READ_ONLY** means that data are copied from the distribution object into the user memory.  
  - **VX_WRITE_ONLY** means that data are copied into the distribution object from the user memory. |
| in   | user_mem_type | A **vx_memory_type_e** enumeration that specifies the memory type of the memory referenced by the user_addr. |

**Returns**

A **vx_status_e** enumeration.

**Return values**

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>distribution is not a valid <strong>vx_distribution</strong> reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

**vx_status VX_API_CALL vxMapDistribution ( vx_distribution distribution, vx_map_id * map_id, void ** ptr, vx_enum usage, vx_enum mem_type, vx_bitfield flags )**

Allows the application to get direct access to distribution object.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>distribution</td>
<td>The reference to the distribution object to map.</td>
</tr>
</tbody>
</table>
| out  | map_id | The address of a **vx_map_id** variable where the function returns a map identifier.  
  - (`*map_id`) must eventually be provided as the map_id parameter of a call to **vxUnmapDistribution**. |
| out  | ptr | The address of a pointer that the function sets to the address where the requested data can be accessed. In the mapped memory area, data are structured as a `vx_uint32` array with a number of elements equal to the value returned via **VX_DISTRIBUTION_BINS**. Each element of this array corresponds to a bin of the distribution, with a range-major ordering. Accessing the memory out of the bound of this array is forbidden and has an undefined behavior. The returned (`*ptr`) address is only valid between the call to the function and the corresponding call to **vxUnmapDistribution**. |
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usage</th>
<th>This declares the access mode for the distribution, using the \texttt{vx_accessor_e} enumeration.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\texttt{VX_READ_ONLY}: after the function call, the content of the memory location pointed by ((\ast ptr)) contains the distribution data. Writing into this memory location is forbidden and its behavior is undefined.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\texttt{VX_READ_AND_WRITE}: after the function call, the content of the memory location pointed by ((\ast ptr)) contains the distribution data; writing into this memory is allowed only for the location of bins and will result in a modification of the affected bins in the distribution object once the distribution is unmapped.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\texttt{VX_WRITE_ONLY}: after the function call, the memory location pointed by ((\ast ptr)) contains undefined data; writing each bin of distribution is required prior to unmapping. Bins not written by the application before unmap will become undefined after unmap, even if they were well defined before map.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>in</th>
<th>mem_type</th>
<th>A \texttt{vx_memory_type_e} enumeration that specifies the type of the memory where the distribution is requested to be mapped.</th>
</tr>
</thead>
</table>

| in  | flags | An integer that allows passing options to the map operation. Use 0 for this option. |

Returns

A \texttt{vx_status_e} enumeration.

Return values

<table>
<thead>
<tr>
<th>\texttt{VX_SUCCESS}</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{VX_ERROR_INVALID_REFERENCE}</td>
<td>Distribution is not a valid \texttt{vx_distribution} reference.</td>
</tr>
<tr>
<td>\texttt{VX_ERROR_INVALID_PARAMETERS}</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

Postcondition

\texttt{vxUnmapDistribution} with same \((\ast map\_id)\) value.

\texttt{vx_status VX_API_CALL vxUnmapDistribution ( vx_distribution distribution, vx_map_id map\_id )}

Unmap and commit potential changes to distribution object that was previously mapped. Unmapping a distribution invalidates the memory location from which the distribution data could be accessed by the application. Accessing this memory location after the unmap function completes has an undefined behavior.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>distribution</th>
<th>The reference to the distribution object to unmap.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>map_id</td>
<td>The unique map identifier that was returned when calling \texttt{vxMapDistribution}.</td>
</tr>
</tbody>
</table>

Returns

A \texttt{vx_status_e} enumeration.

Return values

<table>
<thead>
<tr>
<th>\texttt{VX_SUCCESS}</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{VX_ERROR_INVALID_REFERENCE}</td>
<td>Distribution is not a valid \texttt{vx_distribution} reference.</td>
</tr>
<tr>
<td>\texttt{VX_ERROR_INVALID_PARAMETERS}</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>
Precondition

\texttt{vxMapDistribution} returning the same map\_id value
3.71 Object: Image

3.71.1 Detailed Description
Defines the Image Object interface.

Data Structures
- struct vx_imagepatch_addressing_t
  The addressing image patch structure is used by the Host only to address pixels in an image patch. The fields of the structure are defined as: More...
- union vx_pixel_value_t
  Union that describes the value of a pixel for any image format. Use the field corresponding to the image format. More...

Macros
- #define VX_IMAGEPATCH_ADDR_INIT {0u, 0u, 0, 0, 0u, 0u, 0u, 0u}
  Use to initialize a vx_imagepatch_addressing_t structure on the stack.

Typedefs
- typedef struct _vx_image ∗vx_image
  An opaque reference to an image.
- typedef uintptr_t vx_map_id
  Holds the address of a variable where the map/unmap functions return a map identifier.

Enumerations
- enum vx_channel_range_e {
  VX_CHANNEL_RANGE_FULL = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_COLOR_RANGE << 12)) + 0x0,
  VX_CHANNEL_RANGE_RESTRICTED = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_COLOR_RANGE << 12)) + 0x1
  }
  The image channel range list used by the VX_IMAGE_RANGE attribute of a vx_image.
- enum vx_color_space_e {
  VX_COLOR_SPACE_NONE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_COLOR_SPACE << 12)) + 0x0,
  VX_COLOR_SPACE_BT601_525 = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_COLOR_SPACE << 12)) + 0x1,
  VX_COLOR_SPACE_BT601_625 = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_COLOR_SPACE << 12)) + 0x2,
  VX_COLOR_SPACE_BT709 = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_COLOR_SPACE << 12)) + 0x3,
  VX_COLOR_SPACE_DEFAULT = VX_COLOR_SPACE_BT709
  }
  The image color space list used by the VX_IMAGE_SPACE attribute of a vx_image.
- enum vx_image_attribute_e {
  VX_IMAGE_WIDTH = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_IMAGE << 8)) + 0x0,
  VX_IMAGE_HEIGHT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_IMAGE << 8)) + 0x1,
  VX_IMAGE_FORMAT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_IMAGE << 8)) + 0x2,
  VX_IMAGE_PLANES = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_IMAGE << 8)) + 0x3,
  VX_IMAGE_SPACE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_IMAGE << 8)) + 0x4,
  VX_IMAGE_RANGE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_IMAGE << 8)) + 0x5,
  VX_IMAGE_MEMORY_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_IMAGE << 8)) + 0x7,
  VX_IMAGE_IS_UNIFORM = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_IMAGE << 8)) + 0x8,
  VX_IMAGE_UNIFORM_VALUE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_IMAGE << 8)) + 0x9
  }
  The image attributes list.
• enum vx_map_flag_e { VX_NOGAP_X = 1 }

The Map/Unmap operation enumeration.

Functions

• vx_status VX_API_CALL vxCopyImagePatch (vx_image image, const vx_rectangle_t *image_rect, vx_uint32 image_plane_index, const vx_imagepatch_addressing_t *user_addr, void **user_ptr, vx_enum usage, vx_enum user_mem_type)

Allows the application to copy a rectangular patch from/into an image object plane.

• vx_image VX_API_CALL vxCreateImage (vx_context context, vx_uint32 width, vx_uint32 height, vx_df_image color)

Creates an opaque reference to an image buffer.

• vx_image VX_API_CALL vxCreateImageFromChannel (vx_image img, vx_enum channel)

Create a sub-image from a single plane channel of another image.

• vx_image VX_API_CALL vxCreateImageFromHandle (vx_context context, vx_df_image color, const vx_imagepatch_addressing_t *addr, void *const ptrs[], vx_enum memory_type)

Creates a reference to an image object that was externally allocated.

• vx_image VX_API_CALL vxCreateImageFromROI (vx_image img, const vx_rectangle_t *rect)

Creates an image from another image given a rectangle. This second reference refers to the data in the original image. Updates to this image updates the parent image. The rectangle must be defined within the pixel space of the parent image.

• vx_image VX_API_CALL vxCreateUniformImage (vx_context context, vx_uint32 width, vx_uint32 height, vx_df_image color, const vx_pixel_value_t *value)

Creates a reference to an image object that has a singular, uniform value in all pixels. The uniform image created is read-only.

• vx_image VX_API_CALL vxCreateVirtualImage (vx_graph graph, vx_uint32 width, vx_uint32 height, vx_df_image color)

Creates an opaque reference to an image buffer with no direct user access. This function allows setting the image width, height, or format.

• void *VX_API_CALL vxFormatImagePatchAddress1d (void *ptr, vx_uint32 index, const vx_imagepatch_addressing_t *addr)

Accesses a specific indexed pixel in an image patch.

• void *VX_API_CALL vxFormatImagePatchAddress2d (void *ptr, vx_uint32 x, vx_uint32 y, const vx_imagepatch_addressing_t *addr)

Accesses a specific pixel at a 2d coordinate in an image patch.

• vx_status VX_API_CALL vxGetValidRegionImage (vx_image image, vx_rectangle_t *rect)

Retrieves the valid region of the image as a rectangle.

• vx_status VX_API_CALL vxMapImagePatch (vx_image image, const vx_rectangle_t *rect, vx_uint32 plane_index, vx_map_id *map_id, vx_imagepatch_addressing_t *addr, void **ptr, vx_enum usage, vx_enum mem_type, vx_uint32 flags)

Allows the application to get direct access to a rectangular patch of an image object plane.

• vx_status VX_API_CALL vxQueryImage (vx_image image, vx_enum attribute, void *ptr, vx_size size)

Retrieves various attributes of an image.

• vx_status VX_API_CALL vxReleaseImage (vx_image *image)

Releases a reference to an image object. The object may not be garbage collected until its total reference count is zero.

• vx_status VX_API_CALL vxSetImageAttribute (vx_image image, vx_enum attribute, const void *ptr, vx_size size)

Allows setting attributes on the image.

• vx_status VX_API_CALL vxSetImagePixelValues (vx_image image, const vx_pixel_value_t *pixel_value)

Initialize an image with the given pixel value.

• vx_status VX_API_CALL vxSetImageValidRectangle (vx_image image, const vx_rectangle_t *rect)

Sets the valid rectangle for an image according to a supplied rectangle.
CHAPTER 3. MODULE DOCUMENTATION

230

• `vx_status VX_API_CALL vxSwapImageHandle (vx_image image, void *new_ptrs[], void *prev_ptrs[],
  vx_size num_planes)`

  Swaps the image handle of an image previously created from handle.

• `vx_status VX_API_CALL vxUnmapImagePatch (vx_image image, vx_map_id map_id)`

  Unmap and commit potential changes to a image object patch that were previously mapped. Unmapping an image
  patch invalidates the memory location from which the patch could be accessed by the application. Accessing this
  memory location after the unmap function completes has an undefined behavior.

3.71.2 Data Structure Documentation

`struct vx_imagepatch_addressing_t`

The addressing image patch structure is used by the Host only to address pixels in an image patch. The fields of
the structure are defined as:

- `dim` - The dimensions of the image in logical pixel units in the x & y direction.
- `stride` - The physical byte distance from a logical pixel to the next logically adjacent pixel in the positive x or y
direction.
- `scale` - The relationship of scaling from the primary plane (typically the zero indexed plane) to this plane. An
  integer down-scaling factor of $f$ shall be set to a value equal to $scale = \frac{unity}{f}$ and an integer up-scaling factor
  of $f$ shall be set to a value of $scale = unity \times f$. $unity$ is defined as $VX\_SCALE\_UNITY$.
- `step` - The step is the number of logical pixel units to skip to arrive at the next physically unique pixel. For
  example, on a plane that is half-scaled in a dimension, the step in that dimension is 2 to indicate that every
  other pixel in that dimension is an alias. This is useful in situations where iteration over unique pixels is
  required, such as in serializing or de-serializing the image patch information.

See also

`vxMapImagePatch`

Definition at line 1519 of file `vx_types.h`.

Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vx_uint32</code></td>
<td><code>dim_x</code></td>
<td>Width of patch in X dimension in pixels.</td>
</tr>
<tr>
<td><code>vx_uint32</code></td>
<td><code>dim_y</code></td>
<td>Height of patch in Y dimension in pixels.</td>
</tr>
<tr>
<td><code>vx_int32</code></td>
<td><code>stride_x</code></td>
<td>Stride in X dimension in bytes.</td>
</tr>
<tr>
<td><code>vx_int32</code></td>
<td><code>stride_y</code></td>
<td>Stride in Y dimension in bytes.</td>
</tr>
<tr>
<td><code>vx_uint32</code></td>
<td><code>scale_x</code></td>
<td>Scale of X dimension. For sub-sampled planes this is the scaling factor of the dimension of the plane in relation to the zero plane. Use <code>VX\_SCALE\_UNITY</code> in the numerator.</td>
</tr>
<tr>
<td><code>vx_uint32</code></td>
<td><code>scale_y</code></td>
<td>Scale of Y dimension. For sub-sampled planes this is the scaling factor of the dimension of the plane in relation to the zero plane. Use <code>VX\_SCALE\_UNITY</code> in the numerator.</td>
</tr>
<tr>
<td><code>vx_uint32</code></td>
<td><code>step_x</code></td>
<td>Step of X dimension in pixels.</td>
</tr>
<tr>
<td><code>vx_uint32</code></td>
<td><code>step_y</code></td>
<td>Step of Y dimension in pixels.</td>
</tr>
</tbody>
</table>

union `vx_pixel_value_t`

Union that describes the value of a pixel for any image format. Use the field corresponding to the image format.

Definition at line 1684 of file `vx_types.h`. 
### Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_uint8</td>
<td>YUV[3] All YUV formats in the Y,U,V order</td>
</tr>
<tr>
<td>vx_uint8</td>
<td>U8 VX_DF_IMAGE_U8</td>
</tr>
<tr>
<td>vx_uint16</td>
<td>U16 VX_DF_IMAGE_U16</td>
</tr>
<tr>
<td>vx_int16</td>
<td>S16 VX_DF_IMAGE_S16</td>
</tr>
<tr>
<td>vx_uint32</td>
<td>U32 VX_DF_IMAGE_U32</td>
</tr>
<tr>
<td>vx_int32</td>
<td>S32 VX_DF_IMAGE_S32</td>
</tr>
<tr>
<td>vx_uint8</td>
<td>reserved[16]</td>
</tr>
</tbody>
</table>

### 3.71.3 Typedef Documentation

typedef struct _vx_image* vx_image

An opaque reference to an image.

See also

vxCreateImage

Definition at line 179 of file vx_types.h.

### 3.71.4 Enumeration Type Documentation

enum vx_image_attribute_e

The image attributes list.

**Enumerator**

- **VX_IMAGE_WIDTH** Queries an image for its width. Read-only. Use a vx_uint32 parameter.
- **VX_IMAGE_HEIGHT** Queries an image for its height. Read-only. Use a vx_uint32 parameter.
- **VX_IMAGE_FORMAT** Queries an image for its format. Read-only. Use a vx_df_image parameter.
- **VX_IMAGE_PLANES** Queries an image for its number of planes. Read-only. Use a vx_size parameter.
- **VX_IMAGE_SPACE** Queries an image for its color space (see vx_color_space_e). Read-write. Use a vx_enum parameter.
- **VX_IMAGE_RANGE** Queries an image for its channel range (see vx_channel_range_e). Read-only. Use a vx_enum parameter.
- **VX_IMAGE_MEMORY_TYPE** Queries memory type if created using vxCreateImageFromHandle. If vx_image was not created using vxCreateImageFromHandle, VX_MEMORY_TYPE_NONE is returned. Use a vx_memory_type_e parameter.
- **VX_IMAGE_IS_UNIFORM** Queries if an image is uniform. Read-only. Use a vx_bool parameter.
- **VX_IMAGE_UNIFORM_VALUE** Queries the image uniform value if any. Read-only. Use a vx_pixel_value_t parameter.

Definition at line 946 of file vx_types.h.

enum vx_color_space_e

The image color space list used by the VX_IMAGE_SPACE attribute of a vx_image.

**Enumerator**

- **VX_COLOR_SPACE_NONE** Use to indicate that no color space is used.
**VX_COLOR_SPACE_BT601_525** Use to indicate that the BT.601 coefficients and SMPTE C primaries are used for conversions.

**VX_COLOR_SPACE_BT601_625** Use to indicate that the BT.601 coefficients and BTU primaries are used for conversions.

**VX_COLOR_SPACE_BT709** Use to indicate that the BT.709 coefficients are used for conversions.

**VX_COLOR_SPACE_DEFAULT** All images in VX are by default BT.709.

Definition at line 1295 of file vx_types.h.

**enum vx_channel_range_e**

The image channel range list used by the **VX_IMAGE_RANGE** attribute of a **vx_image**.

**Enumerator**

- **VX_CHANNEL_RANGE_FULL** Full range of the unit of the channel.
- **VX_CHANNEL_RANGE_RESTRICTED** Restricted range of the unit of the channel based on the space given.

Definition at line 1312 of file vx_types.h.

**enum vx_map_flag_e**

The Map/Unmap operation enumeration.

**Enumerator**

- **VX_NOGAP_X** No Gap.

Definition at line 1845 of file vx_types.h.

### 3.71.5 Function Documentation

**vx_image VX_API_CALL vxCreateImage ( vx_context context, vx_uint32 width, vx_uint32 height, vx_df_image color )**

Creates an opaque reference to an image buffer.

Not guaranteed to exist until the **vx_graph** containing it has been verified.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in context</strong></td>
<td>The reference to the implementation context.</td>
</tr>
<tr>
<td><strong>in width</strong></td>
<td>The image width in pixels. The image in the formats of <strong>VX_DF_IMAGE_NV12</strong>, <strong>VX_DF_IMAGE_NV21</strong>, <strong>VX_DF_IMAGE_IYUV</strong>, <strong>VX_DF_IMAGE_UYVY</strong>, <strong>VX_DF_IMAGE_YUYV</strong> must have even width.</td>
</tr>
<tr>
<td><strong>in height</strong></td>
<td>The image height in pixels. The image in the formats of <strong>VX_DF_IMAGE_NV12</strong>, <strong>VX_DF_IMAGE_NV21</strong>, <strong>VX_DF_IMAGE_IYUV</strong> must have even height.</td>
</tr>
<tr>
<td><strong>in color</strong></td>
<td>The <strong>VX_DF_IMAGE</strong> (<strong>vx_df_image_e</strong>) code that represents the format of the image and the color space.</td>
</tr>
</tbody>
</table>

**Returns**

An image reference **vx_image**. Any possible errors preventing a successful creation should be checked using **vxGetStatus**.

**See also**

**vxMapImagePatch** to obtain direct memory access to the image data.
**vx_image VX_API_CALL vxCreateImageFromROI ( vx_image img, const vx_rectangle_t *rect )**

Creates an image from another image given a rectangle. This second reference refers to the data in the original image. Updates to this image updates the parent image. The rectangle must be defined within the pixel space of the parent image.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>img</th>
<th>The reference to the parent image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rect</td>
<td>The region of interest rectangle. Must contain points within the parent image pixel space.</td>
</tr>
</tbody>
</table>

**Returns**

An image reference `vx_image` to the sub-image. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx_image VX_API_CALL vxCreateUniformImage ( vx_context context, vx_uint32 width, vx_uint32 height, vx_df_image color, const vx_pixel_value_t *value )**

Creates a reference to an image object that has a singular, uniform value in all pixels. The uniform image created is read-only.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the implementation context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>width</td>
<td>The image width in pixels. The image in the formats of <code>VX_DF_IMAGE_NV12</code>, <code>VX_DF_IMAGE_NV21</code>, <code>VX_DF_IMAGE_IYUV</code>, <code>VX_DF_IMAGE_UYVY</code>, <code>VX_DF_IMAGE_YUYV</code> must have even width.</td>
</tr>
<tr>
<td>in</td>
<td>height</td>
<td>The image height in pixels. The image in the formats of <code>VX_DF_IMAGE_NV12</code>, <code>VX_DF_IMAGE_NV21</code>, <code>VX_DF_IMAGE_IYUV</code> must have even height.</td>
</tr>
<tr>
<td>in</td>
<td>color</td>
<td>The <code>VX_DF_IMAGE (vx_df_image_e)</code> code that represents the format of the image and the color space.</td>
</tr>
<tr>
<td>in</td>
<td>value</td>
<td>The pointer to the pixel value to which to set all pixels. See <code>vx_pixel_value_t</code>.</td>
</tr>
</tbody>
</table>

**Returns**

An image reference `vx_image`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**See also**

`vxMapImagePatch` to obtain direct memory access to the image data.

**Note**

`vxMapImagePatch` and `vxUnmapImagePatch` may be called with a uniform image reference.

**vx_image VX_API_CALL vxCreateVirtualImage ( vx_graph graph, vx_uint32 width, vx_uint32 height, vx_df_image color )**

Creates an opaque reference to an image buffer with no direct user access. This function allows setting the image width, height, or format.

Virtual data objects allow users to connect various nodes within a graph via data references without access to that data, but they also permit the implementation to take maximum advantage of possible optimizations. Use this API to create a data reference to link two or more nodes together when the intermediate data are not required to be accessed by outside entities. This API in particular allows the user to define the image format of the data without requiring the exact dimensions. Virtual objects are scoped within the graph they are declared a part of, and can’t be shared outside of this scope. All of the following constructions of virtual images are valid.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the parent graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>width</td>
<td>The width of the image in pixels. A value of zero informs the interface that the value is unspecified. The image in the formats of VX_DF_IMAGE_NV12, VX_DF_IMAGE_NV21, VX_DF_IMAGE_IYUV, VX_DF_IMAGE_UYVY, VX_DF_IMAGE_YUYV must have even width.</td>
</tr>
<tr>
<td>in</td>
<td>height</td>
<td>The height of the image in pixels. A value of zero informs the interface that the value is unspecified. The image in the formats of VX_DF_IMAGE_NV12, VX_DF_IMAGE_NV21, VX_DF_IMAGE_IYUV must have even height.</td>
</tr>
<tr>
<td>in</td>
<td>color</td>
<td>The VX_DF_IMAGE (vx_df_image_e) code that represents the format of the image and the color space. A value of VX_DF_IMAGE_VIRT informs the interface that the format is unspecified.</td>
</tr>
</tbody>
</table>

Returns

An image reference vx_image. Any possible errors preventing a successful creation should be checked using vxGetStatus.

Note

Passing this reference to vxMapImagePatch will return an error.

`vx_image VX_API_CALL vxCreateImageFromHandle ( vx_context context, vx_df_image color, const vx_imagepatch_addressing_t addrs[], void *const ptrs[], vx_enum memory_type )`

Creates a reference to an image object that was externally allocated.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the implementation context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>color</td>
<td>See the vx_df_image_e codes. This mandates the number of planes needed to be valid in the addrs and ptrs arrays based on the format given.</td>
</tr>
<tr>
<td>in</td>
<td>addrs[]</td>
<td>The array of image patch addressing structures that define the dimension and stride of the array of pointers. See note below.</td>
</tr>
<tr>
<td>in</td>
<td>ptrs[]</td>
<td>The array of platform-defined references to each plane. See note below.</td>
</tr>
<tr>
<td>in</td>
<td>memory_type</td>
<td>vx_memory_type_e. When giving VX_MEMORY_TYPE_HOST the ptrs array is assumed to be HOST accessible pointers to memory.</td>
</tr>
</tbody>
</table>

Returns

An image reference vx_image. Any possible errors preventing a successful creation should be checked using vxGetStatus.

Note

The user must call vxMapImagePatch prior to accessing the pixels of an image, even if the image was created via vxCreateImageFromHandle. Reads or writes to memory referenced by ptrs[] after calling vxCreateImageFromHandle without first calling vxMapImagePatch will result in undefined behavior.
The property of addr[] and ptrs[] arrays is kept by the caller (it means that the implementation will make an internal copy of the provided information. addr and ptrs can then simply be application's local variables). Only dim_x, dim_y, stride_x and stride_y fields of the vx_imagepatch_addressing_t need to be provided by the application. Other fields (step_x, step_y, scale_x & scale_y) are ignored by this function. The layout of the imported memory must follow a row-major order. In other words, stride_x should be sufficiently large so that there is no overlap between data elements corresponding to different pixels, and stride_y >= stride_x * dim_x.

In order to release the image back to the application we should use vxSwapImageHandle.

Import type of the created image is available via the image attribute vx_image_attribute_e parameter.

vx_status VX_API_CALL vxSwapImageHandle ( vx_image image, void *const new_ptrs[], void *prev_ptrs[], vx_size num_planes )

Swaps the image handle of an image previously created from handle.

This function sets the new image handle (i.e. pointer to all image planes) and returns the previous one.

Once this function call has completed, the application gets back the ownership of the memory referenced by the previous handle. This memory contains up-to-date pixel data, and the application can safely reuse or release it.

The memory referenced by the new handle must have been allocated consistently with the image properties since the import type, memory layout and dimensions are unchanged (see addrs, color, and memory_type in vxCreateImageFromHandle).

All images created from ROI or channel with this image as parent or ancestor will automatically use the memory referenced by the new handle.

The behavior of vxSwapImageHandle when called from a user node is undefined.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>image</th>
<th>The reference to an image created from handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>new_ptrs[]</td>
<td>pointer to a caller owned array that contains the new image handle (image plane pointers)</td>
</tr>
<tr>
<td></td>
<td>new_ptrs is non NULL. new_ptrs[i] must be non NULL for each i such as 0 &lt; i &lt; nbPlanes, otherwise, this is an error. The address of the storage memory for image plane i is set to new_ptrs[i]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>new_ptrs is NULL: the previous image storage memory is reclaimed by the caller, while no new handle is provided.</td>
<td></td>
</tr>
<tr>
<td>out</td>
<td>prev_ptrs[]</td>
<td>pointer to a caller owned array in which the application returns the previous image handle</td>
</tr>
<tr>
<td></td>
<td>prev_ptrs is non NULL. prev_ptrs must have at least as many elements as the number of image planes. For each i such as 0 &lt; i &lt; nbPlanes , prev_ptrs[i] is set to the address of the previous storage memory for plane i.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>prev_ptrs NULL: the previous handle is not returned.</td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>num_planes</td>
<td>Number of planes in the image. This must be set equal to the number of planes of the input image. The number of elements in new_ptrs and prev_ptrs arrays must be equal to or greater than num_planes. If either array has more than num_planes elements, the extra elements are ignored. If either array is smaller than num_planes, the results are undefined.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

| VX_SUCCESS | No errors. |
### Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>image is not a valid vx_image reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>The image was not created from handle or the content of new_ptrs is not valid.</td>
</tr>
<tr>
<td>VX_FAILURE</td>
<td>The image was already being accessed.</td>
</tr>
</tbody>
</table>

#### vx_status VX_API_CALL vxQueryImage ( vx_image image, vx_enum attribute, void *ptr, vx_size size )

Retrieves various attributes of an image.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>The reference to the image to query.</td>
</tr>
<tr>
<td>attribute</td>
<td>The attribute to query. Use a vx_image_attribute_e.</td>
</tr>
<tr>
<td>ptr</td>
<td>The location at which to store the resulting value.</td>
</tr>
<tr>
<td>size</td>
<td>The size in bytes of the container to which ptr points.</td>
</tr>
</tbody>
</table>

**Returns**

A vx_status_e enumeration.

**Return values**

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>image is not a valid vx_image reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>If any of the other parameters are incorrect.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUPPORTED</td>
<td>If the attribute is not supported on this implementation.</td>
</tr>
</tbody>
</table>

#### vx_status VX_API_CALL vxSetImageAttribute ( vx_image image, vx_enum attribute, const void *ptr, vx_size size )

Allows setting attributes on the image.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>The reference to the image on which to set the attribute.</td>
</tr>
<tr>
<td>attribute</td>
<td>The attribute to set. Use a vx_image_attribute_e enumeration.</td>
</tr>
<tr>
<td>ptr</td>
<td>The pointer to the location from which to read the value.</td>
</tr>
<tr>
<td>size</td>
<td>The size in bytes of the object pointed to by ptr.</td>
</tr>
</tbody>
</table>

**Returns**

A vx_status_e enumeration.

**Return values**

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>image is not a valid vx_image reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>If any of the other parameters are incorrect.</td>
</tr>
</tbody>
</table>
vx_status VX_API_CALL vxSetImagePixelValues ( vx_image image, const vx_pixel_value_t * pixel_value )

Initialize an image with the given pixel value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>The reference to the image to initialize.</td>
</tr>
<tr>
<td>pixel_value</td>
<td>The pointer to the constant pixel value to initialize all image pixels. See vx_pixel_value_t.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>If the image is a uniform image, a virtual image, or not a vx_image.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>If any of the other parameters are incorrect.</td>
</tr>
</tbody>
</table>

Note

All pixels of the entire image are initialized to the indicated pixel value, independently from the valid region. The valid region of the image is unaffected by this function. The image remains mutable after the call to this function, so its pixels and mutable attributes may be changed by subsequent functions.

vx_status VX_API_CALL vxReleaseImage ( vx_image * image )

Releases a reference to an image object. The object may not be garbage collected until its total reference count is zero.

An implementation may defer the actual object destruction after its total reference count is zero (potentially until context destruction). Thus, releasing an image created from handle (see vxCreateImageFromHandle) and all others objects that may reference it (nodes, ROI, or channel for instance) are not sufficient to get back the ownership of the memory referenced by the current image handle. The only way for this is to call vxSwapImageHandle before releasing the image.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>The pointer to the image to release.</td>
</tr>
</tbody>
</table>

Postcondition

After returning from this function the reference is zeroed.

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Image is not a valid vx_image reference.</td>
</tr>
</tbody>
</table>
void * VX_API_CALL vxFormatImagePatchAddress1d ( void * ptr, vx_uint32 index, const vx_imagepatch_addressing_t * addr )

Accesses a specific indexed pixel in an image patch.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ptr</th>
<th>The base pointer of the patch as returned from vxMapImagePatch.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>index</td>
<td>The 0 based index of the pixel count in the patch. Indexes increase horizontally by 1 then wrap around to the next row.</td>
</tr>
<tr>
<td>in</td>
<td>addr</td>
<td>The pointer to the addressing mode information returned from vxMapImagePatch.</td>
</tr>
</tbody>
</table>

Returns

void * Returns the pointer to the specified pixel.

Precondition

vxMapImagePatch

void * VX_API_CALL vxFormatImagePatchAddress2d ( void * ptr, vx_uint32 x, vx_uint32 y, const vx_imagepatch_addressing_t * addr )

Accesses a specific pixel at a 2d coordinate in an image patch.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ptr</th>
<th>The base pointer of the patch as returned from vxMapImagePatch.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x</td>
<td>The x dimension within the patch.</td>
</tr>
<tr>
<td>in</td>
<td>y</td>
<td>The y dimension within the patch.</td>
</tr>
<tr>
<td>in</td>
<td>addr</td>
<td>The pointer to the addressing mode information returned from vxMapImagePatch.</td>
</tr>
</tbody>
</table>

Returns

void * Returns the pointer to the specified pixel.

Precondition

vxMapImagePatch

vx_status VX_API_CALL vxGetValidRegionImage ( vx_image image, vx_rectangle_t * rect )

Retrieves the valid region of the image as a rectangle.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>image</th>
<th>The image from which to retrieve the valid region.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>rect</td>
<td>The destination rectangle.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.
### Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Image is not a valid <code>vx_image</code> reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>Invalid rect.</td>
</tr>
</tbody>
</table>

Note

This rectangle can be passed directly to `vxMapImagePatch` to get the full valid region of the image.

```c
vx_status VX_API_CALL vxCopyImagePatch ( vx_image image, const vx_rectangle_t *image_rect, vx_uint32 image_plane_index, const vx_imagepatch_addressing_t *user_addr, void *user_ptr, vx_enum usage, vx_enum user_mem_type )
```

Allows the application to copy a rectangular patch from/into an image object plane.

**Parameters**

<table>
<thead>
<tr>
<th>in image</th>
<th>The reference to the image object that is the source or the destination of the copy.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in image_rect</td>
<td>The coordinates of the image patch. The patch must be within the bounds of the image. (start_x, start_y) gives the coordinates of the topleft pixel inside the patch, while (end_x, end_y) gives the coordinates of the bottomright element out of the patch. Must be 0 &lt; start &lt; end &lt;= number of pixels in the image dimension.</td>
</tr>
<tr>
<td>in image_plane_index</td>
<td>The plane index of the image object that is the source or the destination of the patch copy.</td>
</tr>
<tr>
<td>in user_addr</td>
<td>The address of a structure describing the layout of the user memory location pointed by user_ptr. In the structure, only dim_x, dim_y, stride_x and stride_y fields must be provided, other fields are ignored by the function. The layout of the user memory must follow a row major order: stride_x &gt;= pixel size in bytes, and stride_y &gt;= stride_x * dim_x.</td>
</tr>
<tr>
<td>in user_ptr</td>
<td>The address of the memory location where to store the requested data if the copy was requested in read mode, or from where to get the data to store into the image object if the copy was requested in write mode. The accessible memory must be large enough to contain the specified patch with the specified layout: accessible memory in bytes &gt;= (end_y - start_y) * stride_y.</td>
</tr>
</tbody>
</table>
| in usage | This declares the effect of the copy with regard to the image object using the `vx_accessor_e` enumeration. For uniform images, only VX_READ_ONLY is supported. For other images, Only VX_READ_ONLY and VX_WRITE_ONLY are supported:  
- VX_READ_ONLY means that data is copied from the image object into the application memory  
- VX_WRITE_ONLY means that data is copied into the image object from the application memory |
| in user_mem_type | A `vx_memory_type_e` enumeration that specifies the memory type of the memory referenced by the user_addr. |

**Returns**

A `vx_status_e` enumeration.

**Return values**

| VX_SUCCESS | No errors; any other value indicates failure. |
Return values

<table>
<thead>
<tr>
<th>VX_ERROR_OPTIMIZED_AWAY</th>
<th>This is a reference to a virtual image that cannot be accessed by the application.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>image is not a valid vx_image reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

Note

The application may ask for data outside the bounds of the valid region, but such data has an undefined value.

```c
vx_status VX_API_CALL vxMapImagePatch ( vx_image image, const vx_rectangle_t * rect, vx_uint32 plane_index, vx_map_id * map_id, vx_imagepatch_addressing_t * addr, void ** ptr, vx_enum usage, vx_enum mem_type, vx_uint32 flags )
```

Allows the application to get direct access to a rectangular patch of an image object plane.

Parameters

| in | image | The reference to the image object that contains the patch to map. |
| in | rect | The coordinates of image patch. The patch must be within the bounds of the image. (start_x, start_y) gives the coordinate of the topleft element inside the patch, while (end_x, end_y) give the coordinate of the bottomright element out of the patch. Must be 0 <= start < end. |
| in | plane_index | The plane index of the image object to be accessed. |
| out | map_id | The address of a vx_map_id variable where the function returns a map identifier. |
|     |     | • (∗map_id) must eventually be provided as the map_id parameter of a call to vxUnmapImagePatch. |
| out | addr | The address of a structure describing the memory layout of the image patch to access. The function fills the structure pointed by addr with the layout information that the application must consult to access the pixel data at address (∗ptr). The layout of the mapped memory follows a row-major order: stride_x>0, stride_y>0 and stride_y >= stride_x * dim_x. If the image object being accessed was created via vxCreateImageFromHandle, then the returned memory layout will be the identical to that of the addressing structure provided when vxCreateImageFromHandle was called. |
| out | ptr | The address of a pointer that the function sets to the address where the requested data can be accessed. This returned (∗ptr) address is only valid between the call to this function and the corresponding call to vxUnmapImagePatch. If image was created via vxCreateImageFromHandle then the returned address (∗ptr) will be the address of the patch in the original pixel buffer provided when image was created. |
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usage</th>
<th>This declares the access mode for the image patch, using the <code>vx_accessor_e</code> enumeration. For uniform images, only VX_READ_ONLY is supported.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• VX_READ_ONLY: after the function call, the content of the memory location pointed by <code>∗ptr</code> contains the image patch data. Writing into this memory location is forbidden and its behavior is undefined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• VX_READ_AND_WRITE: after the function call, the content of the memory location pointed by <code>∗ptr</code> contains the image patch data; writing into this memory is allowed only for the location of pixels only and will result in a modification of the written pixels in the image object once the patch is unmapped. Writing into a gap between pixels (when <code>addr-&gt;stride_x &gt; pixel size in bytes</code> or <code>addr-&gt;stride_y &gt; addr-&gt;stride_x*addr-&gt;dim_x</code>) is forbidden and its behavior is undefined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• VX_WRITE_ONLY: after the function call, the memory location pointed by <code>∗ptr</code> contains undefined data; writing each pixel of the patch is required prior to unmap. Pixels not written by the application before unmap will become undefined after unmap, even if they were well defined before map. Like for VX_READ_AND_WRITE, writing into a gap between pixels is forbidden and its behavior is undefined.</td>
</tr>
<tr>
<td></td>
<td>mem_type</td>
<td>A <code>vx_memory_type_e</code> enumeration that specifies the type of the memory where the image patch is requested to be mapped.</td>
</tr>
<tr>
<td></td>
<td>flags</td>
<td>An integer that allows passing options to the map operation. Use the <code>vx_map_flag_e</code> enumeration.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_OPTIMIZED_AWAY</td>
<td>This is a reference to a virtual image that cannot be accessed by the application.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td><code>image</code> is not a valid <code>vx_image</code> reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

Note

The user may ask for data outside the bounds of the valid region, but such data has an undefined value.

Postcondition

```
vxUnmapImagePatch ( vx_image image, vx_map_id map_id )
```

Unmap and commit potential changes to a image object patch that were previously mapped. Unmapping an image patch invalidates the memory location from which the patch could be accessed by the application. Accessing this memory location after the unmap function completes has an undefined behavior.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>image</th>
<th>The reference to the image object to unmap.</th>
</tr>
</thead>
</table>
| out  | map_id | The unique map identifier that was returned by `vxMapImagePatch`.


Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th><code>VX_SUCCESS</code></th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>VX_ERROR_INVALID_REFERENCE</code></td>
<td>Image is not a valid <code>vx_image</code> reference.</td>
</tr>
<tr>
<td><code>VX_ERROR_INVALID_PARAMETERS</code></td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

Precondition

`vxMapImagePatch` with same `map_id` value

**vx_image VX_API_CALL vxCreateImageFromChannel ( vx_image img, vx_enum channel )**

Create a sub-image from a single plane channel of another image.

The sub-image refers to the data in the original image. Updates to this image update the parent image and reversely.

The function supports only channels that occupy an entire plane of a multi-planar images, as listed below. Other cases are not supported. `VX_CHANNEL_Y` from YUV4, IYUV, NV12, NV21 `VX_CHANNEL_U` from YUV4, IYUV

Parameters

<table>
<thead>
<tr>
<th><strong>in</strong></th>
<th><code>img</code></th>
<th>The reference to the parent image.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><code>channel</code></td>
<td>The <code>vx_channel_e</code> channel to use.</td>
</tr>
</tbody>
</table>

Returns

An image reference `vx_image` to the sub-image. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx_status VX_API_CALL vxSetImageValidRectangle ( vx_image image, const vx_rectangle_t * rect )**

Sets the valid rectangle for an image according to a supplied rectangle.

Note

Setting or changing the valid region from within a user node by means other than the call-back, for example by calling `vxSetImageValidRectangle`, might result in an incorrect valid region calculation by the framework.

Parameters

<table>
<thead>
<tr>
<th><strong>in</strong></th>
<th><code>image</code></th>
<th>The reference to the image.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><code>rect</code></td>
<td>The value to be set to the image valid rectangle. A NULL indicates that the valid region is the entire image.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.
Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Image is not a valid <code>vx_image</code> reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>The rect does not define a proper valid rectangle.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.72 Object: LUT

3.72.1 Detailed Description

Defines the Look-Up Table Interface.

A lookup table is an array that simplifies run-time computation by replacing computation with a simpler array indexing operation.

Typedefs

- typedef struct vx_lut *vx_lut

  The Look-Up Table (LUT) Object.

Enumerations

- enum vx_lut_attribute_e {
  VX_LUT_TYPE = ((VX_ID_KHRONOS) << 20) | (VX_TYPE_LUT << 8)) + 0x0,
  VX_LUT_COUNT = ((VX_ID_KHRONOS) << 20) | (VX_TYPE_LUT << 8)) + 0x1,
  VX_LUT_SIZE = ((VX_ID_KHRONOS) << 20) | (VX_TYPE_LUT << 8)) + 0x2,
  VX_LUT_OFFSET = ((VX_ID_KHRONOS) << 20) | (VX_TYPE_LUT << 8)) + 0x3

  The Look-Up Table (LUT) attribute list.

Functions

- vx_status VX_API_CALL vxCopyLUT (vx_lut lut, void *user_ptr, vx_enum usage, vx_enum user_mem_type)

  Allows the application to copy from/into a LUT object.

- vx_lut VX_API_CALL vxCreateLUT (vx_context context, vx_enum data_type, vx_size count)

  Creates LUT object of a given type. The value of VX_LUT_OFFSET is equal to 0 for data_type = VX_TYPE_UINT8,
  and (vx_uint32)(count/2) for VX_TYPE_INT16.

- vx_lut VX_API_CALL vxCreateVirtualLUT (vx_graph graph, vx_enum data_type, vx_size count)

  Creates an opaque reference to a LUT object with no direct user access.

- vx_status VX_API_CALL vxMapLUT (vx_lut lut, vx_map_id *map_id, void **ptr, vx_enum usage, vx_enum mem_type, vx_bitfield flags)

  Allows the application to get direct access to LUT object.

- vx_status VX_API_CALL vxQueryLUT (vx_lut lut, vx_enum attribute, void **ptr, vx_size size)

  Queries attributes from a LUT.

- vx_status VX_API_CALL vxReleaseLUT (vx_lut *lut)

  Releases a reference to a LUT object. The object may not be garbage collected until its total reference count is zero.

- vx_status VX_API_CALL vxUnmapLUT (vx_lut lut, vx_map_id map_id)

  Unmap and commit potential changes to LUT object that was previously mapped. Unmapping a LUT invalidates the
  memory location from which the LUT data could be accessed by the application. Accessing this memory location
  after the unmap function completes has an undefined behavior.

3.72.2 Enumeration Type Documentation

enum vx_lut_attribute_e

The Look-Up Table (LUT) attribute list.

Enumerator

  VX_LUT_TYPE  Indicates the value type of the LUT. Read-only. Use a vx_enum.
  VX_LUT_COUNT Indicates the number of elements in the LUT. Read-only. Use a vx_size.
  VX_LUT_SIZE  Indicates the total size of the LUT in bytes. Read-only. Uses a vx_size.
  VX_LUT_OFFSET Indicates the index of the input value = 0. Read-only. Uses a vx_uint32.

Definition at line 1020 of file vx_types.h.
### 3.72.3 Function Documentation

#### vx_lut VX_API_CALL vxCreateLUT ( vx_context context, vx_enum data_type, vx_size count )

Creates LUT object of a given type. The value of VX_LUT_OFFSET is equal to 0 for data_type = VX_TYPE_UINT8, and (vx_uint32)(count/2) for VX_TYPE_INT16.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>context</th>
<th>The reference to the context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data_type</td>
<td>The type of data stored in the LUT.</td>
</tr>
<tr>
<td>in</td>
<td>count</td>
<td>The number of entries desired.</td>
</tr>
</tbody>
</table>

**Note**

data_type can only be VX_TYPE_UINT8 or VX_TYPE_INT16. If data_type is VX_TYPE_UINT8, count should be not greater than 256. If data_type is VX_TYPE_INT16, count should not be greater than 65536.

**Returns**

An LUT reference vx_lut. Any possible errors preventing a successful creation should be checked using vxGetStatus.

#### vx_lut VX_API_CALL vxCreateVirtualLUT ( vx_graph graph, vx_enum data_type, vx_size count )

Creates an opaque reference to a LUT object with no direct user access.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>graph</th>
<th>The reference to the parent graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data_type</td>
<td>The type of data stored in the LUT.</td>
</tr>
<tr>
<td>in</td>
<td>count</td>
<td>The number of entries desired.</td>
</tr>
</tbody>
</table>

See also vxCreateLUT

**Note**

data_type can only be VX_TYPE_UINT8 or VX_TYPE_INT16. If data_type is VX_TYPE_UINT8, count should be not greater than 256. If data_type is VX_TYPE_INT16, count should not be greater than 65536.

**Returns**

An LUT reference vx_lut. Any possible errors preventing a successful creation should be checked using vxGetStatus.

#### vx_status VX_API_CALL vxReleaseLUT ( vx_lut * lut )

Releases a reference to a LUT object. The object may not be garbage collected until its total reference count is zero.

**Parameters**

| in | lut | The pointer to the LUT to release. |
CHAPTER 3. MODULE DOCUMENTATION

Postcondition
After returning from this function the reference is zeroed.

Returns
A \texttt{vx\_status\_e} enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>lut is not a valid \texttt{vx_lut} reference.</td>
</tr>
</tbody>
</table>

\texttt{vx\_status VX\_API\_CALL vxQueryLUT ( vx\_lut lut, vx\_enum attribute, void * ptr, vx\_size size )}

Queries attributes from a LUT.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>lut</th>
<th>The LUT to query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>attribute</td>
<td>The attribute to query. Use a \texttt{vx_lut_attribute_e} enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The location at which to store the resulting value.</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>The size in bytes of the container to which \texttt{ptr} points.</td>
</tr>
</tbody>
</table>

Returns
A \texttt{vx\_status\_e} enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>lut is not a valid \texttt{vx_lut} reference.</td>
</tr>
</tbody>
</table>

\texttt{vx\_status VX\_API\_CALL vxCopyLUT ( vx\_lut lut, void * user\_ptr, vx\_enum usage, vx\_enum user\_mem\_type )}

Allows the application to copy from/into a LUT object.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>lut</th>
<th>The reference to the LUT object that is the source or the destination of the copy.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>user_ptr</td>
<td>The address of the memory location where to store the requested data if the copy was requested in read mode, or from where to get the data to store into the LUT object if the copy was requested in write mode. In the user memory, the LUT is represented as an array with elements of the type corresponding to \texttt{VX_LUT_TYPE}, and with a number of elements equal to the value returned via \texttt{VX_LUT_COUNT}. The accessible memory must be large enough to contain this array: accessible memory in bytes (\geq) sizeof(data_element) * count.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

Parameters

<table>
<thead>
<tr>
<th>In</th>
<th>usage</th>
<th>This declares the effect of the copy with regard to the LUT object using the vx_accessor_e enumeration. Only VX_READ_ONLY and VX_WRITE_ONLY are supported:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VX_READ_ONLY</td>
<td>means that data are copied from the LUT object into the user memory.</td>
</tr>
<tr>
<td></td>
<td>VX_WRITE_ONLY</td>
<td>means that data are copied into the LUT object from the user memory.</td>
</tr>
</tbody>
</table>

| In  | user_mem_type | A vx_memory_type_e enumeration that specifies the memory type of the memory referenced by the user_addr.                           |

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>lut is not a valid vx_lut reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

vx_status VX_API_CALL vxMapLUT ( vx_lut lut, vx_map_id *map_id, void **ptr, vx_enum usage, vx_enum mem_type, vx_bitfield flags )

Allows the application to get direct access to LUT object.

Parameters

<table>
<thead>
<tr>
<th>In</th>
<th>lut</th>
<th>The reference to the LUT object to map.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>map_id</td>
<td>The address of a vx_map_id variable where the function returns a map identifier.</td>
</tr>
<tr>
<td></td>
<td>(*map_id)</td>
<td>must eventually be provided as the map_id parameter of a call to vxUnmapLUT.</td>
</tr>
<tr>
<td>Out</td>
<td>ptr</td>
<td>The address of a pointer that the function sets to the address where the requested data can be accessed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In the mapped memory area, the LUT data are structured as an array with elements of the type corresponding to VX_LUT_TYPE, with a number of elements equal to the value returned via VX_LUT_COUNT. Accessing the memory out of the bound of this array is forbidden and has an undefined behavior. The returned (+ptr) address is only valid between the call to the function and the corresponding call to vxUnmapLUT.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usage</th>
<th>This declares the access mode for the LUT, using the vx_accessor_e enumeration.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• VX_READ_ONLY: after the function call, the content of the memory location pointed by (+ptr) contains the LUT data. Writing into this memory location is forbidden and its behavior is undefined.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• VX_READ_AND_WRITE: after the function call, the content of the memory location pointed by (+ptr) contains the LUT data; writing into this memory location is allowed only for the location of entries and will result in a modification of the affected entries in the LUT object once the LUT is unmapped.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• VX_WRITE_ONLY: after the function call, the memory location pointed by (+ptr) contains undefined data; writing each entry of LUT is required prior to unmapping. Entries not written by the application before unmap will become undefined after unmap, even if they were well defined before map.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>in</th>
<th>mem_type</th>
<th>A vx_memory_type_e enumeration that specifies the type of the memory where the LUT is requested to be mapped.</th>
</tr>
</thead>
</table>

| in  | flags | An integer that allows passing options to the map operation. Use 0 for this option. |

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>lut is not a valid vx_lut reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

Postcondition

vxUnmapLUT with same (+map_id) value.

vx_status VX_API_CALL vxUnmapLUT ( vx_lut lut, vx_map_id map_id )

Unmap and commit potential changes to LUT object that was previously mapped. Unmapping a LUT invalidates the memory location from which the LUT data could be accessed by the application. Accessing this memory location after the unmap function completes has an undefined behavior.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>lut</th>
<th>The reference to the LUT object to unmap.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>The unique map identifier that was returned when calling vxMapLUT.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>lut is not a valid vx_lut reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>
Precondition

\texttt{vxMapLUT} returning the same map\_id value
3.73 Object: Matrix

3.73.1 Detailed Description
Defines the Matrix Object Interface.

Typedefs
• typedef struct _vx_matrix * vx_matrix

The Matrix Object. An MxN matrix of some unit type.

Enumerations
• enum vx_matrix_attribute_e {
  VX_MATRIX_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_MATRIX << 8)) + 0x0,
  VX_MATRIX_ROWS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_MATRIX << 8)) + 0x1,
  VX_MATRIX_COLUMNS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_MATRIX << 8)) + 0x2,
  VX_MATRIX_SIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_MATRIX << 8)) + 0x3,
  VX_MATRIX_ORIGIN = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_MATRIX << 8)) + 0x4,
  VX_MATRIX_PATTERN = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_MATRIX << 8)) + 0x5 }

The matrix attributes.

Functions
• vx_status VX_API_CALL vxCopyMatrix (vx_matrix matrix, void *user_ptr, vx_enum usage, vx_enum user_mem_type)
  Allows the application to copy from/into a matrix object.

• vx_matrix VX_API_CALL vxCreateMatrix (vx_context c, vx_enum data_type, vx_size columns, vx_size rows)
  Creates a reference to a matrix object.

• vx_matrix VX_API_CALL vxCreateMatrixFromPattern (vx_context context, vx_enum pattern, vx_size columns, vx_size rows)
  Creates a reference to a matrix object from a boolean pattern.

• vx_matrix VX_API_CALL vxCreateMatrixFromPatternAndOrigin (vx_context context, vx_enum pattern, vx_size columns, vx_size rows, vx_size origin_col, vx_size origin_row)
  Creates a reference to a matrix object from a boolean pattern, with a user-specified origin.

• vx_matrix VX_API_CALL vxCreateVirtualMatrix (vx_graph graph, vx_enum data_type, vx_size columns, vx_size rows)
  Creates an opaque reference to a matrix object without direct user access.

• vx_status VX_API_CALL vxQueryMatrix (vx_matrix mat, vx_enum attribute, void *ptr, vx_size size)
  Queries an attribute on the matrix object. The object may not be garbage collected until its total reference count is zero.

• vx_status VX_API_CALL vxReleaseMatrix (vx_matrix *mat)
  Releases a reference to a matrix object. The object may not be garbage collected until its total reference count is zero.

3.73.2 Enumeration Type Documentation
enum vx_matrix_attribute_e
The matrix attributes.

Enumerator
  VX_MATRIX_TYPE The value type of the matrix. Read-only. Use a vx_enum parameter.
  VX_MATRIX_ROWS The M dimension of the matrix. Read-only. Use a vx_size parameter.
  VX_MATRIX_COLUMNS The N dimension of the matrix. Read-only. Use a vx_size parameter.
  VX_MATRIX_SIZE The total size of the matrix in bytes. Read-only. Use a vx_size parameter.
**VX_MATRIX_ORIGIN** The origin of the matrix with a default value of \(\lfloor \text{VX\_MATRIX\_COLUMNS}/2 \rfloor, \lfloor \text{VX\_MATRIX\_ROWS}/2 \rfloor\). Read-only. Use a `vx_coordinates2d_t` parameter.

**VX_MATRIX_PATTERN** The pattern of the matrix. See `vx_pattern_e`. Read-only. Use a `vx_enum` parameter. If the matrix was created via `vxCreateMatrixFromPattern` or `vxCreateMatrixFromPatternAndOrigin`, the attribute corresponds to the given pattern. Otherwise the attribute is `VX_PATTERN_OTHER`.

Definition at line 1075 of file `vx_types.h`.

### 3.73.3 Function Documentation

**vx_matrix VX_API_CALL vxCreateMatrix ( vx_context c, vx_enum data_type, vx_size columns, vx_size rows )**

Creates a reference to a matrix object.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>c</th>
<th>The reference to the overall context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data_type</td>
<td>The unit format of the matrix. <code>VX_TYPE_UINT8</code> or <code>VX_TYPE_INT32</code> or <code>VX_TYPE_FLOAT32</code>.</td>
</tr>
<tr>
<td>in</td>
<td>columns</td>
<td>The first dimensionality.</td>
</tr>
<tr>
<td>in</td>
<td>rows</td>
<td>The second dimensionality.</td>
</tr>
</tbody>
</table>

**Returns**

An matrix reference `vx_matrix`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx_matrix VX_API_CALL vxCreateVirtualMatrix ( vx_graph graph, vx_enum data_type, vx_size columns, vx_size rows )**

Creates an opaque reference to a matrix object without direct user access.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the parent graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data_type</td>
<td>The unit format of the matrix. <code>VX_TYPE_UINT8</code> or <code>VX_TYPE_INT32</code> or <code>VX_TYPE_FLOAT32</code>.</td>
</tr>
<tr>
<td>in</td>
<td>columns</td>
<td>The first dimensionality.</td>
</tr>
<tr>
<td>in</td>
<td>rows</td>
<td>The second dimensionality.</td>
</tr>
</tbody>
</table>

See also

- `vxCreateMatrix`

**Returns**

An matrix reference `vx_matrix`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx_status VX_API_CALL vxReleaseMatrix ( vx_matrix * mat )**

Releases a reference to a matrix object. The object may not be garbage collected until its total reference count is zero.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><em>mat</em></td>
<td>The matrix reference to release.</td>
</tr>
</tbody>
</table>

Postcondition

After returning from this function the reference is zeroed.

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td><em>mat</em> is not a valid <code>vx_matrix</code> reference.</td>
</tr>
</tbody>
</table>

`vx_status VX_API_CALL vxQueryMatrix ( vx_matrix mat, vx_enum attribute, void *ptr, vx_size size )`

Queries an attribute on the matrix object.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><em>mat</em></td>
<td>The matrix object to set.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><em>attribute</em></td>
<td>The attribute to query. Use a <code>vx_matrix_attribute_e</code> enumeration.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td><em>ptr</em></td>
<td>The location at which to store the resulting value.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><em>size</em></td>
<td>The size in bytes of the container to which <em>ptr</em> points.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td><em>mat</em> is not a valid <code>vx_matrix</code> reference.</td>
</tr>
</tbody>
</table>

`vx_status VX_API_CALL vxCopyMatrix ( vx_matrix matrix, void *user_ptr, vx_enum usage, vx_enum user_mem_type )`

Allows the application to copy from/into a matrix object.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><em>matrix</em></td>
<td>The reference to the matrix object that is the source or the destination of the copy.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><em>user_ptr</em></td>
<td>The address of the memory location where to store the requested data if the copy was requested in read mode, or from where to get the data to store into the matrix object if the copy was requested in write mode. In the user memory, the matrix is structured as a row-major 2D array with elements of the type corresponding to <code>VX_MATRIX_TYPE</code>, with a number of rows corresponding to <code>VX_MATRIX_ROWS</code> and a number of columns corresponding to <code>VX_MATRIX_COLUMNS</code>. The accessible memory must be large enough to contain this 2D array: accessible memory in bytes ( \geq ) sizeof(data_element) * rows * columns.</td>
</tr>
</tbody>
</table>
Parameters

| in  | usage          | This declares the effect of the copy with regard to the matrix object using the\n|      | vx_accessor_e  | enumeration. Only VX_READ_ONLY and VX_WRITE_ONLY\n|      | enumeration    | are supported:\n|      |               | • VX_READ_ONLY means that data are copied from the matrix object into the\n|      |               | user memory.\n|      |               | • VX_WRITE_ONLY means that data are copied into the matrix object from the\n|      |               | user memory.\n
| in  | user_mem_type  | A vx_memory_type_e enumeration that specifies the memory type of the memory\n|      |               | referenced by the user_addr.\n
Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Matrix is not a valid vx_matrix reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

vx_matrix VX_API_CALL vxCreateMatrixFromPattern ( vx_context context, vx_enum pattern, vx_size columns, vx_size rows )

Creates a reference to a matrix object from a boolean pattern.

See also

vxCreateMatrixFromPatternAndOrigin for a description of the matrix patterns.

Parameters

| in  | context | The reference to the overall context. |
|     | pattern | The pattern of the matrix. See VX_MATRIX_PATTERN. |
|     | columns | The first dimensionality. |
|     | rows    | The second dimensionality. |

Returns

A matrix reference vx_matrix of type VX_TYPE_UINT8. Any possible errors preventing a successful creation should be checked using vxGetStatus.

vx_matrix VX_API_CALL vxCreateMatrixFromPatternAndOrigin ( vx_context context, vx_enum pattern, vx_size columns, vx_size rows, vx_size origin_col, vx_size origin_row )

Creates a reference to a matrix object from a boolean pattern, with a user-specified origin.

The matrix created by this function is of type VX_TYPE_UINT8, with the value 0 representing False, and the value 255 representing True. It supports the patterns as described below:

• VX_PATTERN_BOX is a matrix with dimensions equal to the given number of rows and columns, and all cells equal to 255. Dimensions of 3x3 and 5x5 must be supported.
• VX_PATTERN_CROSS is a matrix with dimensions equal to the given number of rows and columns, which both must be odd numbers. All cells in the center row and center column are equal to 255, and the rest are equal to zero. Dimensions of 3x3 and 5x5 must be supported.

• VX_PATTERN_DISK is a matrix with dimensions equal to the given number of rows (R) and columns (C), where R and C are odd and cell (c, r) is 255 if:
\[(r-R/2 + 0.5)^2 / (R/2)^2 + (c-C/2 + 0.5)^2 / (C/2)^2\] is less than or equal to 1, and 0 otherwise.

A matrix created from pattern is read-only. The behavior when attempting to modify such a matrix is undefined.

Parameters

| in | context | The reference to the overall context. |
| in | pattern | The pattern of the matrix. See VX_MATRIX_PATTERN. |
| in | columns | The first dimensionality. |
| in | rows | The second dimensionality. |
| in | origin_col | The origin (first dimensionality). |
| in | origin_row | The origin (second dimensionality). |

Returns

A matrix reference vx_matrix of type VX_TYPE_UINT8. Any possible errors preventing a successful creation should be checked using vxGetStatus.
### 3.74 Object: Pyramid

#### 3.74.1 Detailed Description

Defines the Image Pyramid Object Interface.

A Pyramid object in OpenVX represents a collection of related images. Typically, these images are created by either downscaling or upscaling a base image, contained in level zero of the pyramid. Successive levels of the pyramid increase or decrease in size by a factor given by the VX_PYRAMID_SCALE attribute. For instance, in a pyramid with 3 levels and VX_SCALE_PYRAMID_HALF, the level one image is one-half the width and one-half the height of the level zero image, and the level two image is one-quarter the width and one-quarter the height of the level zero image. When downscaling or upscaling results in a non-integral number of pixels at any level, fractional pixels always get rounded up to the nearest integer. (E.g., a 3-level image pyramid beginning with level zero having a width of 9 and a scaling of VX_SCALE_PYRAMID_HALF results in the level one image with a width of $9 \times \text{cei}(0.5) = 5$ and a level two image with a width of $5 \times \text{cei}(0.5) = 3$. Position $(r_N, c_N)$ at level $N$ corresponds to position $(r_{N-1}/\text{scale}, c_{N-1}/\text{scale})$ at level $N-1$.

#### Macros

- `#define VX_SCALE_PYRAMID_HALF (0.5f)`
  Use to indicate a half-scale pyramid.
- `#define VX_SCALE_PYRAMID_ORB ((vx_float32)0.8408964f)`
  Use to indicate a ORB scaled pyramid whose scaling factor is $\frac{1}{\sqrt{2}}$.

#### Typedefs

- `typedef struct _vx_pyramid + vx_pyramid`  
  The Image Pyramid object. A set of scaled images.

#### Enumerations

- `enum vx_pyramid_attribute_e {
  VX_PYRAMID_LEVELS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PYRAMID << 8)) + 0x0,
  VX_PYRAMID_SCALE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PYRAMID << 8)) + 0x1,
  VX_PYRAMID_WIDTH = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PYRAMID << 8)) + 0x2,
  VX_PYRAMID_HEIGHT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PYRAMID << 8)) + 0x3,
  VX_PYRAMID_FORMAT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PYRAMID << 8)) + 0x4 }
  The pyramid object attributes.

#### Functions

- `vx_pyramid VX_API_CALL vxCreatePyramid (vx_context context, vx_size levels, vx_float32 scale, vx_uint32 width, vx_uint32 height, vx_df_image format)`  
  Creates a reference to a pyramid object of the supplied number of levels.

- `vx_pyramid VX_API_CALL vxCreateVirtualPyramid (vx_graph graph, vx_size levels, vx_float32 scale, vx_uint32 width, vx_uint32 height, vx_df_image format)`  
  Creates a reference to a virtual pyramid object of the supplied number of levels.

- `vx_image VX_API_CALL vxGetPyramidLevel (vx_pyramid pyr, vx_uint32 index)`  
  Retrieves a level of the pyramid as a vx_image, which can be used elsewhere in OpenVX. A call to vxReleaseImage is necessary to release an image for each call of vxGetPyramidLevel.

- `vx_status VX_API_CALL vxQueryPyramid (vx_pyramid pyr, vx_enum attribute, void *ptr, vx_size size)`  
  Queries an attribute from an image pyramid.

- `vx_status VX_API_CALL vxReleasePyramid (vx_pyramid *pyr)`  
  Releases a reference to a pyramid object. The object may not be garbage collected until its total reference count is zero.
3.74.2 Enumeration Type Documentation

```c
enum vx_pyramid_attribute_e
```

The pyramid object attributes.

**Enumerator**

- **VX_PYRAMID_LEVELS**  The number of levels of the pyramid. Read-only. Use a `vx_size` parameter.
- **VX_PYRAMID_SCALE**  The scale factor between each level of the pyramid. Read-only. Use a `vx_float32` parameter.
- **VX_PYRAMID_WIDTH**  The width of the 0th image in pixels. Read-only. Use a `vx_uint32` parameter.
- **VX_PYRAMID_HEIGHT**  The height of the 0th image in pixels. Read-only. Use a `vx_uint32` parameter.
- **VX_PYRAMID_FORMAT**  The `vx_df_image_e` format of the image. Read-only. Use a `vx_df_image` parameter.

Definition at line 1115 of file `vx_types.h`.

3.74.3 Function Documentation

```c
vx_pyramid VX_API_CALL vxCreatePyramid ( vx_context context, vx_size levels, vx_float32 scale, vx_uint32 width, vx_uint32 height, vx_df_image format )
```

Creates a reference to a pyramid object of the supplied number of levels.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>context</code></td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td><code>levels</code></td>
<td>The number of levels desired. This is required to be a non-zero value.</td>
</tr>
<tr>
<td><code>scale</code></td>
<td>Used to indicate the scale between pyramid levels. This is required to be a non-zero positive value. <code>VX_SCALE_PYRAMID_HALF</code> and <code>VX_SCALE_PYRAMID_ORB</code> must be supported.</td>
</tr>
<tr>
<td><code>width</code></td>
<td>The width of the 0th level image in pixels.</td>
</tr>
<tr>
<td><code>height</code></td>
<td>The height of the 0th level image in pixels.</td>
</tr>
<tr>
<td><code>format</code></td>
<td>The format of all images in the pyramid. NV12, NV21, IYUV, UYVY and YUYV formats are not supported.</td>
</tr>
</tbody>
</table>

**Returns**

A pyramid reference `vx_pyramid` containing the sub-images. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

```c
vx_pyramid VX_API_CALL vxCreateVirtualPyramid ( vx_graph graph, vx_size levels, vx_float32 scale, vx_uint32 width, vx_uint32 height, vx_df_image format )
```

Creates a reference to a virtual pyramid object of the supplied number of levels.

Virtual Pyramids can be used to connect Nodes together when the contents of the pyramids will not be accessed by the user of the API. All of the following constructions are valid:

```c
00001 vx_context context = vxCreateContext();
00002 vx_graph graph = vxCreateGraph(context);
00003 vx_pyramid virt[] = {
00004  vxCreateVirtualPyramid(graph, 4, VX_SCALE_PYRAMID_HALF, 0, 0, VX_DF_IMAGE_VIRT), // no dimension and format specified for level 0
00005  vxCreateVirtualPyramid(graph, 4, VX_SCALE_PYRAMID_HALF, 640, 480, VX_DF_IMAGE_VIRT), // no format specified.
00006  vxCreateVirtualPyramid(graph, 4, VX_SCALE_PYRAMID_HALF, 640, 480, VX_DF_IMAGE_U8), // no access
00007   );
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>graph</code></td>
<td>The reference to the parent graph.</td>
</tr>
<tr>
<td><code>levels</code></td>
<td>The number of levels desired. This is required to be a non-zero value.</td>
</tr>
<tr>
<td><code>scale</code></td>
<td>Used to indicate the scale between pyramid levels. This is required to be a non-zero positive value. <code>VX_SCALE_PYRAMID_HALF</code> and <code>VX_SCALE_PYRAMID_ORB</code> must be supported.</td>
</tr>
<tr>
<td><code>width</code></td>
<td>The width of the 0th level image in pixels. This may be set to zero to indicate to the interface that the value is unspecified.</td>
</tr>
<tr>
<td><code>height</code></td>
<td>The height of the 0th level image in pixels. This may be set to zero to indicate to the interface that the value is unspecified.</td>
</tr>
<tr>
<td><code>format</code></td>
<td>The format of all images in the pyramid. This may be set to <code>VX_DF_IMAGE_VIRT</code> to indicate that the format is unspecified.</td>
</tr>
</tbody>
</table>

Returns

A pyramid reference `vx_pyramid`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

Note

Images extracted with `vxGetPyramidLevel` behave as Virtual Images and cause `vxMapImagePatch` to return errors.

`vx_status VX_API_CALL vxReleasePyramid ( vx_pyramid ∗ pyr )`

Releases a reference to a pyramid object. The object may not be garbage collected until its total reference count is zero.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pyr</code></td>
<td>The pointer to the pyramid to release.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>VX_SUCCESS</code></td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td><code>VX_ERROR_INVALID_REFERENCE</code></td>
<td><code>pyr</code> is not a valid <code>vx_pyramid</code> reference.</td>
</tr>
</tbody>
</table>

Postcondition

After returning from this function the reference is zeroed.

`vx_status VX_API_CALL vxQueryPyramid ( vx_pyramid pyr, vx_enum attribute, void ∗ ptr, vx_size size )`

Queries an attribute from an image pyramid.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pyr</code></td>
<td>The pyramid to query.</td>
</tr>
<tr>
<td><code>attribute</code></td>
<td>The attribute for which to query. Use a <code>vx_pyramid_attribute_e</code> enumeration.</td>
</tr>
<tr>
<td><code>ptr</code></td>
<td>The location at which to store the resulting value.</td>
</tr>
<tr>
<td><code>size</code></td>
<td>The size in bytes of the container to which <code>ptr</code> points.</td>
</tr>
</tbody>
</table>
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td><code>pyr</code> is not a valid <code>vx_pyramid</code> reference.</td>
</tr>
</tbody>
</table>

`vx_image VX_API_CALL vxGetPyramidLevel ( vx_pyramid pyr, vx_uint32 index )`

Retrieves a level of the pyramid as a `vx_image`, which can be used elsewhere in OpenVX. A call to `vxReleaseImage` is necessary to release an image for each call of `vxGetPyramidLevel`.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>pyr</code></th>
<th>The pyramid object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>index</code></td>
<td>The index of the level, such that index is less than levels.</td>
</tr>
</tbody>
</table>

Returns

A `vx_image` reference. Any possible errors preventing a successful function completion should be checked using `vxGetStatus`. 
CHAPTER 3. MODULE DOCUMENTATION

3.75 Object: Remap

3.75.1 Detailed Description

Defines the Remap Object Interface.

Typedefs

- typedef struct _vx_remap * vx_remap

  The remap table object. A remap table contains per-pixel mapping of output pixels to input pixels.

Enumerations

- enum vx_remap_attribute_e
  - VX_REMAP_SOURCE_WIDTH = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_REMAP << 8)) + 0x0,
  - VX_REMAP_SOURCE_HEIGHT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_REMAP << 8)) + 0x1,
  - VX_REMAP_DESTINATION_WIDTH = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_REMAP << 8)) + 0x2,
  - VX_REMAP_DESTINATION_HEIGHT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_REMAP << 8)) + 0x3

  The remap object attributes.

Functions

- vx_status VX_API_CALL vxCopyRemapPatch (vx_remap remap, const vx_rectangle_t *rect, vx_size user←_stride_y, void *user_ptr, vx_enum user_coordinate_type, vx_enum usage, vx_enum user_mem_type)

  Allows the application to copy a rectangular patch from/into a remap object.

- vx_remap VX_API_CALL vxCreateRemap (vx_context context, vx_uint32 src_width, vx_uint32 src_height, vx_uint32 dst_width, vx_uint32 dst_height)

  Creates a remap table object.

- vx_remap VX_API_CALL vxCreateVirtualRemap (vx_graph graph, vx_uint32 src_width, vx_uint32 src_height, vx_uint32 dst_width, vx_uint32 dst_height)

  Creates an opaque reference to a remap table object without direct user access.

- vx_status VX_API_CALL vxMapRemapPatch (vx_remap remap, const vx_rectangle_t *rect, vx_map_id *map_id, vx_size *stride_y, void **ptr, vx_enum coordinate_type, vx_enum usage, vx_enum mem_type)

  Allows the application to get direct access to a rectangular patch of a remap object.

- vx_status VX_API_CALL vxQueryRemap (vx_remap table, vx_enum attribute, void *ptr, vx_size size)

  Queries attributes from a Remap table.

- vx_status VX_API_CALL vxReleaseRemap (vx_remap *table)

  Releases a reference to a remap table object. The object may not be garbage collected until its total reference count is zero.

- vx_status VX_API_CALL vxUnmapRemapPatch (vx_remap remap, vx_map_id map_id)

  Unmap and commit potential changes to a remap object patch that was previously mapped.

3.75.2 Enumeration Type Documentation

enum vx_remap_attribute_e

The remap object attributes.

Enumerator

- VX_REMAP_SOURCE_WIDTH  The source width. Read-only. Use a vx_uint32 parameter.
- VX_REMAP_SOURCE_HEIGHT  The source height. Read-only. Use a vx_uint32 parameter.
- VX_REMAP_DESTINATION_WIDTH  The destination width. Read-only. Use a vx_uint32 parameter.
- VX_REMAP_DESTINATION_HEIGHT  The destination height. Read-only. Use a vx_uint32 parameter.

Definition at line 1131 of file vx_types.h.
### 3.75.3 Function Documentation

`vx_remap VX_API_CALL vxCreateRemap ( vx_context context, vx_uint32 src_width, vx_uint32 src_height, vx_uint32 dst_width, vx_uint32 dst_height )`

Creates a remap table object.

**Parameters**

| in | context | The reference to the overall context. |
| in | src_width | Width of the source image in pixels. |
| in | src_height | Height of the source image in pixels. |
| in | dst_width | Width of the destination image in pixels. |
| in | dst_height | Height of the destination image in pixels. |

**Returns**

A remap reference `vx_remap`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

`vx_remap VX_API_CALL vxCreateVirtualRemap ( vx_graph graph, vx_uint32 src_width, vx_uint32 src_height, vx_uint32 dst_width, vx_uint32 dst_height )`

Creates an opaque reference to a remap table object without direct user access.

**Parameters**

| in | graph | The reference to the parent graph. |
| in | src_width | Width of the source image in pixel. |
| in | src_height | Height of the source image in pixels. |
| in | dst_width | Width of the destination image in pixels. |
| in | dst_height | Height of the destination image in pixels. |

**See also**

`vxCreateRemap`

**Returns**

A remap reference `vx_remap`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

`vx_status VX_API_CALL vxReleaseRemap ( vx_remap *table )`

Releases a reference to a remap table object. The object may not be garbage collected until its total reference count is zero.

**Parameters**

| in | table | The pointer to the remap table to release. |

**Postcondition**

After returning from this function the reference is zeroed.
CHAPTER 3. MODULE DOCUMENTATION

Returns

A **vx_status_e** enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>table is not a valid vx_remap reference.</td>
</tr>
</tbody>
</table>

**vx_status VX_API_CALL vxMapRemapPatch ( vx_remap remap, const vx_rectangle_t * rect, vx_map_id * map_id, vx_size * stride_y, void ** ptr, vx_enum coordinate_type, vx_enum usage, vx_enum mem_type )**

Allows the application to get direct access to a rectangular patch of a remap object.

The patch is specified within the destination dimensions and its data provide the corresponding coordinate within the source dimensions. The patch is mapped as a 2D array of elements of the type associated with the **coordinate_type** parameter (i.e., **vx_coordinates2df_t** for **VX_TYPE_COORDINATES2DF**). The memory layout of the mapped 2D array follows a row-major order where rows are compact (without any gap between elements), and where the potential padding after each lines is determined by (*stride_y).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>remap</strong></td>
<td>The reference to the remap object that contains the patch to map.</td>
</tr>
<tr>
<td><strong>rect</strong></td>
<td>The coordinates of remap patch. The patch must be specified within the bounds of the remap destination dimensions (<strong>VX_REMAP_DESTINATION_WIDTH</strong> x <strong>VX_REMAP_DESTINATION_HEIGHT</strong>). (start_x, start_y) gives the coordinate of the topleft element inside the patch, while (end_x, end_y) gives the coordinate of the bottomright element out of the patch.</td>
</tr>
<tr>
<td><strong>map_id</strong></td>
<td>The address of a <strong>vx_map_id</strong> variable where the function returns a map identifier.</td>
</tr>
<tr>
<td></td>
<td>• (+map_id) must eventually be provided as the map_id parameter of a call to <strong>vxUnmapRemapPatch</strong>.</td>
</tr>
<tr>
<td><strong>stride_y</strong></td>
<td>The address of a vx_size variable where the function returns the difference between the address of the first element of two successive lines in the mapped remap patch. The stride value follows the following rule: (+stride_y) = sizeof(&lt;ELEMENT_TYPE&gt;) * (rect-&gt;end_x - rect-&gt;start_x)</td>
</tr>
<tr>
<td><strong>ptr</strong></td>
<td>The address of a pointer where the function returns where remap patch data can be accessed. (+ptr) is the address of the the top-left element of the remap patch. The returned (+ptr) address is only valid between the call to this function and the corresponding call to <strong>vxUnmapRemapPatch</strong>.</td>
</tr>
<tr>
<td><strong>coordinate_type</strong></td>
<td>This declares the type of the source coordinate data that the application wants to access in the remap patch. It must be <strong>VX_TYPE_COORDINATES2DF</strong>.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usage</th>
<th>This declares the access mode for the remap patch, using the vx_accessor_e enumeration.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• VX_READ_ONLY: after the function call, the content of the memory location pointed by (∗ptr) contains the remap patch data. Writing into this memory location is forbidden and its behavior is undefined.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• VX_READ_AND_WRITE: after the function call, the content of the memory location pointed by (∗ptr) contains the remap patch data; writing into this memory is allowed for the location of elements only and will result in a modification of the written elements in the remap object once the patch is unmapped. Writing into a gap between element lines (when (∗stride_y) &gt; sizeof(&lt;ELEMENT_TYPE&gt;) * (rect-&gt;end_x - rect-&gt;start_x)) is forbidden and its behavior is undefined.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• VX_WRITE_ONLY: after the function call, the memory location pointed by (∗ptr) contains undefined data; writing each element of the patch is required prior to unmapping. Elements not written by the application before unmap will become undefined after unmap, even if they were well defined before map. Like for VX_READ_AND_WRITE, writing into a gap between element lines is forbidden and its behavior is undefined.</td>
<td></td>
</tr>
</tbody>
</table>

| in  | mem_type | A vx_memory_type_e enumeration that specifies the type of the memory where the remap patch is requested to be mapped. |

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>remap is not a valid vx_remap reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

Postcondition

vxUnmapRemapPatch with same (∗map_id) value.

vx_status VX_API_CALL vxUnmapRemapPatch ( vx_remap remap, vx_map_id map_id )

Unmap and commit potential changes to a remap object patch that was previously mapped.

Unmapping a remap patch invalidates the memory location from which the patch could be accessed by the application. Accessing this memory location after the unmap function completes has an undefined behavior.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>remap</th>
<th>The reference to the remap object to unmap.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>map_id</td>
<td>The unique map identifier that was returned by vxMapRemapPatch.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.
Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>remap is not a valid vx_remap reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

Precondition

vxMapRemapPatch with same map_id value

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

Allows the application to copy a rectangular patch from/into a remap object.

The patch is specified within the destination dimensions and its data provide the corresponding coordinate within the source dimensions. The patch in user memory is a 2D array of elements of the type associated with the `coordinate_type` parameter (i.e., `vx_coordinates2df_t` for `VX_TYPE_COORDINATES2DF`). The memory layout of this array follows a row-major order where rows are compact (without any gap between elements), and where the potential padding after each line is determined by the `user_stride_y` parameter.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remap</td>
<td>The reference to the remap object that is the source or the destination of the patch copy.</td>
</tr>
<tr>
<td>rect</td>
<td>The coordinates of remap patch. The patch must be specified within the bounds of the remap destination dimensions (<code>VX_REMAP_DESTINATION_WIDTH</code> x <code>VX_REMAP_DESTINATION_HEIGHT</code>). <code>(start_x, start_y)</code> gives the coordinate of the topleft element inside the patch, while <code>(end_x, end_y)</code> gives the coordinate of the bottomright element out of the patch.</td>
</tr>
<tr>
<td>user_stride_y</td>
<td>The difference between the address of the first element of two successive lines of the remap patch in user memory (pointed by <code>user_ptr</code>). The layout of the user memory must follow a row major order and <code>user_stride_y</code> must follow the following rule: <code>user_stride_y &gt;= sizeof(&lt;ELEMENT_TYPE&gt;) * (rect-&gt;end_x - rect-&gt;start_x)</code>.</td>
</tr>
<tr>
<td>user_ptr</td>
<td>The address of the user memory location where to store the requested remap data if the copy was requested in read mode, or from where to get the remap data to store into the remap object if the copy was requested in write mode. <code>user_ptr</code> is the address of the the top-left element of the remap patch. The accessible user memory must be large enough to contain the specified patch with the specified layout: accessible memory in bytes &gt;= <code>(rect-&gt;end_y - rect-&gt;start_y) * user_stride_y</code>.</td>
</tr>
<tr>
<td>user_coordinate_type</td>
<td>This declares the type of the source coordinate remap data in the user memory. It must be <code>VX_TYPE_COORDINATES2DF</code>.</td>
</tr>
<tr>
<td>usage</td>
<td>This declares the effect of the copy with regard to the remap object using the <code>vx_accessor_e</code> enumeration. Only <code>VX_READ_ONLY</code> and <code>VX_WRITE_ONLY</code> are supported:</td>
</tr>
<tr>
<td></td>
<td>• <code>VX_READ_ONLY</code> means that data is copied from the remap object into the user memory pointer by <code>user_ptr</code>. The potential padding after each line in user memory will stay unchanged.</td>
</tr>
<tr>
<td></td>
<td>• <code>VX_WRITE_ONLY</code> means that data is copied into the remap object from the user memory.</td>
</tr>
<tr>
<td>user_mem_type</td>
<td>A <code>vx_memory_type_e</code> enumeration that specifies the type of the memory pointer by <code>user_ptr</code>.</td>
</tr>
</tbody>
</table>
Returns

A \texttt{vx\_status\_e} enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{VX_SUCCESS}</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>\texttt{VX_ERROR_INVALID_REFERENCE}</td>
<td>remap is not a valid \texttt{vx_remap} reference.</td>
</tr>
<tr>
<td>\texttt{VX_ERROR_INVALID_PARAMETERS}</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

\texttt{vx\_status VX\_API\_CALL vxQueryRemap ( vx\_remap table, vx\_enum attribute, void * ptr, vx\_size size )}

Queries attributes from a Remap table.

Parameters

<table>
<thead>
<tr>
<th>\texttt{in table}</th>
<th>The remap to query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{in attribute}</td>
<td>The attribute to query. Use a \texttt{vx_remap_attribute_e} enumeration.</td>
</tr>
<tr>
<td>\texttt{out ptr}</td>
<td>The location at which to store the resulting value.</td>
</tr>
<tr>
<td>\texttt{in size}</td>
<td>The size in bytes of the container to which \texttt{ptr} points.</td>
</tr>
</tbody>
</table>

Returns

A \texttt{vx\_status\_e} enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{VX_SUCCESS}</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>\texttt{VX_ERROR_INVALID_REFERENCE}</td>
<td>table is not a valid \texttt{vx_remap} reference.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.76 Object: Scalar

3.76.1 Detailed Description
Defines the Scalar Object interface.

Typedefs

- typedef struct _vx_scalar * vx_scalar
  
  An opaque reference to a scalar.

Enumerations

- enum vx_scalar_attribute_e { VX_SCALAR_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_SCALAR << 8)) + 0x0 }
  
  The scalar attributes list.

- enum vx_scalar_operation_e {
    VX_SCALAR_OP_AND = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0x0,
    VX_SCALAR_OP_OR = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0x1,
    VX_SCALAR_OP_XOR = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0x2,
    VX_SCALAR_OP_NAND = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0x3,
    VX_SCALAR_OP_EQUAL = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0x4,
    VX_SCALAR_OP_NOTEQUAL = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0x5,
    VX_SCALAR_OP_LESS = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0x6,
    VX_SCALAR_OP_LESSEQ = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0x7,
    VX_SCALAR_OP_GREATER = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0x8,
    VX_SCALAR_OP_GREATEREQ = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0x9,
    VX_SCALAR_OP_ADD = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0xA,
    VX_SCALAR_OP_SUBTRACT = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0xB,
    VX_SCALAR_OP_MULTIPLY = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0xC,
    VX_SCALAR_OP_DIVIDE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0xD,
    VX_SCALAR_OP_MODULUS = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0xE,
    VX_SCALAR_OP_MIN = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0xF,
    VX_SCALAR_OP_MAX = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_SCALAR_OPERATION << 12)) + 0x10 }

  A type of operation in which both operands are scalars.

Functions

- vx_status VX_API_CALL vxCopyScalar (vx_scalar scalar, void *user_ptr, vx_enum usage, vx_enum user_mem_type)

  Allows the application to copy from/into a scalar object.


- **vx_status VX_API_CALL vxCopyScalarWithSize (vx_scalar scalar, vx_size size, void *user_ptr, vx_enum usage, vx_enum user_mem_type)**
  Allows the application to copy from/into a scalar object with size.

- **vx_scalar VX_API_CALL vxCreateScalar (vx_context context, vx_enum data_type, const void *ptr)**
  Creates a reference to a scalar object. Also see Node Parameters.

- **vx_scalar VX_API_CALL vxCreateScalarWithSize (vx_context context, vx_enum data_type, const void *ptr, vx_size size)**
  Creates a reference to a scalar object. Also see Node Parameters.

- **vx_scalar VX_API_CALL vxCreateVirtualScalar (vx_graph graph, vx_enum data_type)**
  Creates an opaque reference to a scalar object with no direct user access.

- **vx_status VX_API_CALL vxQueryScalar (vx_scalar scalar, vx_enum attribute, void *ptr, vx_size size)**
  Queries attributes from a scalar.

- **vx_status VX_API_CALL vxReleaseScalar (vx_scalar *scalar)**
  Releases a reference to a scalar object. The object may not be garbage collected until its total reference count is zero.

### 3.76.2 Typedef Documentation

typedef struct _vx_scalar* vx_scalar

An opaque reference to a scalar.

A scalar can be up to 64 bits wide.

See also

vxCreateScalar

Definition at line 172 of file vx_types.h.

### 3.76.3 Enumeration Type Documentation

**enum vx_scalar_attribute_e**

The scalar attributes list.

**Enumerator**

- **VX_SCALAR_TYPE** Queries the type of atomic that is contained in the scalar. Read-only. Use a vx_enum parameter.

Definition at line 971 of file vx_types.h.

**enum vx_scalar_operation_e**

A type of operation in which both operands are scalars.

See also

Object: Scalar

**Enumerator**

- **VX_SCALAR_OP_AND** logical and.
- **VX_SCALAR_OP_OR** logical or.
- **VX_SCALAR_OP_XOR** logical exclusive or.
- **VX_SCALAR_OP_NAND** logical nand.
- **VX_SCALAR_OP_EQUAL** comparison (equal).
- **VX_SCALAR_OP_NOTEQUAL** comparison (not equal).
- **VX_SCALAR_OP_LESS** comparison (less than).
- **VX_SCALAR_OP_LESSEQ** comparison (less than or equal to).
**VX_SCALAR_OP_GREATER** comparison (greater than).
**VX_SCALAR_OP_GREATEREQ** comparison (greater than or equal to).
**VX_SCALAR_OP_ADD** arithmetic addition.
**VX_SCALAR_OP_SUBTRACT** arithmetic subtraction.
**VX_SCALAR_OP_MULTIPLY** arithmetic multiplication.
**VX_SCALAR_OP_DIVIDE** arithmetic division.
**VX_SCALAR_OP_MODULUS** arithmetic (modulo operator).
**VX_SCALAR_OP_MIN** minimum of two scalars.
**VX_SCALAR_OP_MAX** maximum of two scalars.

Definition at line 980 of file vx_types.h.

### 3.76.4 Function Documentation

**vx_scalar VX_API_CALL vxCreateScalar ( vx_context context, vx_enum data_type, const void * ptr )**

Creates a reference to a scalar object. Also see Node Parameters.

**Parameters**

- **in** `context` The reference to the system context.
- **in** `data_type` The type of data to hold. Must be greater than `VX_TYPE_INVALID` and less than or equal to `VX_TYPE_VENDOR_STRUCT_END`. Or must be a `vx_enum` returned from `vxRegisterUserStruct`.
- **in** `ptr` The pointer to the initial value of the scalar.

**Returns**

A scalar reference `vx_scalar`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx_scalar VX_API_CALL vxCreateScalarWithSize ( vx_context context, vx_enum data_type, const void * ptr, vx_size size )**

Creates a reference to a scalar object. Also see Node Parameters.

**Parameters**

- **in** `context` The reference to the system context.
- **in** `data_type` The type of data to hold. Must be greater than `VX_TYPE_INVALID` and less than or equal to `VX_TYPE_VENDOR_STRUCT_END`. Or must be a `vx_enum` returned from `vxRegisterUserStruct`.
- **in** `ptr` The pointer to the initial value of the scalar.
- **in** `size` Size of data at ptr in bytes.

**Returns**

A scalar reference `vx_scalar`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx_scalar VX_API_CALL vxCreateVirtualScalar ( vx_graph graph, vx_enum data_type )**

Creates an opaque reference to a scalar object with no direct user access.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the parent graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data_type</td>
<td>The type of data to hold. Must be greater than VX_TYPE_INVALID and less than or equal to VX_TYPE_VENDOR_STRUCT_END. Or must be a vx_enum returned from vxRegisterUserStruct.</td>
</tr>
</tbody>
</table>

See also

vxCreateScalar

Returns

A scalar reference vx_scalar. Any possible errors preventing a successful creation should be checked using vxGetStatus.

vx_status VX_API_CALL vxReleaseScalar ( vx_scalar * scalar )

Releases a reference to a scalar object. The object may not be garbage collected until its total reference count is zero.

Parameters

| in | scalar | The pointer to the scalar to release. |

Postcondition

After returning from this function the reference is zeroed.

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>scalar is not a valid vx_scalar reference.</td>
</tr>
</tbody>
</table>

vx_status VX_API_CALL vxQueryScalar ( vx_scalar scalar, vx_enum attribute, void * ptr, vx_size size )

Queries attributes from a scalar.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>scalar</th>
<th>The scalar object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>attribute</td>
<td>The enumeration to query. Use a vx_scalar_attribute_e enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The location at which to store the resulting value.</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>The size of the container to which ptr points.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.
### Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>scalar is not a valid vx_scalar reference.</td>
</tr>
</tbody>
</table>

### vx_status VX_API_CALL vxCopyScalar ( vx_scalar scalar, void *user_ptr, vx_enum usage, vx_enum user_mem_type )

Allows the application to copy from/into a scalar object.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>scalar</th>
<th>The reference to the scalar object that is the source or the destination of the copy.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>user_ptr</td>
<td>The address of the memory location where to store the requested data if the copy was requested in read mode, or from where to get the data to store into the scalar object if the copy was requested in write mode. In the user memory, the scalar is a variable of the type corresponding to VX_SCALAR_TYPE. The accessible memory must be large enough to contain this variable.</td>
</tr>
</tbody>
</table>
| in | usage | This declares the effect of the copy with regard to the scalar object using the vx_accessor_e enumeration. Only VX_READ_ONLY and VX_WRITE_ONLY are supported:  
  • VX_READ_ONLY means that data are copied from the scalar object into the user memory.  
  • VX_WRITE_ONLY means that data are copied into the scalar object from the user memory. |
| in | user_mem_type | A vx_memory_type_e enumeration that specifies the memory type of the memory referenced by the user_addr. |

#### Returns

A vx_status_e enumeration.

#### Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>scalar is not a valid vx_scalar reference.</td>
</tr>
<tr>
<td>VX_ERROR.INVALID_PARAMETERS</td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

### vx_status VX_API_CALL vxCopyScalarWithSize ( vx_scalar scalar, vx_size size, void *user_ptr, vx_enum usage, vx_enum user_mem_type )

Allows the application to copy from/into a scalar object with size.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>scalar</th>
<th>The reference to the scalar object that is the source or the destination of the copy.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>The size in bytes of the container to which user_ptr points.</td>
</tr>
<tr>
<td>in</td>
<td>user_ptr</td>
<td>The address of the memory location where to store the requested data if the copy was requested in read mode, or from where to get the data to store into the scalar object if the copy was requested in write mode. In the user memory, the scalar is a variable of the type corresponding to VX_SCALAR_TYPE. The accessible memory must be large enough to contain this variable.</td>
</tr>
</tbody>
</table>
Parameters

| in  | usage       | This declares the effect of the copy with regard to the scalar object using the vx_accessor_e enumeration. Only VX_READ_ONLY and VX_WRITE_ONLY are supported:
|     |             | • VX_READ_ONLY means that data are copied from the scalar object into the user memory.
|     |             | • VX_WRITE_ONLY means that data are copied into the scalar object from the user memory. |

| in  | user_mem_type | A vx_memory_type_e enumeration that specifies the memory type of the memory referenced by the user_addr. |

Returns

A vx_status_e enumeration.

Return values

| vx_error_invalid_reference | The scalar reference is not actually a scalar reference. |
| vx_error_invalid_parameters | An other parameter is incorrect. |
3.77 Object: Threshold

3.77.1 Detailed Description

Defines the Threshold Object Interface.

**Typedefs**

- typedef struct _vx_threshold *vx_threshold
  
  The Threshold Object. A thresholding object contains the types and limit values of the thresholding required.

**Enumerations**

- enum vx_threshold_attribute_e {
  
  VX_THRESHOLD_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_THRESHOLD << 8)) + 0x0,
  VX_THRESHOLD_INPUT_FORMAT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_THRESHOLD << 8)) + 0x7,
  VX_THRESHOLD_OUTPUT_FORMAT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_THRESHOLD << 8)) + 0x8
}

  The threshold attributes.

- enum vx_threshold_type_e {
  
  VX_THRESHOLD_TYPE_BINARY = ((( VX_ID_KHRONOS ) << 34) | ( VX_ENUM_THRESHOLD_TYPE << 12)) + 0x0,
  VX_THRESHOLD_TYPE_RANGE = ((( VX_ID_KHRONOS ) << 34) | ( VX_ENUM_THRESHOLD_TYPE << 12)) + 0x1
}

  The Threshold types.

**Functions**

- vx_status VX_API_CALL vxCopyThresholdOutput (vx_threshold thresh, vx_pixel_value_t *true_value_ptr, vx_pixel_value_t *false_value_ptr, vx_enum usage, vx_enum user_mem_type)
  
  Allows the application to copy the true and false output values from/into a threshold object.

- vx_status VX_API_CALL vxCopyThresholdRange (vx_threshold thresh, vx_pixel_value_t *lower_value_ptr, vx_pixel_value_t *upper_value_ptr, vx_enum usage, vx_enum user_mem_type)
  
  Allows the application to copy thresholding values from/into a threshold object with type VX_THRESHOLD_TYPE←_RANGE.

- vx_status VX_API_CALL vxCopyThresholdValue (vx_threshold thresh, vx_pixel_value_t *value_ptr, vx‿e¬num usage, vx_enum user_mem_type)
  
  Allows the application to copy the thresholding value from/into a threshold object with type VX_THRESHOLD_TYP¬E_BINARY.

- vx_threshold VX_API_CALL vxCreateThresholdForImage (vx_context context, vx_enum thresh_type, vx_df_image input_format, vx_df_image output_format)
  
  Creates a threshold object and returns a reference to it.

- vx_threshold VX_API_CALL vxCreateVirtualThresholdForImage (vx_graph graph, vx_enum thresh_type, vx_df_image input_format, vx_df_image output_format)
  
  Creates an opaque reference to a threshold object without direct user access.

- vx_status VX_API_CALL vxQueryThreshold (vx_threshold thresh, vx_enum attribute, void *ptr, vx_size size)
  
  Queries an attribute on the threshold object.

- vx_status VX_API_CALL vxReleaseThreshold (vx_threshold *thresh)
  
  Releases a reference to a threshold object. The object may not be garbage collected until its total reference count is zero.

- vx_status VX_API_CALL vxSetThresholdAttribute (vx_threshold thresh, vx_enum attribute, const void *ptr, vx_size size)
  
  Sets attributes on the threshold object.
3.77.2 Enumeration Type Documentation

```c
enum vx_threshold_type_e

The Threshold types.

Enumerator

VX_THRESHOLD_TYPE_BINARY A threshold with only 1 value.

VX_THRESHOLD_TYPE_RANGE A threshold with 2 values (upper/lower). Use with Canny Edge Detection.

Definition at line 1053 of file vx_types.h.
```

```c
enum vx_threshold_attribute_e

The threshold attributes.

Enumerator

VX_THRESHOLD_TYPE The value type of the threshold. Read-only. Use a vx_enum parameter. Will contain a vx_threshold_type_e.

VX_THRESHOLD_INPUT_FORMAT The input image format the threshold was created for. Read-only. Use a vx_enum parameter. Will contain a vx_df_image_e.

VX_THRESHOLD_OUTPUT_FORMAT The output image format the threshold was created for. Read-only. Use a vx_enum parameter. Will contain a vx_df_image_e.

Definition at line 1063 of file vx_types.h.
```

3.77.3 Function Documentation

```c
vx_threshold VX_API_CALL vxCreateThresholdForImage ( vx_context context, vx_enum thresh_type, vx_df_image input_format, vx_df_image output_format )

Creates a threshold object and returns a reference to it.

The threshold object defines the parameters of a thresholding operation to an input image, that generates an output image that can have a different format. The thresholding ‘false’ or ‘true’ output values are specified per pixel channels of the output format and can be modified with vxCopyThresholdOutput. The default ‘false’ output value of pixels channels should be 0, and the default ‘true’ value should be non-zero. For standard image formats, default output pixel values are defined as following:

- VX_DF_IMAGE_RGB : false=(0, 0, 0), true=(255,255,255)
- VX_DF_IMAGE_RGBX : false=(0, 0, 0, 0), true=(255,255,255,255)
- VX_DF_IMAGE_NV12 : false=(0, 0, 0), true=(255,255,255)
- VX_DF_IMAGE_NV21 : false=(0, 0, 0), true=(255,255,255)
- VX_DF_IMAGE_UYVY : false=(0, 0, 0), true=(255,255,255)
- VX_DF_IMAGE_YUYV : false=(0, 0, 0), true=(255,255,255)
- VX_DF_IMAGE_IYUV : false=(0, 0, 0), true=(255,255,255)
- VX_DF_IMAGE_YUV4 : false=(0, 0, 0), true=(255,255,255)
- VX_DF_IMAGE_U8 : false=0, true=0xFF
- VX_DF_IMAGE_S16 : false=0, true=-1
- VX_DF_IMAGE_U16 : false=0, true=0xFFFF
- VX_DF_IMAGE_S32 : false=0, true=-1
- VX_DF_IMAGE_U32 : false=0, true=0xFFFFFFFF
```
CHAPTER 3. MODULE DOCUMENTATION

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the context in which the object is created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>thresh_type</td>
<td>The type of thresholding operation.</td>
</tr>
<tr>
<td>in</td>
<td>input_format</td>
<td>The format of images that will be used as input of the thresholding operation.</td>
</tr>
<tr>
<td>in</td>
<td>output_format</td>
<td>The format of images that will be generated by the thresholding operation.</td>
</tr>
</tbody>
</table>

Returns

A threshold reference vx_threshold. Any possible errors preventing a successful creation should be checked using vxGetStatus.

vx_threshold VX_API_CALL vxCreateVirtualThresholdForImage ( vx_graph graph, vx_enum thresh_type, vx_df_image input_format, vx_df_image output_format )

Creates an opaque reference to a threshold object without direct user access.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the parent graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>thresh_type</td>
<td>The type of thresholding operation.</td>
</tr>
<tr>
<td>in</td>
<td>input_format</td>
<td>The format of images that will be used as input of the thresholding operation.</td>
</tr>
<tr>
<td>in</td>
<td>output_format</td>
<td>The format of images that will be generated by the thresholding operation.</td>
</tr>
</tbody>
</table>

See also

vxCreateThresholdForImage

Returns

A threshold reference vx_threshold. Any possible errors preventing a successful creation should be checked using vxGetStatus.

vx_status VX_API_CALL vxCopyThresholdValue ( vx_threshold thresh, vx_pixel_value_t * value_ptr, vx_enum usage, vx_enum user_mem_type )

Allows the application to copy the thresholding value from/into a threshold object with type VX_THRESHOLD_TYPE→PE_BINARY.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>thresh</th>
<th>The reference to the threshold object that is the source or the destination of the copy.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>value_ptr</td>
<td>The address of the memory location where to store the thresholding value if the copy was requested in read mode, or from where to get the thresholding value to store into the threshold object if the copy was requested in write mode.</td>
</tr>
</tbody>
</table>
### Parameters

**in**  
**usage**  
This declares the effect of the copy with regard to the threshold object using the `vx_accessor_e` enumeration. Only `VX_READ_ONLY` and `VX_WRITE_ONLY` are supported:

- **VX_READ_ONLY** means that the thresholding value is copied from the threshold object into the user memory. After the copy, only the field of the `value_ptr` union that corresponds to the input image format of the threshold object is meaningful.

- **VX_WRITE_ONLY** means the field of the `value_ptr` union corresponding to the input format of the threshold object is copied into the threshold object.

**in**  
**user_mem_type**  
A `vx_memory_type_e` enumeration that specifies the type of the memory referenced by `value_ptr`.

### Returns

A `vx_status_e` enumeration.

### Return values

<table>
<thead>
<tr>
<th><code>vx_status_e</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VX_ERROR_INVALID_REFERENCE</strong></td>
<td>The threshold reference is not actually a threshold reference.</td>
</tr>
<tr>
<td><strong>VX_ERROR_NOT_COMPATIBLE</strong></td>
<td>The threshold object doesn't have type <code>VX_THRESHOLD_TYPE_BINARY</code></td>
</tr>
<tr>
<td><strong>VX_ERROR_INVALID_PARAMETERS</strong></td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

### vx_status VX_API_CALL vxCopyThresholdRange ( vx_threshold thresh, vx_pixel_value_t * lower_value_ptr, vx_pixel_value_t * upper_value_ptr, vx_enum usage, vx_enum user_mem_type )

Allows the application to copy thresholding values from/into a threshold object with type `VX_THRESHOLD_TYPE_BINARY`.

### Parameters

**in**  
**thresh**  
The reference to the threshold object that is the source or the destination of the copy.

**in,out**  
**lower_value_ptr**  
The address of the memory location where to store the lower thresholding value if the copy was requested in read mode, or from where to get the lower thresholding value to store into the threshold object if the copy was requested in write mode.

**in,out**  
**upper_value_ptr**  
The address of the memory location where to store the upper thresholding value if the copy was requested in read mode, or from where to get the upper thresholding value to store into the threshold object if the copy was requested in write mode.
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usage</th>
<th>This declares the effect of the copy with regard to the threshold object using the ( \text{vx_accessor_e} ) enumeration. Only ( \text{VX_READ_ONLY} ) and ( \text{VX_WRITE_ONLY} ) are supported:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{VX_READ_ONLY} ) means that thresholding values are copied from the threshold object into the user memory. After the copy, only the field of ((\ast\text{lower_value_ptr})) and ((\ast\text{upper_value_ptr})) unions that corresponds to the input image format of the threshold object is meaningful.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \text{VX_WRITE_ONLY} ) means the field of the ((\ast\text{lower_value_ptr})) and ((\ast\text{upper_value_ptr})) unions corresponding to the input format of the threshold object is copied into the threshold object.</td>
<td></td>
</tr>
</tbody>
</table>

| in | user_mem_type | A \( \text{vx\_memory\_type\_e} \) enumeration that specifies the type of the memory referenced by \( \text{lower\_value\_ptr} \) and \( \text{upper\_value\_ptr} \). |

### Returns

A \( \text{vx\_status\_e} \) enumeration.

### Return values

- **\( \text{VX\_ERROR\_INVALID\_REFERENCE} \)**: The threshold reference is not actually a threshold reference.
- **\( \text{VX\_ERROR\_NOT\_COMPATIBLE} \)**: The threshold object doesn't have type \( \text{VX\_THRESHOLD\_TYPE\_RANGE} \).
- **\( \text{VX\_ERROR\_INVALID\_PARAMETERS} \)**: An other parameter is incorrect.

### vx\_status VX\_API\_CALL vxCopyThresholdOutput ( \( \text{vx\_threshold} \) \( \text{thresh} \), \( \text{vx\_pixel\_value\_t} \) \( \ast\text{true\_value\_ptr} \), \( \text{vx\_pixel\_value\_t} \) \( \ast\text{false\_value\_ptr} \), \( \text{vx\_enum} \) \( \text{usage} \), \( \text{vx\_enum} \) \( \text{user\_mem\_type} \) )

Allows the application to copy the true and false output values from/into a threshold object.

### Parameters

| in, out | true_value_ptr | The address of the memory location where to store the true output value if the copy was requested in read mode, or from where to get the true output value to store into the threshold object if the copy was requested in write mode. |
| in, out | false_value_ptr | The address of the memory location where to store the false output value if the copy was requested in read mode, or from where to get the false output value to store into the threshold object if the copy was requested in write mode. |
| in     | usage         | This declares the effect of the copy with regard to the threshold object using the \( \text{vx\_accessor\_e} \) enumeration. Only \( \text{VX\_READ\_ONLY} \) and \( \text{VX\_WRITE\_ONLY} \) are supported: |
|        |               | \( \text{VX\_READ\_ONLY} \) means that true and false output values are copied from the threshold object into the user memory. After the copy, only the field of \((\ast\text{true\_value\_ptr})\) and \((\ast\text{false\_value\_ptr})\) unions that corresponds to the output image format of the threshold object is meaningful. |
|        |               | \( \text{VX\_WRITE\_ONLY} \) means the field of the \((\ast\text{true\_value\_ptr})\) and \((\ast\text{false\_value\_ptr})\) unions corresponding to the output format of the threshold object is copied into the threshold object. |
Parameters

| in | user_mem_type | A vx_memory_type_e enumeration that specifies the type of the memory referenced by true_value_ptr and false_value_ptr. |

Returns

A vx_status_e enumeration.

Return values

| VX_ERROR_INVALID_REFERENCE | The threshold reference is not actually a threshold reference. |
| VX_ERROR_INVALID_PARAMETERS | An other parameter is incorrect. |

vx_status VX_API_CALL vxReleaseThreshold ( vx_threshold * thresh )

Releases a reference to a threshold object. The object may not be garbage collected until its total reference count is zero.

Parameters

| in | thresh | The pointer to the threshold to release. |

Postcondition

After returning from this function the reference is zeroed.

Returns

A vx_status_e enumeration.

Return values

| VX_SUCCESS | No errors; any other value indicates failure. |
| VX_ERROR_INVALID_REFERENCE | thresh is not a valid vx_threshold reference. |

vx_status VX_API_CALL vxSetThresholdAttribute ( vx_threshold thresh, vx_enum attribute, const void * ptr, vx_size size )

Sets attributes on the threshold object.

Parameters

| in | thresh | The threshold object to set. |
| in | attribute | The attribute to modify. Use a vx_threshold_attribute_e enumeration. |
| in | ptr | The pointer to the value to which to set the attribute. |
| in | size | The size of the data pointed to by ptr. |

Returns

A vx_status_e enumeration.
Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>thresh is not a valid vx_threshold reference.</td>
</tr>
</tbody>
</table>

vx_status VX_API_CALL vxQueryThreshold ( vx_threshold thresh, vx_enum attribute, void * ptr, vx_size size )

Queries an attribute on the threshold object.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>thresh</th>
<th>The threshold object to set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>attribute</td>
<td>The attribute to query. Use a vx_threshold_attribute_e enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The location at which to store the resulting value.</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>The size of the container to which ptr points.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>thresh is not a valid vx_threshold reference.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.78 Object: ObjectArray

3.78.1 Detailed Description

An opaque array object that could be an array of any data-object (not data-type) of OpenVX except Delay and ObjectArray objects.

ObjectArray is a strongly-typed container of OpenVX data-objects. ObjectArray refers to the collection of similar data-objects as a single entity that can be created or assigned as inputs/outputs and as a single entity. In addition, a single object from the collection can be accessed individually by getting its reference. The single object remains as part of the ObjectArray through its entire life cycle.

Typedefs

- typedef struct _vx_object_array ∗vx_object_array

  The ObjectArray Object. ObjectArray is a strongly-typed container of OpenVX data-objects.

Enumerations

- enum vx_object_array_attribute_e {
  VX_OBJECT_ARRAY_ITEMTYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_OBJECT_ARRAY << 8)) + 0x0,
  VX_OBJECT_ARRAY_NUMITEMS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_OBJECT_ARRAY << 8)) + 0x1
  }

  The ObjectArray object attributes.

Functions

- vx_object_array VX_API_CALL vxCreateObjectArray (vx_context context, vx_reference exemplar, vx_size count)

  Creates a reference to an ObjectArray of count objects.

- vx_object_array VX_API_CALL vxCreateVirtualObjectArray (vx_graph graph, vx_reference exemplar, vx← size count)

  Creates an opaque reference to a virtual ObjectArray with no direct user access.

- vx_reference VX_API_CALL vxGetObjectArrayItem (vx_object_array arr, vx_uint32 index)

  Retrieves the reference to the OpenVX Object in location index of the ObjectArray.

- vx_status VX_API_CALL vxQueryObjectArray (vx_object_array arr, vx_enum attribute, void ∗ptr, vx_size size)

  Queries an attribute from the ObjectArray.

- vx_status VX_API_CALL vxReleaseObjectArray (vx_object_array ∗arr)

  Releases a reference of an ObjectArray object.

3.78.2 Enumeration Type Documentation

enum vx_object_array_attribute_e

The ObjectArray object attributes.

Enumerator

- VX_OBJECT_ARRAY_ITEMTYPE The type of the ObjectArray items. Read-only. Use a vx_enum parameter.

- VX_OBJECT_ARRAY_NUMITEMS The number of items in the ObjectArray. Read-only. Use a vx_size parameter.

Definition at line 1159 of file vx_types.h.
**3.78.3 Function Documentation**

**vx_object_array VX_API_CALL vxCreateObjectArray ( vx_context context, vx_reference exemplar, vx_size count )**

Creates a reference to an ObjectArray of count objects.

It uses the metadata of the exemplar to determine the object attributes, ignoring the object data. It does not alter the exemplar or keep or release the reference to the exemplar. For the definition of supported attributes see `vxSetMetaFormatAttribute`. In case the exemplar is a virtual object it must be of immutable metadata, thus it is not allowed to be dimensionless or formatless.

**Parameters**

- **context**: The reference to the overall Context.
- **exemplar**: The exemplar object that defines the metadata of the created objects in the ObjectArray.
- **count**: Number of Objects to create in the ObjectArray. This value must be greater than zero.

**Returns**

An ObjectArray reference `vx_object_array`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`. Data objects are not initialized by this function.

**vx_object_array VX_API_CALL vxCreateVirtualObjectArray ( vx_graph graph, vx_reference exemplar, vx_size count )**

Creates an opaque reference to a virtual ObjectArray with no direct user access.

This function creates an ObjectArray of count objects with similar behavior as `vxCreateObjectArray`. The only difference is that the objects that are created are virtual in the given graph.

**Parameters**

- **graph**: Reference to the graph where to create the virtual ObjectArray.
- **exemplar**: The exemplar object that defines the type of object in the ObjectArray. Only exemplar type of `vx_image`, `vx_array` and `vx_pyramid` are allowed.
- **count**: Number of Objects to create in the ObjectArray.

**Returns**

A ObjectArray reference `vx_object_array`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx_reference VX_API_CALL vxGetObjectArrayItem ( vx_object_array arr, vx_uint32 index )**

Retrieves the reference to the OpenVX Object in location index of the ObjectArray.

This is a `vx_reference`, which can be used elsewhere in OpenVX. A call to `vxRelease<Object>` or `vxReleaseReference` is necessary to release the Object for each call to this function.

**Parameters**

- **arr**: The ObjectArray.
- **index**: The index of the object in the ObjectArray.

**Returns**

A reference to an OpenVX data object. Any possible errors preventing a successful completion of the function should be checked using `vxGetStatus`. 
vxsstatus VX_API_CALL vxReleaseObjectArray ( vx_object_array * arr )

Releases a reference of an ObjectArray object.

The object may not be garbage collected until its total reference and its contained objects count is zero. After returning from this function the reference is zeroed/cleared.

Parameters

| in  | arr | The pointer to the ObjectArray to release. |

Returns

A vxsstatus e enumeration.

Return values

<table>
<thead>
<tr>
<th>vxsstatus</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>arr is not a valid vx_object_array reference.</td>
</tr>
</tbody>
</table>

vxsstatus VX_API_CALL vxQueryObjectArray ( vx_object_array arr, vx_enum attribute, void * ptr, vx_size size )

Queries an attribute from the ObjectArray.

Parameters

| in  | arr | The reference to the ObjectArray. |
| in  | attribute | The attribute to query. Use a vx_object_array_attribute_e. |
| out | ptr | The location at which to store the resulting value. |
| in  | size | The size in bytes of the container to which ptr points. |

Returns

A vxsstatus e enumeration.

Return values

<table>
<thead>
<tr>
<th>vxsstatus</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>arr is not a valid vx_object_array reference.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUPPORTED</td>
<td>If the attribute is not a value supported on this implementation.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>If any of the other parameters are incorrect.</td>
</tr>
</tbody>
</table>
3.79 Object: Tensor

3.79.1 Detailed Description

Defines The Tensor Object Interface.

The `vx_tensor` object represents an opaque multidimensional array. The object is said to be opaque because the programmer has no visibility into the internal implementation of the object, and can only manipulate them via the defined API. Implementations can apply many optimizations that are transparent to the user. OpenVX implementations must support `vx_tensor` objects of at least 4 dimensions, although a vendor can choose to support more dimensions in his implementation. The maximum number of dimensions supported by a given implementation can be queried via the context attribute `VX_CONTEXT_MAX_TENSOR_DIMS`. Implementations must support tensors from one dimension (i.e., vectors) through `VX_CONTEXT_MAX_TENSOR_DIMS`, inclusive. The individual elements of the tensor object may be any numerical data type. For each kernel in the specification, it is specified which data types a compliant implementations must support. Integer elements can represent fractional values by assigning a non-zero radix point. As an example: `VX_TYPE_INT16` element with radix point of 8, corresponds to Q7.8 signed fixed-point in “Q” notation. A vendor may choose to support whatever values for the radix point in his implementation. Since functions using tensors, need to understand the context of each dimension. We describe a layout of the dimensions in each function. That layout is not mandated. It is done specifically to explain the functions and not to mandate layout. Different implementation may have different layout. Therefore the layout description is logical and not physical. It refers to the order of dimensions given in `vxCreateTensor` and `vxCreateVirtualTensor`.

**Typedefs**

- typedef struct _vx_tensor_t *vx_tensor
  
  The multidimensional data object (Tensor).

**Enumerations**

- enum vx_tensor_attribute_e {
  
  `VX_TENSOR_NUMBER_OF_DIMS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_TENSOR << 8)) + 0x0,
  VX_TENSOR_DIMS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_TENSOR << 8)) + 0x1,
  VX_TENSOR_DATA_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_TENSOR << 8)) + 0x2,
  VX_TENSOR_FIXED_POINT_POSITION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_TENSOR << 8)) + 0x3
  }

  tensor Data attributes.

**Functions**

- vx_status VX_API_CALL vxCopyTensorPatch (vx_tensor tensor, vx_size number_of_dims, const vx_size *view_start, const vx_size *view_end, const vx_size *user_stride, void *user_ptr, vx_enum usage, vx_enum user_memory_type)
  
  Allows the application to copy a view patch from/into an tensor object.

- vx_object_array VX_API_CALL vxCreateImageObjectArrayFromTensor (vx_tensor tensor, const vx_rectangle_t *rect, vx_size array_size, vx_size jump, vx_df_image image_format)
  
  Creates an array of images into the multi-dimension data, this can be adjacent 2D images or not depending on the stride value. The stride value is representing bytes in the third dimension. The OpenVX image object that points to a three dimension data and access it as an array of images. This has to be portion of the third lowest dimension, and the stride correspond to that third dimension. The returned Object array is an array of images. Where the image data is pointing to a specific memory in the input tensor.

- vx_tensor VX_API_CALL vxCreateTensor (vx_context context, vx_size number_of_dims, const vx_size *dims, vx_enum data_type, vx_int8 fixed_point_position)
  
  Creates an opaque reference to a tensor data buffer.

- vx_tensor VX_API_CALL vxCreateTensorFromView (vx_tensor tensor, vx_size number_of_dims, const vx_size *view_start, const vx_size *view_end)
  
  Creates a tensor data from another tensor data given a view. This second reference refers to the data in the original tensor data. Updates to this tensor data updates the parent tensor data. The view must be defined within the dimensions of the parent tensor data.
• vx_tensor VX_API_CALL vxCreateVirtualTensor (vx_graph graph, vx_size number_of_dims, const vx_size *dims, vx_enum data_type, vx_int8 fixed_point_position)
  Creates an opaque reference to a tensor data buffer with no direct user access. This function allows setting the tensor data dimensions or data format.

• vx_status VX_API_CALL vxQueryTensor (vx_tensor tensor, vx_enum attribute, void *ptr, vx_size size)
  Retrieves various attributes of a tensor data.

• vx_status VX_API_CALL vxReleaseTensor (vx_tensor *tensor)
  Releases a reference to a tensor data object. The object may not be garbage collected until its total reference count is zero.

### 3.79.2 Typedef Documentation

typedef struct _vx_tensor_t vx_tensor
The multidimensional data object (Tensor).

See also
vxCreateTensor

Definition at line 284 of file vx_types.h.

### 3.79.3 Enumeration Type Documentation

enum vx_tensor_attribute_e

tensor Data attributes.

Enumerator

  VX_TENSOR_NUMBER_OF_DIMS  Number of dimensions.
  VX_TENSOR_DIMS  Dimension sizes.
  VX_TENSOR_DATA_TYPE  tensor Data element data type. vx_type_e
  VX_TENSOR_FIXED_POINT_POSITION  fixed point position when the input element type is integer.

Definition at line 1168 of file vx_types.h.

### 3.79.4 Function Documentation

vx_tensor VX_API_CALL vxCreateTensor ( vx_context context, vx_size number_of_dims, const vx_size *dims, vx_enum data_type, vx_int8 fixed_point_position )

Creates an opaque reference to a tensor data buffer.

Not guaranteed to exist until the vx_graph containing it has been verified. Since functions using tensors, need to understand the context of each dimension. We describe a layout of the dimensions in each function using tensors. That layout is not mandatory. It is done specifically to explain the functions and not to mandate layout. Different implementation may have different layout. Therefore the layout description is logical and not physical. It refers to the order of dimensions given in this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>The reference to the implementation context.</td>
</tr>
<tr>
<td>number_of_dims</td>
<td>The number of dimensions.</td>
</tr>
<tr>
<td>dims</td>
<td>Dimensions sizes in elements.</td>
</tr>
<tr>
<td>data_type</td>
<td>The vx_type_e that represents the data type of the tensor data elements.</td>
</tr>
<tr>
<td>fixed_point_position</td>
<td>Specifies the fixed point position when the input element type is integer. if 0, calculations are performed in integer math.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

Returns

A tensor data reference. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

`vx_object_array VX_API_CALL vxCreateImageObjectFromArrayFromTensor ( vx_tensor tensor, const vx_rectangle_t * rect, vx_size array_size, vx_size jump, vx_df_image image_format )`

Creates an array of images into the multi-dimension data, this can be adjacent 2D images or not depending on the stride value. The stride value is representing bytes in the third dimension. The OpenVX image object that points to a three dimension data and access it as an array of images. This has to be portion of the third lowest dimension, and the stride correspond to that third dimension. The returned Object array is an array of images. Where the image data is pointing to a specific memory in the input tensor.

Parameters

| in  | tensor       | The tensor data from which to extract the images. Has to be a 3d tensor. |
| in  | rect         | Image coordinates within tensor data.                                      |
| in  | array_size   | Number of images to extract.                                               |
| in  | jump         | Delta between two images in the array.                                     |
| in  | image_format | The requested image format. Should match the tensor data’s data type.       |

Returns

An array of images pointing to the tensor data’s data.

`vx_tensor VX_API_CALL vxCreateTensorFromView ( vx_tensor tensor, vx_size number_of_dims, const vx_size * view_start, const vx_size * view_end )`

Creates a tensor data from another tensor data given a view. This second reference refers to the data in the original tensor data. Updates to this tensor data updates the parent tensor data. The view must be defined within the dimensions of the parent tensor data.

Parameters

| in        | tensor       | The reference to the parent tensor data.                                  |
| in        | number_of_dims | Number of dimensions in the view. Error return if 0 or greater than number of tensor dimensions. If smaller than number of tensor dimensions, the lower dimensions are assumed. |
| in        | view_start   | View start coordinates                                                    |
| in        | view_end     | View end coordinates                                                      |

Returns

The reference to the sub-tensor. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

`vx_tensor VX_API_CALL vxCreateVirtualTensor ( vx_graph graph, vx_size number_of_dims, const vx_size * dims, vx_enum data_type, vx_int8 fixed_point_position )`

Creates an opaque reference to a tensor data buffer with no direct user access. This function allows setting the tensor data dimensions or data format.

Virtual data objects allow users to connect various nodes within a graph via data references without access to that data, but they also permit the implementation to take maximum advantage of possible optimizations. Use this API to create a data reference to link two or more nodes together when the intermediate data are not required to be accessed by outside entities. This API in particular allows the user to define the tensor data format of the
data without requiring the exact dimensions. Virtual objects are scoped within the graph they are declared a part of, and can’t be shared outside of this scope. Since functions using tensors, need to understand the context of each dimension. We describe a layout of the dimensions in each function. That layout is not mandated. It is done specifically to explain the functions and not to mandate layout. Different implementation may have different layout. Therefore the layout description is logical and not physical. It refers to the order of dimensions given in vxCreateTensor and vxCreateVirtualTensor.

Parameters

| in | graph | The reference to the parent graph. |
| in | number_of_dims | The number of dimensions. |
| in | dims | Dimensions sizes in elements. |
| in | data_type | The vx_type_e that represents the data type of the tensor data elements. |
| in | fixed_point_position | Specifies the fixed point position when the input element type is integer. If 0, calculations are performed in integer math. |

Returns

A tensor data reference. Any possible errors preventing a successful creation should be checked using vxGetStatus.

Note

Passing this reference to vxCopyTensorPatch will return an error.

vx_status VX_API_CALL vxCopyTensorPatch ( vx_tensor tensor, vx_size number_of_dims, const vx_size *view_start, const vx_size *view_end, const vx_size *user_stride, void *user_ptr, vx_enum usage, vx_enum user_memory_type )

Allows the application to copy a view patch from/into an tensor object.

Parameters

| in | tensor | The reference to the tensor object that is the source or the destination of the copy. |
| in | number_of_dims | Number of patch dimension. Error return if 0 or greater than number of tensor dimensions. If smaller than number of tensor dimensions, the lower dimensions are assumed. |
| in | view_start | Array of patch start points in each dimension |
| in | view_end | Array of patch end points in each dimension |
| in | user_stride | Array of user memory strides in each dimension |
| in | user_ptr | The address of the memory location where to store the requested data if the copy was requested in read mode, or from where to get the data to store into the tensor object if the copy was requested in write mode. The accessible memory must be large enough to contain the specified patch with the specified layout: accessible memory in bytes \( \geq (end[\text{last dimension}] - start[\text{last dimension}]) \times stride[\text{last dimension}] \). The layout of the user memory must follow a row major order. |
| in | usage | This declares the effect of the copy with regard to the tensor object using the vx_accessor_e enumeration. Only VX_READ_ONLY and VX_WRITE ONLY are supported: |
| | | • VX_READ ONLY means that data is copied from the tensor object into the application memory |
| | | • VX_WRITE ONLY means that data is copied into the tensor object from the application memory |
| in | user_memory_type | A vx_memory_type_e enumeration that specifies the memory type of the memory referenced by the user_addr. |
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th><code>VX_ERROR_OPTIMIZED_AWAY</code></th>
<th>This is a reference to a virtual tensor that cannot be accessed by the application.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>VX_ERROR_INVALID_REFERENCE</code></td>
<td>The tensor reference is not actually a tensor reference.</td>
</tr>
<tr>
<td><code>VX_ERROR_INVALID_PARAMETERS</code></td>
<td>An other parameter is incorrect.</td>
</tr>
</tbody>
</table>

`vx_status VX_API_CALL vxQueryTensor ( vx_tensor tensor, vx_enum attribute, void * ptr, vx_size size )`

Retrieves various attributes of a tensor data.

Parameters

- `in` `tensor` The reference to the tensor data to query.
- `in` `attribute` The attribute to query. Use a `vx_tensor_attribute_e`.
- `out` `ptr` The location at which to store the resulting value.
- `in` `size` The size of the container to which `ptr` points.

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th><code>VX_SUCCESS</code></th>
<th>No errors.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>VX_ERROR_INVALID_REFERENCE</code></td>
<td>If data is not a <code>vx_tensor</code>.</td>
</tr>
<tr>
<td><code>VX_ERROR_INVALID_PARAMETERS</code></td>
<td>If any of the other parameters are incorrect.</td>
</tr>
</tbody>
</table>

`vx_status VX_API_CALL vxReleaseTensor ( vx_tensor * tensor )`

Releases a reference to a tensor data object. The object may not be garbage collected until its total reference count is zero.

Parameters

- `in` `tensor` The pointer to the tensor data to release.

Postcondition

After returning from this function the reference is zeroed.

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th><code>VX_SUCCESS</code></th>
<th>No errors; all other values indicate failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.80 Administrative Features

3.80.1 Detailed Description

Defines the Administrative Features of OpenVX. These features are administrative in nature and require more understanding and are more complex to use.

Modules

- **Advanced Objects**
  
  Defines the Advanced Objects of OpenVX.

- **Advanced Framework API**
  
  Describes components that are considered to be advanced.
3.81 Advanced Objects

3.81.1 Detailed Description
Defines the Advanced Objects of OpenVX.

Modules

- **Object: Array (Advanced)**
  Defines the advanced features of the Array Interface.

- **Object: Node (Advanced)**
  Defines the advanced features of the Node Interface.

- **Object: Delay**
  Defines the Delay Object interface.

- **Object: Kernel**
  Defines the Kernel Object and Interface.

- **Object: Parameter**
  Defines the Parameter Object interface.
3.82 Object: Array (Advanced)

3.82.1 Detailed Description
Defines the advanced features of the Array Interface.

Functions

- `vx_enum VX_API_CALL vxRegisterUserStruct (vx_context context, vx_size size)`
  Registers user-defined structures to the context.

3.82.2 Function Documentation

`vx_enum VX_API_CALL vxRegisterUserStruct ( vx_context context, vx_size size )`

Registers user-defined structures to the context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the implementation context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>The size of user struct in bytes.</td>
</tr>
</tbody>
</table>

Returns

A `vx_enum` value that is a type given to the User to refer to their custom structure when declaring a `vx_<-array` of that structure.

Return values

- `VX_TYPE_INVALID` If the namespace of types has been exhausted.

Note

This call should only be used once within the lifetime of a context for a specific structure.
3.83 Object: Node (Advanced)

3.83.1 Detailed Description
Defines the advanced features of the Node Interface.

Modules

- **Node: Border Modes**
  
  Defines the border mode behaviors.

Functions

- **vx_node VX_API_CALL vxCreateGenericNode** (vx_graph graph, vx_kernel kernel)
  
  Creates a reference to a node object for a given kernel.

3.83.2 Function Documentation

vx_node VX_API_CALL vxCreateGenericNode ( vx_graph graph, vx_kernel kernel )

Creates a reference to a node object for a given kernel.

This node has no references assigned as parameters after completion. The client is then required to set these parameters manually by vxSetParameterByIndex. When clients supply their own node creation functions (for use with User Kernels), this is the API to use along with the parameter setting API.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph in which this node exists.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>kernel</td>
<td>The kernel reference to associate with this new node.</td>
</tr>
</tbody>
</table>

Returns

A node reference vx_node. Any possible errors preventing a successful creation should be checked using vxGetStatus.

Note

A call to this API sets all parameters to NULL.

Postcondition

Call vxSetParameterByIndex for as many parameters as needed to be set.
### 3.84 Node: Border Modes

#### 3.84.1 Detailed Description

Defines the border mode behaviors.

Border Mode behavior is set as an attribute of the node, not as a direct parameter to the kernel. This allows clients to set-and-forget the modes of any particular node that supports border modes. All nodes shall support VX_BORDER_UNDEFINED.

#### Data Structures

- **struct vx_border_t**
  
  Use with the enumeration VX_NODE_BORDER to set the border mode behavior of a node that supports borders.

**More...**

#### Enumerations

- **enum vx_border_e {**
  
  VX_BORDER_UNDEFINED = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_BORDER << 12)) + 0x0,
  
  VX_BORDER_CONSTANT = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_BORDER << 12)) + 0x1,
  
  VX_BORDER_REPLICATE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_BORDER << 12)) + 0x2 }  

  The border mode list.

- **enum vx_border_policy_e {**
  
  VX_BORDER_POLICY_DEFAULT_TO_UNDEFINED = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_BORDER_POLICY << 12)) + 0x0,
  
  VX_BORDER_POLICY_RETURN_ERROR = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_BORDER_POLICY << 12)) + 0x1 }  

  The unsupported border mode policy list.

#### 3.84.2 Data Structure Documentation

**struct vx_border_t**

Use with the enumeration VX_NODE_BORDER to set the border mode behavior of a node that supports borders.

If the indicated border mode is not supported, an error VX_ERROR_NOT_SUPPORTED will be reported either at the time the VX_NODE_BORDER is set or at the time of graph verification.

**Definition at line 1729 of file vx_types.h.**

**Data Fields**

<table>
<thead>
<tr>
<th>vx_enum</th>
<th>mode</th>
<th>See vx_border_e.</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_pixel_← value_t</td>
<td>constant_value</td>
<td>For the mode VX_BORDER_CONSTANT, this union contains the value of out-of-bound pixels.</td>
</tr>
</tbody>
</table>

#### 3.84.3 Enumeration Type Documentation

**enum vx_border_e**

The border mode list.

**Enumerator**

- **VX_BORDER_UNDEFINED** No defined border mode behavior is given.
- **VX_BORDER_CONSTANT** For nodes that support this behavior, a constant value is filled-in when accessing out-of-bounds pixels.
- **VX_BORDER_REPLICATE** For nodes that support this behavior, a replication of the nearest edge pixels value is given for out-of-bounds pixels.

**Definition at line 1336 of file vx_types.h.**
enum vx_border_policy_e

The unsupported border mode policy list.

Enumerator

- **VX_BORDER_POLICY_DEFAULT_TO_UNDEFINED** Use VX BORDER_UNDEFINED instead of unsupported border modes.
- **VX_BORDER_POLICY_RETURN_ERROR** Return VX_ERROR_NOT_SUPPORTED for unsupported border modes.

Definition at line 1352 of file vx_types.h.
3.85 Object: Delay

3.85.1 Detailed Description

Defines the Delay Object interface.

A Delay is an opaque object that contains a manually-controlled, temporally-delayed list of objects. A Delay cannot be an output of a kernel. Also, aging of a Delay (see vxAgeDelay) cannot be performed during graph execution. Supported delay object types include:

- VX_TYPE_ARRAY,
- VX_TYPE_IMAGE,
- VX_TYPE_PYRAMID,
- VX_TYPE_MATRIX,
- VX_TYPE_CONVOLUTION,
- VX_TYPE_DISTRIBUTION,
- VX_TYPE_REMAP,
- VX_TYPE_LUT,
- VX_TYPE_THRESHOLD,
- VX_TYPE_SCALAR

Typedefs

- typedef struct _vx_delay * vx_delay
  
  The delay object. This is like a ring buffer of objects that is maintained by the OpenVX implementation.

Enumerations

- enum vx_delay_attribute_e {
  
  VX_DELAY_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DELAY << 8)) + 0x0,
  VX_DELAY_SLOTS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DELAY << 8)) + 0x1
  
  The delay attribute list.
}

Functions

- vx_status VX_API_CALL vxAgeDelay (vx_delay delay)
  
  Shifts the internal delay ring by one.

- vx_delay VX_API_CALL vxCreateDelay (vx_context context, vx_reference exemplar, vx_size num_slots)
  
  Creates a Delay object.

- vx_reference VX_API_CALL vxGetReferenceFromDelay (vx_delay delay, vx_int32 index)
  
  Retrieves a reference to a delay slot object.

- vx_status VX_API_CALL vxQueryDelay (vx_delay delay, vx_enum attribute, void *ptr, vx_size size)
  
  Queries a vx_delay object attribute.

- vx_status VX_API_CALL vxReleaseDelay (vx_delay *delay)
  
  Releases a reference to a delay object. The object may not be garbage collected until its total reference count is zero.

3.85.2 Typedef Documentation

typedef struct _vx_delay* vx_delay

The delay object. This is like a ring buffer of objects that is maintained by the OpenVX implementation.

See also

vxCreateDelay

Definition at line 223 of file vx_types.h.
3.85.3 Enumeration Type Documentation

enum vx_delay_attribute_e

The delay attribute list.

Enumerator

\texttt{VX\_DELAY\_TYPE} \hspace{1em} The type of objects in the delay. Read-only. Use a \texttt{vx\_enum} parameter.

\texttt{VX\_DELAY\_SLOTS} \hspace{1em} The number of items in the delay. Read-only. Use a \texttt{vx\_size} parameter.

Definition at line 1388 of file \texttt{vx\_types\_h}.

3.85.4 Function Documentation

\texttt{vx\_status VX\_API\_CALL vxQueryDelay ( vx\_delay \ delay\ , \ vx\_enum \ attribute\ , \ void \* \ ptr\ , \ vx\_size \ size\ )}

Queries a \texttt{vx\_delay} object attribute.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>delay</th>
<th>The reference to a delay object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>attribute</td>
<td>The attribute to query. Use a \texttt{vx_delay_attribute_e} enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The location at which to store the resulting value.</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>The size of the container to which \texttt{ptr} points.</td>
</tr>
</tbody>
</table>

Returns

A \texttt{vx\_status\_e} enumeration.

Return values

<table>
<thead>
<tr>
<th>\texttt{VX_SUCCESS}</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{VX_ERROR_INVALID_REFERENCE}</td>
<td>\texttt{delay} is not a valid \texttt{vx_delay} reference.</td>
</tr>
</tbody>
</table>

\texttt{vx\_status VX\_API\_CALL vxReleaseDelay ( vx\_delay \* \ delay\ )}

Releases a reference to a delay object. The object may not be garbage collected until its total reference count is zero.

Parameters

| in   | delay | The pointer to the delay object reference to release. |

Postcondition

After returning from this function the reference is zeroed.

Returns

A \texttt{vx\_status\_e} enumeration.

Return values

<table>
<thead>
<tr>
<th>\texttt{VX_SUCCESS}</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{VX_ERROR_INVALID_REFERENCE}</td>
<td>\texttt{delay} is not a valid \texttt{vx_delay} reference.</td>
</tr>
</tbody>
</table>
vx_delay VX_API_CALL vxCreateDelay ( vx_context context, vx_reference exemplar, vx_size num_slots )

Creates a Delay object.

This function creates a delay object with \texttt{num_slots} slots. Each slot contains a clone of the exemplar. The clones only inherit the metadata of the exemplar. The data content of the exemplar is ignored and the clones have their data undefined at delay creation time. The function does not alter the exemplar. Also, it doesn't retain or release the reference to the exemplar.

\textbf{Note}

For the definition of metadata attributes see \texttt{vxSetMetaFormatAttribute}.

\begin{tabular}{|c|c|}
\hline
\textbf{Parameters} & \\
\hline
\texttt{in context} & The reference to the context. \\
\hline
\texttt{in exemplar} & The exemplar object. Supported exemplar object types are: \\
\hline
\texttt{in num_slots} & The number of objects in the delay. This value must be greater than zero. \\
\hline
\end{tabular}

\textbf{Returns}

A delay reference \texttt{vx_delay}. Any possible errors preventing a successful creation should be checked using \texttt{vxGetStatus}.

vx_reference VX_API_CALL vxGetReferenceFromDelay ( vx_delay delay, vx_int32 index )

Retrieves a reference to a delay slot object.

\begin{tabular}{|c|c|}
\hline
\textbf{Parameters} & \\
\hline
\texttt{in delay} & The reference to the delay object. \\
\hline
\texttt{in index} & The index of the delay slot from which to extract the object reference. \\
\hline
\end{tabular}

\textbf{Returns}

\texttt{vx_reference}. Any possible errors preventing a successful completion of the function should be checked using \texttt{vxGetStatus}.
Note

The delay index is in the range \([-\text{count} + 1, 0]\). 0 is always the current object.
A reference retrieved with this function must not be given to its associated release API (e.g. \texttt{vxReleaseImage}) unless \texttt{vxRetainReference} is used.

\begin{verbatim}
vx_status VX_API_CALL vxAgeDelay ( vx_delay delay )
Shifts the internal delay ring by one.
This function performs a shift of the internal delay ring by one. This means that, the data originally at index 0 move to index -1 and so forth until index \(-\text{count} + 1\). The data originally at index \(-\text{count} + 1\) move to index 0. Here \texttt{count} is the number of slots in delay ring. When a delay is aged, any graph making use of this delay (delay object itself or data objects in delay slots) gets its data automatically updated accordingly.

Parameters
\begin{verbatim}
in delay
\end{verbatim}

Returns

\begin{verbatim}
A \texttt{vx_status_e} enumeration.
\end{verbatim}

Return values
\begin{verbatim}
\begin{tabular}{|c|p{10cm}|}
\hline
\texttt{VX_SUCCESS} & Delay was aged; any other value indicates failure. \\
\texttt{VX_ERROR_INVALID_REFERENCE} & delay is not a valid \texttt{vx_delay} reference. \\
\hline
\end{tabular}
\end{verbatim}
3.86 Object: Kernel

3.86.1 Detailed Description

Defines the Kernel Object and Interface.

A Kernel in OpenVX is the abstract representation of an computer vision function, such as a “Sobel Gradient” or “Lucas Kanade Feature Tracking”. A vision function may implement many similar or identical features from other functions, but it is still considered a single unique kernel as long as it is named by the same string and enumeration and conforms to the results specified by OpenVX. Kernels are similar to function signatures in this regard.

In each of the cases, a client of OpenVX could request the kernels in nearly the same manner. There are two main approaches, which depend on the method a client calls to get the kernel reference. The first uses enumerations.

```c
vx_kernel kernel = vxGetKernelByEnum(context,
    VX_KERNEL_SOBEL_3x3);
vx_node node = vxCreateGenericNode(graph, kernel);
```

The second method depends on using strings to get the kernel reference.

```c
vx_kernel kernel = vxGetKernelByName(context, "org.khronos.openvx.sobel_3x3");
vx_node node = vxCreateGenericNode(graph, kernel);
```

Data Structures

- struct vx_kernel_info_t

The Kernel Information Structure. This is returned by the Context to indicate which kernels are available in the OpenVX implementation. More...

Macros

- #define VX_MAX_KERNEL_NAME (256)

Defines the length of a kernel name string to be added to OpenVX, including the trailing zero.

Typedefs

- typedef struct _vx_kernel_vx_kernel

An opaque reference to the descriptor of a kernel.

Enumerations

- enum vx_kernel_attribute_e {
  VX_KERNEL_PARAMETERS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_KERNEL << 8)) + 0x0,
  VX_KERNEL_NAME = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_KERNEL << 8)) + 0x1,
  VX_KERNEL_ENUM = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_KERNEL << 8)) + 0x2,
  VX_KERNEL_LOCAL_DATA_SIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_KERNEL << 8)) + 0x3
}

The kernel attributes list.

- enum vx_kernel_e {
  VX_KERNEL_COLOR_CONVERT = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x1,
  VX_KERNEL_CHANNEL_EXTRACT = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x2,
  VX_KERNEL_CHANNEL_COMBINE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x3,
  VX_KERNEL_SOBEL_3x3 = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x4,
  VX_KERNEL_MAGNITUDE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x5,
  VX_KERNEL_PHASE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x6,
  VX_KERNEL_SCALE_IMAGE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x7,
  VX_KERNEL_TABLE_LOOKUP = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) +
0x8,
\texttt{VX\_KERNEL\_HISTOGRAM} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x9,
\texttt{VX\_KERNEL\_EQUALIZE\_HISTOGRAM} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0xA,
\texttt{VX\_KERNEL\_ABSDIFF} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0xB,
\texttt{VX\_KERNEL\_MEAN\_STDDEV} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0xC,
\texttt{VX\_KERNEL\_THRESHOLD} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0xD,
\texttt{VX\_KERNEL\_INTEGRAL\_IMAGE} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0xE,
\texttt{VX\_KERNEL\_DILATE\_3x3} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0xF,
\texttt{VX\_KERNEL\_ERODE\_3x3} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x10,
\texttt{VX\_KERNEL\_MEDIAN\_3x3} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x11,
\texttt{VX\_KERNEL\_BOX\_3x3} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x12,
\texttt{VX\_KERNEL\_GAUSSIAN\_3x3} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x13,
\texttt{VX\_KERNEL\_CUSTOM\_CONVOLUTION} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x14,
\texttt{VX\_KERNEL\_GAUSSIAN\_PYRAMID} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x15,
\texttt{VX\_KERNEL\_ACCUMULATE} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x16,
\texttt{VX\_KERNEL\_ACCUMULATE\_WEIGHTED} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x17,
\texttt{VX\_KERNEL\_ACCUMULATE\_SQUARE} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x18,
\texttt{VX\_KERNEL\_MINMAXLOC} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x19,
\texttt{VX\_KERNEL\_CONVERTDEPTH} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x1A,
\texttt{VX\_KERNEL\_CANNY\_EDGE\_DETECTOR} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x1B,
\texttt{VX\_KERNEL\_AND} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x1C,
\texttt{VX\_KERNEL\_OR} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x1D,
\texttt{VX\_KERNEL\_XOR} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x1E,
\texttt{VX\_KERNEL\_NOT} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x1F,
\texttt{VX\_KERNEL\_MULTIPLY} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x20,
\texttt{VX\_KERNEL\_ADD} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x21,
\texttt{VX\_KERNEL\_SUBTRACT} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x22,
\texttt{VX\_KERNEL\_WARP\_AFFINE} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x23,
\texttt{VX\_KERNEL\_WARP\_PERSPECTIVE} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x24,
\texttt{VX\_KERNEL\_HARRIS\_CORNERS} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x25,
\texttt{VX\_KERNEL\_FAST\_CORNERS} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x26,
\texttt{VX\_KERNEL\_OPTICAL\_FLOW\_PYR\_LK} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x27,
\texttt{VX\_KERNEL\_REMAP} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x28,
\texttt{VX\_KERNEL\_HALFSCALE\_GAUSSIAN} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x29,
\texttt{VX\_KERNEL\_MAX\_1\_0} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x2A,
\texttt{VX\_KERNEL\_LAPLACIAN\_PYRAMID} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x2B,
\texttt{VX\_KERNEL\_LAPLACIAN\_RECONSTRUCT} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x2C,
\texttt{VX\_KERNEL\_NON\_LINEAR\_FILTER} = \texttt{VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE)} + 0x2D,
ASE) + 0x2C,
\texttt{VX\_KERNEL\_MAX\_1\_1},
\texttt{VX\_KERNEL\_MATCH\_TEMPLATE} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x2D,
\texttt{VX\_KERNEL\_LBP} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x2E,
\texttt{VX\_KERNEL\_HOUGH\_LINES\_P} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x2F,
\texttt{VX\_KERNEL\_TENSOR\_MULTIPLY} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x30,
\texttt{VX\_KERNEL\_TENSOR\_ADD} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x31,
\texttt{VX\_KERNEL\_TENSOR\_SUBTRACT} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x32,
\texttt{VX\_KERNEL\_TENSOR\_TABLE\_LOOKUP} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x33,
\texttt{VX\_KERNEL\_TENSOR\_TRANSPOSE} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x34,
\texttt{VX\_KERNEL\_TENSOR\_CONVERT\_DEPTH} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x35,
\texttt{VX\_KERNEL\_TENSOR\_MATRIX\_MULTIPLY} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x36,
\texttt{VX\_KERNEL\_COPY} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x37,
\texttt{VX\_KERNEL\_NON\_MAX\_SUPPRESSION} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x38,
\texttt{VX\_KERNEL\_SCALAR\_OPERATION} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x39,
\texttt{VX\_KERNEL\_HOG\_FEATURES} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x3A,
\texttt{VX\_KERNEL\_HOG\_CELLS} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x3B,
\texttt{VX\_KERNEL\_BILATERAL\_FILTER} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x3C,
\texttt{VX\_KERNEL\_SELECT} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x3D,
\texttt{VX\_KERNEL\_MAX\_1\_2},
\texttt{VX\_KERNEL\_MAX} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x3E,
\texttt{VX\_KERNEL\_MIN} = VX\_KERNEL\_BASE(VX\_ID\_KHRONOS, VX\_LIBRARY\_KHR\_BASE) + 0x3F

The standard list of available vision kernels.

- enum vx\_library\_e { VX\_LIBRARY\_KHR\_BASE = 0x0 }

The standard list of available libraries.

Functions

- \texttt{vx\_kernel VX\_API\_CALL vxGetKernelByEnum (vx\_context context, vx\_enum kernel)}
  Obtains a reference to the kernel using the \texttt{vx\_kernel\_e} enumeration.

- \texttt{vx\_kernel VX\_API\_CALL vxGetKernelByName (vx\_context context, const vx\_char \*name)}
  Obtains a reference to a kernel using a string to specify the name.

- \texttt{vx\_status VX\_API\_CALL vxQueryKernel (vx\_kernel kernel, vx\_enum attribute, void \*ptr, vx\_size size)}
  This allows the client to query the kernel to get information about the number of parameters, enum values, etc.

- \texttt{vx\_status VX\_API\_CALL vxReleaseKernel (vx\_kernel \*kernel)}
  Release the reference to the kernel. The object may not be garbage collected until its total reference count is zero.

3.86.2 Data Structure Documentation

\texttt{struct vx\_kernel\_info\_t}  
The Kernel Information Structure. This is returned by the Context to indicate which kernels are available in the OpenVX implementation.
Definition at line 1608 of file vx_types.h.
CHAPTER 3. MODULE DOCUMENTATION

3.86.3 Typedef Documentation

typedef struct _vx_kernel* vx_kernel

An opaque reference to the descriptor of a kernel.

See also

vxGetKernelByName
vxGetKernelByEnum

Definition at line 187 of file vx_types.h.

3.86.4 Enumeration Type Documentation

enum vx_library_e

The standard list of available libraries.

Enumerator

VX_LIBRARY_KHR_BASE  The base set of kernels as defined by Khronos.

Definition at line 34 of file vx_kernels.h.

definition vx_kernel_e

The standard list of available vision kernels.

Each kernel listed here can be used with the vxGetKernelByEnum call. When programming the parameters, use

• VX_INPUT for [in]

• VX_OUTPUT for [out]

• VX_BIDIRECTIONAL for [in,out]

When programming the parameters, use

• VX_TYPE_IMAGE for a vx_image in the size field of vxGetParameterByIndex or vxSet← ParameterByIndex *

• VX_TYPE_ARRAY for a vx_array in the size field of vxGetParameterByIndex or vxSet← ParameterByIndex *

• or other appropriate types in vx_type_e.

Enumerator

VX_KERNEL_COLOR_CONVERT  The Color Space conversion kernel. The conversions are based on the vx_df_image_e code in the images.
See also

**Color Convert**

**VX_KERNEL_CHANNEL_EXTRACT** The Generic Channel Extraction Kernel. This kernel can remove individual color channels from an interleaved or semi-planar, planar, sub-sampled planar image. A client could extract a red channel from an interleaved RGB image or do a Luma extract from a YUV format.

See also

**Channel Extract**

**VX_KERNEL_CHANNEL_COMBINE** The Generic Channel Combine Kernel. This kernel combines multiple individual planes into a single multiplanar image of the type specified in the output image.

See also

**Channel Combine**

**VX_KERNEL_SOBE_3x3** The Sobel 3x3 Filter Kernel.

See also

**Sobel 3x3**

**VX_KERNEL_MAGNITUDE** The Magnitude Kernel. This kernel produces a magnitude plane from two input gradients.

See also

**Magnitude**

**VX_KERNEL_PHASE** The Phase Kernel. This kernel produces a phase plane from two input gradients.

See also

**Phase**

**VX_KERNEL_SCALE_IMAGE** The Scale Image Kernel. This kernel provides resizing of an input image to an output image. The scaling factor is determined by the relative sizes of the input and output.

See also

**Scale Image**

**VX_KERNEL_TABLE_LOOKUP** The Table Lookup kernel.

See also

**Table Lookup**

**VX_KERNEL_HISTOGRAM** The Histogram Kernel.

See also

**Histogram**

**VX_KERNEL_EQUALIZE_HISTOGRAM** The Histogram Equalization Kernel.

See also

**Equalize Histogram**

**VX_KERNEL_ABSDIFF** The Absolute Difference Kernel.

See also

**Absolute Difference**

**VX_KERNEL_MEAN_STDDEV** The Mean and Standard Deviation Kernel.

See also

**Mean and Standard Deviation**

**VX_KERNEL_THRESHOLD** The Threshold Kernel.

See also

**Thresholding**

**VX_KERNEL_INTEGRAL_IMAGE** The Integral Image Kernel.
See also

**Integral Image**

**VX_KERNEL_DILATE_3x3** The dilate kernel.

See also

**Dilate Image**

**VX_KERNEL_ERODE_3x3** The erode kernel.

See also

**Erode Image**

**VX_KERNEL_MEDIAN_3x3** The median image filter.

See also

**Median Filter**

**VX_KERNEL_BOX_3x3** The box filter kernel.

See also

**Box Filter**

**VX_KERNEL_GAUSSIAN_3x3** The gaussian filter kernel.

See also

**Gaussian Filter**

**VX_KERNEL_CUSTOM_CONVOLUTION** The custom convolution kernel.

See also

**Custom Convolution**

**VX_KERNEL_GAUSSIAN_PYRAMID** The gaussian image pyramid kernel.

See also

**Gaussian Image Pyramid**

**VX_KERNEL_ACCUMULATE** The accumulation kernel.

See also

**Accumulate**

**VX_KERNEL_ACCUMULATE_WEIGHTED** The weighed accumulation kernel.

See also

**Accumulate Weighted**

**VX_KERNEL_ACCUMULATE_SQUARE** The squared accumulation kernel.

See also

**Accumulate Squared**

**VX_KERNEL_MINMAXLOC** The min and max location kernel.

See also

**Min, Max Location**

**VX_KERNEL_CONVERTDEPTH** The bit-depth conversion kernel.

See also

**Convert Bit depth**

**VX_KERNEL_CANNY_EDGE_DETECTOR** The Canny Edge Detector.

See also

**Canny Edge Detector**

**VX_KERNEL_AND** The Bitwise And Kernel.
See also

**Bitwise AND**

**VX_KERNEL_OR**  The Bitwise Inclusive Or Kernel.

See also

**Bitwise INCLUSIVE OR**

**VX_KERNEL_XOR**  The Bitwise Exclusive Or Kernel.

See also

**Bitwise EXCLUSIVE OR**

**VX_KERNEL_NOT**  The Bitwise Not Kernel.

See also

**Bitwise NOT**

**VX_KERNEL_MULTIPLY**  The Pixelwise Multiplication Kernel.

See also

**Pixel-wise Multiplication**

**VX_KERNEL_ADD**  The Addition Kernel.

See also

**Arithmetic Addition**

**VX_KERNEL_SUBTRACT**  The Subtraction Kernel.

See also

**Arithmetic Subtraction**

**VX_KERNEL_WARP_AFFINE**  The Warp Affine Kernel.

See also

**Warp Affine**

**VX_KERNEL_WARP_PERSPECTIVE**  The Warp Perspective Kernel.

See also

**Warp Perspective**

**VX_KERNEL_HARRIS_CORNERS**  The Harris Corners Kernel.

See also

**Harris Corners**

**VX_KERNEL_FAST_CORNERS**  The FAST Corners Kernel.

See also

**Fast Corners**

**VX_KERNEL_OPTICAL_FLOW_PYR_LK**  The Optical Flow Pyramid (LK) Kernel.

See also

**Optical Flow Pyramid (LK)**

**VX_KERNEL_REMAP**  The Remap Kernel.

See also

**Remap**

**VX_KERNEL_HALFSIZE_GAUSSIAN**  The Half Scale Gaussian Kernel.

See also

**Scale Image**

**VX_KERNEL_LAPLACIAN_PYRAMID**  The Laplacian Image Pyramid Kernel.
See also

- Laplacian Image Pyramid

**VX_KERNEL_LAPLACIAN_RECONSTRUCT**  The Laplacian Pyramid Reconstruct Kernel.

See also

- Laplacian Image Pyramid

**VX_KERNEL_NON_LINEAR_FILTER**  The Non Linear Filter Kernel.

See also

- Non Linear Filter

**VX_KERNEL_MATCH_TEMPLATE**  The Match Template Kernel.

See also

- group_vision_match_template

**VX_KERNEL_LBP**  The LBP Kernel.

See also

- group_lbp

**VX_KERNEL_HOUGH_LINES_P**  The hough lines probability Kernel.

See also

- group_vision_hough_lines_p

**VX_KERNEL_TENSOR_MULTIPLY**  The tensor multiply Kernel.

See also

- Tensor Multiply

**VX_KERNEL_TENSOR_ADD**  The tensor add Kernel.

See also

- Tensor Add

**VX_KERNEL_TENSOR_SUBTRACT**  The tensor subtract Kernel.

See also

- Tensor Subtract

**VX_KERNEL_TENSOR_TABLE_LOOKUP**  The tensor table look up Kernel.

See also

- Tensor TableLookUp

**VX_KERNEL_TENSOR_TRANSPOSE**  The tensor transpose Kernel.

See also

- Tensor Transpose

**VX_KERNEL_TENSOR_CONVERT_DEPTH**  The tensor convert depth Kernel.

See also

- Tensor Convert Bit-Depth

**VX_KERNEL_TENSOR_MATRIX_MULTIPLY**  The tensor matrix multiply Kernel.

See also

- Tensor Matrix Multiply

**VX_KERNEL_COPY**  The data object copy kernel.

See also

- Data Object Copy

**VX_KERNEL_NON_MAX_SUPPRESSION**  The non-max suppression kernel.
See also Non-Maxima Suppression

**VX_KERNEL_SCALAR_OPERATION** The scalar operation kernel.

See also Control Flow

**VX_KERNEL_HOG_FEATURES** The HOG features kernel.

See also HOG

**VX_KERNEL_HOG_CELLS** The HOG Cells kernel.

See also HOG

**VX_KERNEL_BILATERAL_FILTER** The bilateral filter kernel.

See also Bilateral Filter

**VX_KERNEL_SELECT** The select kernel.

See also Control Flow

**VX_KERNEL_MAX** The max kernel.

See also Max

**VX_KERNEL_MIN** The min kernel.

See also Min

Definition at line 52 of file `vx_kernels.h`.

---

```c
enum vx_kernel_attribute_e
{
    VX_KERNEL_PARAMETERS,  /* Queries a kernel for the number of parameters the kernel supports. Read-only. Use a vx_uint32 parameter. */
    VX_KERNEL_NAME,        /* Queries the name of the kernel. Not settable. Read-only. Use a vx_char[VX_MAX[X_KERNEL_NAME]] array (not a vx_array). */
    VX_KERNEL_ENUM,        /* Queries the enum of the kernel. Not settable. Read-only. Use a vx_enum parameter. */
    VX_KERNEL_LOCAL_DATA_SIZE,  /* The local data area allocated with each kernel when it becomes a node. Read-write. Can be written only before user-kernel finalization. Use a vx_size parameter. */
    VX_KERNEL_CUSTOM
};
```

**Note**

If not set it will default to zero.

Definition at line 862 of file `vx_types.h`. 

---
3.86.5 Function Documentation

`vx_kernel VX_API_CALL vxGetKernelByName ( vx_context context, const vx_char * name )`

Obtains a reference to a kernel using a string to specify the name.

User Kernels follow a "dotted" hierarchical syntax. For example: "com.company.example.xyz". The following are strings specifying the kernel names:

- org.khronos.openvx.color_convert
- org.khronos.openvx.channel_extract
- org.khronos.openvx.channel_combine
- org.khronos.openvx.sobel_3x3
- org.khronos.openvx.magnitude
- org.khronos.openvx.phase
- org.khronos.openvx.scale_image
- org.khronos.openvx.table_lookup
- org.khronos.openvx.histogram
- org.khronos.openvx.equalize_histogram
- org.khronos.openvx.absdiff
- org.khronos.openvx.mean_stddev
- org.khronos.openvx.threshold
- org.khronos.openvx.integral_image
- org.khronos.openvx.dilate_3x3
- org.khronos.openvx.erode_3x3
- org.khronos.openvx.median_3x3
- org.khronos.openvx.box_3x3
- org.khronos.openvx.gaussian_3x3
- org.khronos.openvx.custom_convolution
- org.khronos.openvx.gaussian_pyramid
- org.khronos.openvx.accumulate
- org.khronos.openvx.accumulate_weighted
- org.khronos.openvx.accumulate_square
- org.khronos.openvx.minmaxloc
- org.khronos.openvx.convertdepth
- org.khronos.openvx.canny_edge_detector
- org.khronos.openvx.and
- org.khronos.openvx.or
- org.khronos.openvx.xor
- org.khronos.openvx.not
- org.khronos.openvx.multiply
- org.khronos.openvx.add
- org.khronos.openvx.subtract
- org.khronos.openvx.warp_affine
- org.khronos.openvx.warp_perspective
- org.khronos.openvx.harris_corners
- org.khronos.openvx.fast_corners
- org.khronos.openvx.optical_flow_pyr_lk
- org.khronos.openvx.remap
- org.khronos.openvx.halfscale_gaussian
- org.khronos.openvx.laplacian_pyramid
- org.khronos.openvx.laplacian_reconstruct
- org.khronos.openvx.non_linear_filter
- org.khronos.openvx.match_template
- org.khronos.openvx.libp
- org.khronos.openvx.hough_lines_p
- org.khronos.openvx.tensor_multiply
- org.khronos.openvx.tensor_add
- org.khronos.openvx.tensor_subtract
- org.khronos.openvx.tensor_table_lookup
- org.khronos.openvx.tensor_transpose
CHAPTER 3. MODULE DOCUMENTATION

org.khronos.openvx.tensor_convert_depth
org.khronos.openvx.tensor_matrix_multiply
org.khronos.openvx.copy
org.khronos.openvx.non_max_suppression
org.khronos.openvx.scalar_operation
org.khronos.openvx.hog_features
org.khronos.openvx.hog_cells
org.khronos.openvx.bilateral_filter
org.khronos.openvx.select
org.khronos.openvx.min
org.khronos.openvx.max

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the implementation context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>The string of the name of the kernel to get.</td>
</tr>
</tbody>
</table>

Returns

A kernel reference. Any possible errors preventing a successful completion of the function should be checked using `vxGetStatus`.

Precondition

`vxLoadKernels` if the kernel is not provided by the OpenVX implementation.

Note

User Kernels should follow a "dotted" hierarchical syntax. For example: "com.company.example.xyz".

`vx_kernel VX_API_CALL vxGetKernelByEnum ( vx_context context, vx_enum kernel )`

Obtains a reference to the kernel using the `vx_kernel_e` enumeration. Enum values above the standard set are assumed to apply to loaded libraries.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the implementation context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>kernel</td>
<td>A value from <code>vx_kernel_e</code> or a vendor or client-defined value.</td>
</tr>
</tbody>
</table>

Returns

A `vx_kernel` reference. Any possible errors preventing a successful completion of the function should be checked using `vxGetStatus`.

Precondition

`vxLoadKernels` if the kernel is not provided by the OpenVX implementation.

`vx_status VX_API_CALL vxQueryKernel ( vx_kernel kernel, vx_enum attribute, void * ptr, vx_size size )`

This allows the client to query the kernel to get information about the number of parameters, enum values, etc.

Parameters

| in  | kernel | The kernel reference to query. |
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>attribute</th>
<th>The attribute to query. Use a vx_kernel_attribute_e.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>ptr</td>
<td>The pointer to the location at which to store the resulting value.</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>The size of the container to which ptr points.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Kernel is not a valid vx_kernel reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>If any of the other parameters are incorrect.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUPPORTED</td>
<td>If the attribute value is not supported in this implementation.</td>
</tr>
</tbody>
</table>

vx_status VX_API_CALL vxReleaseKernel ( vx_kernel *kernel )

Release the reference to the kernel. The object may not be garbage collected until its total reference count is zero.

Parameters

| in | kernel | The pointer to the kernel reference to release. |

Postcondition

After returning from this function the reference is zeroed.

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Kernel is not a valid vx_kernel reference.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.87 Object: Parameter

3.87.1 Detailed Description

Defines the Parameter Object interface.

An abstract input, output, or bidirectional data object passed to a computer vision function. This object contains the signature of that parameter’s usage from the kernel description. This information includes:

- **Signature Index** - The numbered index of the parameter in the signature.
- **Object Type** - e.g., `VX_TYPE_IMAGE` or `VX_TYPE_ARRAY` or some other object type from `vx_type_e`.
- **Usage Model** - e.g., `VX_INPUT`, `VX_OUTPUT`, or `VX_BIDIRECTIONAL`.
- **Presence State** - e.g., `VX_PARAMETER_STATE_REQUIRED` or `VX_PARAMETER_STATE_OPTIONAL`.

**Typedefs**

- typedef struct _vx_parameter ∗vx_parameter

  An opaque reference to a single parameter.

**Enumerations**

- enum vx_direction_e {
  
  VX_INPUT = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_DIRECTION << 12)) + 0x0,
  VX_OUTPUT = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_DIRECTION << 12)) + 0x1,
  VX_BIDIRECTIONAL = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_DIRECTION << 12)) + 0x2
  }

  An indication of how a kernel will treat the given parameter.

- enum vx_parameter_attribute_e {
  
  VX_PARAMETER_INDEX = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PARAMETER << 8)) + 0x0,
  VX_PARAMETER_DIRECTION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PARAMETER << 8)) + 0x1,
  VX_PARAMETER_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PARAMETER << 8)) + 0x2,
  VX_PARAMETER_STATE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PARAMETER << 8)) + 0x3,
  VX_PARAMETER_REF = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PARAMETER << 8)) + 0x4
  }

  The parameter attributes list.

- enum vx_parameter_state_e {
  
  VX_PARAMETER_STATE_REQUIRED = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_PARAMETER_STATE << 12)) + 0x0,
  VX_PARAMETER_STATE_OPTIONAL = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_PARAMETER_STATE << 12)) + 0x1
  }

  The parameter state type.

**Functions**

- vx_parameter VX_API_CALL vxGetKernelParameterByIndex (vx_kernel kernel, vx_uint32 index)

  Retrieves a `vx_parameter` from a `vx_kernel`.

- vx_parameter VX_API_CALL vxGetParameterByIndex (vx_node node, vx_uint32 index)

  Retrieves a `vx_parameter` from a `vx_node`.

- vx_status VX_API_CALL vxQueryParameter (vx_parameter parameter, vx_enum attribute, void ∗ptr, vx_size size)

  Allows the client to query a parameter to determine its meta-information.

- vx_status VX_API_CALL vxReleaseParameter (vx_parameter ∗param)

  Releases a reference to a parameter object. The object may not be garbage collected until its total reference count is zero.

- vx_status VX_API_CALL vxSetParameterByIndex (vx_node node, vx_uint32 index, vx_reference value)

  Sets the specified parameter data for a kernel on the node.

- vx_status VX_API_CALL vxSetParameterByReference (vx_parameter parameter, vx_reference value)

  Associates a parameter reference and a data reference with a kernel on a node.
3.87.2 Typedef Documentation

typedef struct _vx_parameter* vx_parameter

An opaque reference to a single parameter.

See also

vxGetParameterByIndex

Definition at line 194 of file vx_types.h.

3.87.3 Enumeration Type Documentation

enum vx_direction_e

An indication of how a kernel will treat the given parameter.

Enumerator

VX_INPUT  The parameter is an input only.
VX_OUTPUT  The parameter is an output only.
VX_BIDIRECTIONAL  The parameter is both an input and output.

Definition at line 611 of file vx_types.h.

enum vx_parameter_attribute_e

The parameter attributes list.

Enumerator

VX_PARAMETER_INDEX  Queries a parameter for its index value on the kernel with which it is associated. Read-only. Use a vx_uint32 parameter.
VX_PARAMETER_DIRECTION  Queries a parameter for its direction value on the kernel with which it is associated. Read-only. Use a vx_enum parameter.
VX_PARAMETER_TYPE  Queries a parameter for its type, vx_type_e is returned. Read-only. The size of the parameter is implied for plain data objects. For opaque data objects like images and arrays a query to their attributes has to be called to determine the size.
VX_PARAMETER_STATE  Queries a parameter for its state. A value in vx_parameter_state_e is returned. Read-only. Use a vx_enum parameter.
VX_PARAMETER_REF  Use to extract the reference contained in the parameter. Read-only. Use a vx_reference parameter.

Definition at line 930 of file vx_types.h.

enum vx_parameter_state_e

The parameter state type.

Enumerator

VX_PARAMETER_STATE_REQUIRED  Default. The parameter must be supplied. If not set, during Verify, an error is returned.
VX_PARAMETER_STATE_OPTIONAL  The parameter may be unspecified. The kernel takes care not to deference optional parameters until it is certain they are valid.

Definition at line 1322 of file vx_types.h.

3.87.4 Function Documentation

vx_parameter VX_API_CALL vxGetKernelParameterByIndex ( vx_kernel kernel, vx_uint32 index )

Retrieves a vx_parameter from a vx_kernel.
CHAPTER 3. MODULE DOCUMENTATION

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel</td>
<td>The reference to the kernel.</td>
</tr>
<tr>
<td>index</td>
<td>The index of the parameter.</td>
</tr>
</tbody>
</table>

Returns

A vx_parameter reference. Any possible errors preventing a successful completion of the function should be checked using vxGetStatus.

vx_parameter VX_API_CALL vxGetParameterByIndex ( vx_node node, vx_uint32 index )

Retrieves a vx_parameter from a vx_node.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>The node from which to extract the parameter.</td>
</tr>
<tr>
<td>index</td>
<td>The index of the parameter to which to get a reference.</td>
</tr>
</tbody>
</table>

Returns

A parameter reference vx_parameter. Any possible errors preventing a successful completion of the function should be checked using vxGetStatus.

vx_status VX_API_CALL vxReleaseParameter ( vx_parameter * param )

Releases a reference to a parameter object. The object may not be garbage collected until its total reference count is zero.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>param</td>
<td>The pointer to the parameter to release.</td>
</tr>
</tbody>
</table>

Postcondition

After returning from this function the reference is zeroed.

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>param is not a valid vx_parameter reference.</td>
</tr>
</tbody>
</table>

vx_status VX_API_CALL vxSetParameterByIndex ( vx_node node, vx_uint32 index, vx_reference value )

Sets the specified parameter data for a kernel on the node.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>The node that contains the kernel.</td>
</tr>
<tr>
<td>index</td>
<td>The index of the parameter desired.</td>
</tr>
</tbody>
</table>
Parameters

| in | value | The desired value of the parameter. |

Note

A user may not provide a NULL value for a mandatory parameter of this API.

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>parameter is not a valid vx_parameter reference, or value is not a valid vx_reference reference.</td>
</tr>
</tbody>
</table>

See also

vxSetParameterByReference

vx_status VX_API_CALL vxSetParameterByReference ( vx_parameter parameter, vx_reference value )

Associates a parameter reference and a data reference with a kernel on a node.

Parameters

| in | parameter | The reference to the kernel parameter. |
| in | value | The value to associate with the kernel parameter. |

Note

A user may not provide a NULL value for a mandatory parameter of this API.

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>parameter is not a valid vx_parameter reference, or value is not a valid vx_reference reference.</td>
</tr>
</tbody>
</table>

See also

vxGetParameterByIndex

vx_status VX_API_CALL vxQueryParameter ( vx_parameter parameter, vx_enum attribute, void *ptr, vx_size size )

Allows the client to query a parameter to determine its meta-information.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>parameter</td>
</tr>
<tr>
<td>in</td>
<td>attribute</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>parameter is not a valid <code>vx_parameter</code> reference.</td>
</tr>
</tbody>
</table>
3.88 Advanced Framework API

3.88.1 Detailed Description

Describes components that are considered to be advanced.

Advanced topics include: extensions through User Kernels; Reflection and Introspection; Performance Tweaking through Hinting and Directives; and Debugging Callbacks.

Modules

- **Framework: Node Callbacks**
  
  Allows Clients to receive a callback after a specific node has completed execution.

- **Framework: Performance Measurement**

  Defines Performance measurement and reporting interfaces.

- **Framework: Log**

  Defines the debug logging interface.

- **Framework: Hints**

  Defines the Hints Interface.

- **Framework: Directives**

  Defines the Directives Interface.

- **Framework: User Kernels**

  Defines the User Kernels, which are a method to extend OpenVX with new vision functions.

- **Framework: Graph Parameters**

  Defines the Graph Parameter API.
CHAPTER 3. MODULE DOCUMENTATION

3.89 Framework: Node Callbacks

3.89.1 Detailed Description

Allows Clients to receive a callback after a specific node has completed execution.

Callbacks are not guaranteed to be called immediately after the Node completes. Callbacks are intended to be used to create simple early exit conditions for Vision graphs using \texttt{vx\_action\_e} return values. An example of setting up a callback can be seen below:

```c
vx_graph graph = vxCreateGraph(context);
status = vxGetStatus((vx_reference)graph);
if (status == VX_SUCCESS) {
    vx_uint8 lmin = 0, lmax = 0;
    vx_uint8 minCount = 0, maxCount = 0;
    vx_scalar scalars[] = {
        vxCreateScalar(context, VX_TYPE_UINT8, &lmin),
        vxCreateScalar(context, VX_TYPE_UINT8, &lmax),
        vxCreateScalar(context, VX_TYPE_UINT8, &minCount),
        vxCreateScalar(context, VX_TYPE_UINT8, &maxCount),
    };
    vx_array arrays[] = {
        vxCreateArray(context, VX_TYPE_COORDINATES2D, 1),
        vxCreateArray(context, VX_TYPE_COORDINATES2D, 1)
    };
    vx_node nodes[] = {
        vxMinMaxLocNode(graph, input, scalars[0], scalars[1], arrays[0], arrays[1],
                        scalars[2], scalars[3]),
    };
    status = vxAssignNodeCallback(nodes[0], &analyze_brightness);
    // do other
}
```

Once the graph has been initialized and the callback has been installed then the callback itself will be called during graph execution.

```c
#define MY_DESIRED_THRESHOLD (10)
vx_action VX_CALLBACK analyze_brightness(vx_node node) {
    // extract the max value
    vx_action action = VX_ACTION_ABANDON;
    vx_parameter pmax = vxGetParameterByIndex(node, 2); // Max Value
    if (pmax) {
        vx_scalar smax = 0;
        vxQueryParameter(pmax, VX_PARAMETER_REF, &smax, sizeof(smax));
        if (smax) {
            vx_uint8 value = 0u;
            vxCopyScalar(smax, &value, VX_READ_ONLY,
                         VX_MEMORY_TYPE_HOST);
            if (value >= MY_DESIRED_THRESHOLD) {
                action = VX_ACTION_CONTINUE;
            }
            vxReleaseScalar(&smax);
        }
        vxReleaseParameter(4pmax);
    }
    return action;
}
```

Warning

This should be used with \texttt{extreme} caution as it can ruin optimizations in the power/performance efficiency of a graph.

The callback must return a \texttt{vx\_action\_e} code indicating how the graph processing should proceed.

- If \texttt{VX\_ACTION\_CONTINUE} is returned, the graph will continue execution with no changes.
- If \texttt{VX\_ACTION\_ABANDON} is returned, execution is unspecified for all nodes for which this node is a dominator. Nodes that are dominators of this node will have executed. Execution of any other node is unspecified.
### CHAPTER 3. MODULE DOCUMENTATION

#### Client OpenVX

**Graph Setup**

- `vxAssignNodeCallback(node, &myfunc);`
- `vx_status vxVerifyGraph(graph);`
- `vx_status vxProcessGraph(graph);`

---

- `myfunc(node, ...);`
- `vx_action action taken`
- `vx_status vxRetrieveNodeCallback(vx_nodecomplete_f) myfunc`

---

**Figure 3.2: Node Callback Sequence**

---

**Typedefs**

- `typedef vx_enum vx_action` 
  The formal typedef of the response from the callback.
- `typedef vx_action(vx_nodecomplete_f) (vx_node node)` 
  A callback to the client after a particular node has completed.

---

**Enumerations**

- `enum vx_action_e {
  VX_ACTION_CONTINUE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_ACTION << 12)) + 0x0,
  VX_ACTION_ABANDON = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_ACTION << 12)) + 0x1
}`

  A return code enumeration from a `vx_nodecomplete_f` during execution.

---

**Functions**

- `vx_status VX_API_CALL vxAssignNodeCallback (vx_node node, vx_nodecomplete_f callback)` 
  Assigns a callback to a node. If a callback already exists in this node, this function must return an error and the user may clear the callback by passing a NULL pointer as the callback.
- `vx_nodecomplete_f VX_API_CALL vxRetrieveNodeCallback (vx_node node)` 
  Retrieves the current node callback function pointer set on the node.
3.89.2 Typedef Documentation

typedef vx_enum vx_action
The formal typedef of the response from the callback.
See also
   vx_action_e
   Definition at line 451 of file vx_types.h.

typedef vx_action( * vx_nodecomplete_f)(vx_node node)
A callback to the client after a particular node has completed.
See also
   vx_action
   vxAssignNodeCallback

Parameters

| in | node | The node to which the callback was attached. |

Returns

An action code from vx_action_e.

Definition at line 460 of file vx_types.h.

3.89.3 Enumeration Type Documentation

enum vx_action_e
A return code enumeration from a vx_nodecomplete_f during execution.
See also
   vxAssignNodeCallback

Enumerator

   VX_ACTION_CONTINUE  Continue executing the graph with no changes.
   VX_ACTION_ABANDON   Stop executing the graph.

Definition at line 601 of file vx_types.h.

3.89.4 Function Documentation

vx_status VX_API_CALL vxAssignNodeCallback ( vx_node node, vx_nodecomplete_f callback )
Assigns a callback to a node. If a callback already exists in this node, this function must return an error and the user may clear the callback by passing a NULL pointer as the callback.

Parameters

| in | node | The reference to the node. |
| in | callback | The callback to associate with completion of this specific node. |
Warning

This must be used with extreme caution as it can ruin optimizations in the power/performance efficiency of a graph.

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Callback assigned; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>node is not a valid vx_node reference.</td>
</tr>
</tbody>
</table>

vx_nodecomplete_f VX_API_CALL vxRetrieveNodeCallback ( vx_node node )

Retrieves the current node callback function pointer set on the node.

Parameters

in  node  The reference to the vx_node object.

Returns

vx_nodecomplete_f The pointer to the callback function.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>No callback is set.</td>
</tr>
<tr>
<td>*</td>
<td>The node callback function.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.90 Framework: Performance Measurement

3.90.1 Detailed Description

Defines Performance measurement and reporting interfaces.

In OpenVX, both `vx_graph` objects and `vx_node` objects track performance information. A client can query either object type using their respective `vxQuery<Object>` function with their attribute enumeration `VX_<OBJECT>_PERFORMANCE` along with a `vx_perf_t` structure to obtain the performance information.

```c
vx_perf_t perf;
vxQueryNode(node, VX_NODE_PERFORMANCE, &perf, sizeof(perf));
```

Data Structures

- `struct vx_perf_t`

  The performance measurement structure. The time or durations are in units of nano seconds. More...

3.90.2 Data Structure Documentation

`struct vx_perf_t`

The performance measurement structure. The time or durations are in units of nano seconds. Definition at line 1538 of file `vx_types.h`.

Data Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vx_uint64</code> tmp</td>
<td>Holds the last measurement.</td>
<td></td>
</tr>
<tr>
<td><code>vx_uint64</code> beg</td>
<td>Holds the first measurement in a set.</td>
<td></td>
</tr>
<tr>
<td><code>vx_uint64</code> end</td>
<td>Holds the last measurement in a set.</td>
<td></td>
</tr>
<tr>
<td><code>vx_uint64</code> sum</td>
<td>Holds the summation of durations.</td>
<td></td>
</tr>
<tr>
<td><code>vx_uint64</code> avg</td>
<td>Holds the average of the durations.</td>
<td></td>
</tr>
<tr>
<td><code>vx_uint64</code> min</td>
<td>Holds the minimum of the durations.</td>
<td></td>
</tr>
<tr>
<td><code>vx_uint64</code> num</td>
<td>Holds the number of measurements.</td>
<td></td>
</tr>
<tr>
<td><code>vx_uint64</code> max</td>
<td>Holds the maximum of the durations.</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.91 Framework: Log

3.91.1 Detailed Description

Defines the debug logging interface.

The functions of the debugging interface allow clients to receive important debugging information about OpenVX. See also \texttt{vx\_status\_e} for the list of possible errors.

![Diagram of Log messages only can be received after the callback is installed.](image)

**Typedefs**

- typedef void(\texttt{vx\_log\_callback\_f}) (\texttt{vx\_context context, vx\_reference ref, vx\_status status, const vx\_char string[]})

  \textit{The log callback function.}

**Functions**

- void \texttt{VX\_API\_CALL vx\_Add\_Log\_Entry (vx\_reference ref, vx\_status status, const char *message,...)}

  \textit{Adds a line to the log.}

- void \texttt{VX\_API\_CALL vx\_Register\_Log\_Callback (vx\_context context, vx\_log\_callback\_f callback, vx\_bool reentrant)}

  \textit{ Registers a callback facility to the OpenVX implementation to receive error logs.}

**3.91.2 Function Documentation**

\texttt{void VX\_API\_CALL vx\_Add\_Log\_Entry ( vx\_reference ref, vx\_status status, const char * message, ... )}

Adds a line to the log.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{ref}</td>
<td>The reference to add the log entry against. Some valid value must be provided.</td>
</tr>
<tr>
<td>\texttt{status}</td>
<td>The status code. \texttt{VX_SUCCESS} status entries are ignored and not added.</td>
</tr>
<tr>
<td>\texttt{message}</td>
<td>The human readable message to add to the log.</td>
</tr>
<tr>
<td>\texttt{...}</td>
<td>a list of variable arguments to the message.</td>
</tr>
</tbody>
</table>
Note

Messages may not exceed `VX_MAX_LOG_MESSAGE_LEN` bytes and will be truncated in the log if they exceed this limit.

```c
void VX_API_CALL vxRegisterLogCallback ( vx_context context, vx_log_callback_f callback, vx_bool reentrant )
```

Registers a callback facility to the OpenVX implementation to receive error logs.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>context</code></td>
<td>The overall context to OpenVX.</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>callback</code></td>
<td>The callback function. If NULL, the previous callback is removed.</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>reentrant</code></td>
<td>If reentrancy flag is <code>vx_true_e</code>, then the callback may be entered from multiple simultaneous tasks or threads (if the host OS supports this).</td>
</tr>
</tbody>
</table>
3.92 Framework: Hints

3.92.1 Detailed Description

Defines the Hints Interface.

Hints are messages given to the OpenVX implementation that it may support. (These are optional.)

Enumerations

- enum vx_hint_e {
  
  VX_HINT_PERFORMANCE_DEFAULT = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_HINT << 12) + 0x1,
  VX_HINT_PERFORMANCE_LOW_POWER = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_HINT << 12))
  + 0x2,
  VX_HINT_PERFORMANCE_HIGH_SPEED = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_HINT << 12))
  + 0x3 }

  These enumerations are given to the vxHint API to enable/disable platform optimizations and/or features. Hints are optional and usually are vendor-specific.

Functions

- vx_status VX_API_CALL vxHint (vx_reference reference, vx_enum hint, const void *data, vx_size data_size)

  Provides a generic API to give platform-specific hints to the implementation.

3.92.2 Enumeration Type Documentation

definex vx_hint_e

These enumerations are given to the vxHint API to enable/disable platform optimizations and/or features. Hints are optional and usually are vendor-specific.

See also

vxHint

Enumerators

VX_HINT_PERFORMANCE_DEFAULT Indicates to the implementation that user do not apply any specific requirements for performance.

VX_HINT_PERFORMANCE_LOW_POWER Indicates the user preference is low power consumption versus highest performance.

VX_HINT_PERFORMANCE_HIGH_SPEED Indicates the user preference for highest performance over low power consumption.

Definition at line 625 of file vx_types.h.

3.92.3 Function Documentation

vx_status VX_API_CALL vxHint ( vx_reference reference, vx_enum hint, const void *data, vx_size data_size )

Provides a generic API to give platform-specific hints to the implementation.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in reference</td>
<td>The reference to the object to hint at. This could be vx_context, vx_graph, vx_node, vx_image, vx_array, or any other reference.</td>
</tr>
<tr>
<td>in hint</td>
<td>A vx_hint_e hint to give to a vx_context. This is a platform-specific optimization or implementation mechanism.</td>
</tr>
<tr>
<td>in data</td>
<td>Optional vendor specific data.</td>
</tr>
<tr>
<td>in data_size</td>
<td>Size of the data structure data.</td>
</tr>
</tbody>
</table>
Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>reference is not a valid <code>vx_reference</code> reference.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUPPORTED</td>
<td>If the hint is not supported.</td>
</tr>
</tbody>
</table>
3.93 Framework: Directives

3.93.1 Detailed Description

Defines the Directives Interface.

Directives are messages given the OpenVX implementation that it must support. (These are required, i.e., non-optional.)

Enumerations

- enum vx_directive_e {
  VX_DIRECTIVE_DISABLE_LOGGING = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_DIRECTIVE << 12)) + 0x0,
  VX_DIRECTIVE_ENABLE_LOGGING = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_DIRECTIVE << 12)) + 0x1,
  VX_DIRECTIVE_DISABLE_PERFORMANCE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_DIRECTIVE << 12)) + 0x2,
  VX_DIRECTIVE_ENABLE_PERFORMANCE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_DIRECTIVE << 12)) + 0x3
}

These enumerations are given to the vxDirective API to enable/disable platform optimizations and/or features. Directives are not optional and usually are vendor-specific, by defining a vendor range of directives and starting their enumeration from there.

Functions

- vx_status VX_API_CALL vxDirective (vx_reference reference, vx_enum directive)

  Provides a generic API to give platform-specific directives to the implementations.

3.93.2 Enumeration Type Documentation

den vx_directive_e

These enumerations are given to the vxDirective API to enable/disable platform optimizations and/or features. Directives are not optional and usually are vendor-specific, by defining a vendor range of directives and starting their enumeration from there.

See also

vxDirective

Enumerator

- VX_DIRECTIVE_DISABLE_LOGGING  Disables recording information for graph debugging.
- VX_DIRECTIVE_ENABLE_LOGGING    Enables recording information for graph debugging.
- VX_DIRECTIVE_DISABLE_PERFORMANCE Disables performance counters for the context. By default performance counters are disabled.
- VX_DIRECTIVE_ENABLE_PERFORMANCE Enables performance counters for the context.

Definition at line 647 of file vx_types.h.

3.93.3 Function Documentation

vx_status VX_API_CALL vxDirective ( vx_reference reference, vx_enum directive )

Provides a generic API to give platform-specific directives to the implementations.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td>The reference to the object to set the directive on. This could be vx_context, vx_graph, vx_node, vx_image, vx_array, or any other reference.</td>
</tr>
<tr>
<td>directive</td>
<td>The directive to set. See vx_directive_e.</td>
</tr>
</tbody>
</table>
Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Reference is not a valid vx_reference reference.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUPPORTED</td>
<td>If the directive is not supported.</td>
</tr>
</tbody>
</table>

Note

The performance counter directives are only available for the reference vx_context. Error VX_ERROR_NOT_SUPPORTED is returned when used with any other reference.
3.94 Framework: User Kernels

3.94.1 Detailed Description

Defines the User Kernels, which are a method to extend OpenVX with new vision functions.

User Kernels can be loaded by OpenVX and included as nodes in the graph or as immediate functions (if the Client supplies the interface). User Kernels will typically be loaded and executed on High Level Operating System/CPU compatible targets, not on remote processors or other accelerators. This specification does not mandate what constitutes compatible platforms.

---

![Call sequence of User Kernels Installation](image)

Figure 3.4: Call sequence of User Kernels Installation
Figure 3.5: Call sequence of a Graph Verify and Release with User Kernels.

During the first graph verification, the implementation will perform the following action sequence:

1. Initialize local data node attributes

   - If VX_KERNEL_LOCAL_DATA_SIZE == 0, then set VX_NODE_LOCAL_DATA_SIZE to 0 and set VX_NODE_LOCAL_DATA_PTR to NULL.
• If \texttt{VX\_KERNEL\_LOCAL\_DATA\_SIZE} \neq 0, set \texttt{VX\_NODE\_LOCAL\_DATA\_SIZE} to \texttt{VX\_KERNEL\_LOCAL\_DATA\_SIZE} and set \texttt{VX\_NODE\_LOCAL\_DATA\_PTR} to the address of a buffer of \texttt{VX\_KERNEL\_LOCAL\_DATA\_SIZE} bytes.

2. Call the \texttt{vx\_kernel\_validate\_f} callback.

3. Call the \texttt{vx\_kernel\_initialize\_f} callback (if not NULL):
   • If \texttt{VX\_KERNEL\_LOCAL\_DATA\_SIZE} == 0, the callback is allowed to set \texttt{VX\_NODE\_LOCAL\_DATA\_SIZE} and \texttt{VX\_NODE\_LOCAL\_DATA\_PTR}.
   • If \texttt{VX\_KERNEL\_LOCAL\_DATA\_SIZE} \neq 0, then any attempt by the callback to set \texttt{VX\_NODE\_LOCAL\_DATA\_SIZE} or \texttt{VX\_NODE\_LOCAL\_DATA\_PTR} attributes will generate an error.

4. Provide the buffer optionally requested by the application
   • If \texttt{VX\_KERNEL\_LOCAL\_DATA\_SIZE} == 0 and \texttt{VX\_NODE\_LOCAL\_DATA\_SIZE} \neq 0, and \texttt{VX\_NODE\_LOCAL\_DATA\_PTR} == NULL, then the implementation will set \texttt{VX\_NODE\_LOCAL\_DATA\_PTR} to the address of a buffer of \texttt{VX\_NODE\_LOCAL\_DATA\_SIZE} bytes.

At node destruction time, the implementation will perform the following action sequence:

1. Call \texttt{vx\_kernel\_deinitialize\_f} callback (if not NULL): If the \texttt{VX\_NODE\_LOCAL\_DATA\_PTR} was set earlier by the implementation, then any attempt by the callback to set the \texttt{VX\_NODE\_LOCAL\_DATA\_PTR} attributes will generate an error.

2. If the \texttt{VX\_NODE\_LOCAL\_DATA\_PTR} was set earlier by the implementation, then the pointed memory must not be used anymore by the application after the \texttt{vx\_kernel\_deinitialize\_f} callback completes.

A user node requires re-verification, if any changes below occurred after the last node verification:

1. The \texttt{VX\_NODE\_BORDER} node attribute was modified.

2. At least one of the node parameters was replaced by a data object with different meta-data, or was replaced by the 0 reference for optional parameters, or was set to a data object if previously not set because optional.

The node re-verification can by triggered explicitly by the application by calling \texttt{vx\_Verify\_Graph} that will perform a complete graph verification. Otherwise, it will be triggered automatically at the next graph execution.

During user node re-verification, the following action sequence will occur:

1. Call the \texttt{vx\_kernel\_deinitialize\_f} callback (if not NULL): If the \texttt{VX\_NODE\_LOCAL\_DATA\_PTR} was set earlier by the \texttt{OpenVX} implementation, then any attempt by the callback to set the \texttt{VX\_NODE\_LOCAL\_DATA\_PTR} attributes will generate an error.

2. Reinitialize local data node attributes if needed if \texttt{VX\_KERNEL\_LOCAL\_DATA\_SIZE} == 0:
   • set \texttt{VX\_NODE\_LOCAL\_DATA\_PTR} to NULL.
   • set \texttt{VX\_NODE\_LOCAL\_DATA\_SIZE} to 0.

3. Call the \texttt{vx\_kernel\_validate\_f} callback.

4. Call the \texttt{vx\_kernel\_initialize\_f} callback (if not NULL):
   • If \texttt{VX\_KERNEL\_LOCAL\_DATA\_SIZE} == 0, the callback is allowed to set \texttt{VX\_NODE\_LOCAL\_DATA\_SIZE} and \texttt{VX\_NODE\_LOCAL\_DATA\_PTR}.
   • If \texttt{VX\_KERNEL\_LOCAL\_DATA\_SIZE} \neq 0, then any attempt by the callback to set \texttt{VX\_NODE\_LOCAL\_DATA\_SIZE} or \texttt{VX\_NODE\_LOCAL\_DATA\_PTR} attributes will generate an error.

5. Provide the buffer optionally requested by the application
   • If \texttt{VX\_KERNEL\_LOCAL\_DATA\_SIZE} == 0 and \texttt{VX\_NODE\_LOCAL\_DATA\_SIZE} \neq 0, and \texttt{VX\_NODE\_LOCAL\_DATA\_PTR} == NULL, then the \texttt{OpenVX} implementation will set \texttt{VX\_NODE\_LOCAL\_DATA\_PTR} to the address of a buffer of \texttt{VX\_NODE\_LOCAL\_DATA\_SIZE} bytes.

When an \texttt{OpenVX} implementation sets the \texttt{VX\_NODE\_LOCAL\_DATA\_PTR}, the data inside the buffer will not be persistent between kernel executions.
CHAPTER 3. MODULE DOCUMENTATION

Typedefs

- typedef vx_status(vx_node node, const vx_reference *parameters, vx_uint32 num) vx_kernel_deinitialize_f
  The pointer to the kernel deinitializer. If the host code requires a call to deinitialize data during a node garbage collection, this function is called if not NULL.

- typedef vx_status(vx_node node, const vx_reference *parameters, vx_uint32 num) vx_kernel_f
  The pointer to the Host side kernel.

- typedef vx_status(vx_node node, vx_uint32 index, const vx_rectangle_t *const input_valid[], vx_rectangle_t *const output_valid[]) vx_kernel_image_valid_rectangle_f
  A user-defined callback function to set the valid rectangle of an output image.

- typedef vx_status(vx_node node, const vx_reference *parameters, vx_uint32 num) vx_kernel_initialize_f
  The pointer to the kernel initializer. If the host code requires a call to initialize data once all the parameters have been validated, this function is called if not NULL.

- typedef vx_status(vx_node node, const vx_reference parameters[], vx_uint32 num, vx_meta_format metas[]) vx_kernel_validate_f
  The user-defined kernel node parameters validation function. The function only needs to fill in the meta data structure(s).

- typedef vx_meta_format * vx_meta_format
  This object is used by output validation functions to specify the meta data of the expected output data object.

- typedef vx_status(vx_context context) vx_publish_kernels_f
  The type of the vxPublishKernels entry function of modules loaded by vxLoadKernels and unloaded by vxUnloadKernels.

- typedef vx_status(vx_context context) vx_unpublish_kernels_f
  The type of the vxUnpublishKernels entry function of modules loaded by vxLoadKernels and unloaded by vxUnloadKernels.

Enumerations

- enum vx_meta_valid_rect_attribute_e { VX_VALID_RECT_CALLBACK = ((( VX_ID_KHRONOS )< 20) | (VX_TYPE_META_FORMAT << 8)) + 0x1 }
  The meta valid rectangle attributes.

Functions

- vx_status VX_API_CALL vxAddParameterToKernel(vx_kernel kernel, vx_uint32 index, vx_enum dir, vx_enum data_type, vx_enum state)
  Allows users to set the signatures of the custom kernel.

- vx_kernel VX_API_CALL vxAddUserKernel(vx_context context, const vx_char name[VX_MAX_KERNEL_NAME], vx_enum enumeration, vx_kernel_f func_ptr, vx_uint32 numParams, vx_kernel_validate_f validate, vx_kernel_initialize_f init, vx_kernel_deinitialize_f deinit)
  Allows users to add custom kernels to a context at run-time.

- vx_status VX_API_CALL vxAllocateKernelId(vx_context context, vx_enum *pKernelEnumId)
  Allocates and registers user-defined kernel enumeration to a context. The allocated enumeration is from available pool of 4096 enumerations reserved for dynamic allocation from VX_KERNEL_BASE(VX_ID_USER,0).

- vx_status VX_API_CALL vxAllocateKernelLibraryId(vx_context context, vx_enum *pLibraryId)
  Allocates and registers user-defined kernel library ID to a context.

- vx_status VX_API_CALL vxFinalizeKernel(vx_kernel kernel)
  This API is called after all parameters have been added to the kernel and the kernel is ready to be used. Notice that the reference to the kernel created by vxAddUserKernel is still valid after the call to vxFinalizeKernel. If an error occurs, the kernel is not available for usage by the clients of OpenVX. Typically this is due to a mismatch between the number of parameters requested and given.

- vx_status VX_API_CALL vxLoadKernels(vx_context context, const vx_char *module)
  Loads a library of kernels, called module, into a context.

- vx_status VX_API_CALL vxRemoveKernel(vx_kernel kernel)
Removes a custom kernel from its context and releases it.

- **vx_status VX_API_CALL vxSetKernelAttribute (vx_kernel kernel, vx_enum attribute, const void *ptr, vx_size size)**
  
  Sets kernel attributes.

- **vx_status VX_API_CALL vxSetMetaFormatAttribute (vx_meta_format meta, vx_enum attribute, const void *ptr, vx_size size)**
  
  This function allows a user to set the attributes of a vx_meta_format object in a kernel output validator.

- **vx_status VX_API_CALL vxSetMetaFormatFromReference (vx_meta_format meta, vx_reference exemplar)**
  
  Set a meta format object from an exemplar data object reference.

- **vx_status VX_API_CALL vxUnloadKernels (vx_context context, const vx_char *module)**
  
  Unloads all kernels from the OpenVX context that had been loaded from the module using the vxLoadKernels function.

### 3.94.2 Typedef Documentation

**typedef struct _vx_meta_format* vx_meta_format**

This object is used by output validation functions to specify the meta data of the expected output data object.

**Note**

When the actual output object of the user node is virtual, the information given through the vx_meta_format object allows the OpenVX framework to automatically create the data object when meta data were not specified by the application at object creation time.

Definition at line 320 of file vx_types.h.

**typedef vx_status(* vx_publish_kernels_f) (vx_context context)**

The type of the vxPublishKernels entry function of modules loaded by vxLoadKernels and unloaded by vxUnloadKernels.

**Parameters**

- **in context** The reference to the context kernels must be added to.

Definition at line 1744 of file vx_types.h.

**typedef vx_status(* vx_unpublish_kernels_f) (vx_context context)**

The type of the vxUnpublishKernels entry function of modules loaded by vxLoadKernels and unloaded by vxUnloadKernels.

**Parameters**

- **in context** The reference to the context kernels have been added to.

Definition at line 1752 of file vx_types.h.

**typedef vx_status(* vx_kernel_f) (vx_node node, const vx_reference *parameters, vx_uint32 num)**

The pointer to the Host side kernel.

**Parameters**

- **in node** The handle to the node that contains this kernel.
- **in parameters** The array of parameter references.
- **in num** The number of parameters.
CHAPTER 3. MODULE DOCUMENTATION

Definition at line 1761 of file vx_types.h.

typedef vx_status(*vx_kernel_initialize_f)(vx_node node, const vx_reference*parameters, vx_uint32 num)

The pointer to the kernel initializer. If the host code requires a call to initialize data once all the parameters have been validated, this function is called if not NULL.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>node</th>
<th>The handle to the node that contains this kernel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>parameters</td>
<td>The array of parameter references.</td>
</tr>
<tr>
<td>in</td>
<td>num</td>
<td>The number of parameters.</td>
</tr>
</tbody>
</table>

Definition at line 1772 of file vx_types.h.

typedef vx_status(*vx_kernel_deinitialize_f)(vx_node node, const vx_reference*parameters, vx_uint32 num)

The pointer to the kernel deinitializer. If the host code requires a call to deinitialize data during a node garbage collection, this function is called if not NULL.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>node</th>
<th>The handle to the node that contains this kernel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>parameters</td>
<td>The array of parameter references.</td>
</tr>
<tr>
<td>in</td>
<td>num</td>
<td>The number of parameters.</td>
</tr>
</tbody>
</table>

Definition at line 1783 of file vx_types.h.

typedef vx_status(*vx_kernel_validate_f)(vx_node node, const vx_reference parameters[], vx_uint32 num, vx_meta_format metas[])

The user-defined kernel node parameters validation function. The function only needs to fill in the meta data structure(s).

Note

This function is called once for whole set of parameters.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>node</th>
<th>The handle to the node that is being validated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>parameters</td>
<td>The array of parameters to be validated.</td>
</tr>
<tr>
<td>in</td>
<td>num</td>
<td>Number of parameters to be validated.</td>
</tr>
<tr>
<td>in</td>
<td>metas</td>
<td>A pointer to a pre-allocated array of structure references that the system holds. The system pre-allocates a number of vx_meta_format structures for the output parameters only, indexed by the same indices as parameters[]. The validation function fills in the correct type, format, and dimensionality for the system to use either to create memory or to check against existing memory.</td>
</tr>
</tbody>
</table>

Returns

An error code describing the validation status on parameters.

Definition at line 1799 of file vx_types.h.
typedef vx_status ( ∗ vx_kernel_image_valid_rectangle_f) (vx_node node, vx_uint32 index, const vx_rectangle_t ∗ const input_valid[], vx_rectangle_t ∗ const output_valid[])

A user-defined callback function to set the valid rectangle of an output image.

The VX_VALID_RECT_CALLBACK attribute in the vx_meta_format object should be set to the desired callback during user node's output validator. The callback must not call vxGetValidRegionImage or vx->SetImageValidRectangle. Instead, an array of the valid rectangles of all the input images is supplied to the callback to calculate the output valid rectangle. The output of the user node may be a pyramid, or just an image. If it is just an image, the 'Out' array associated with that output only has one element. If the output is a pyramid, the array size is equal to the number of pyramid levels. Notice that the array memory allocation passed to the callback is managed by the framework, the application must not allocate or deallocate those pointers.

The behavior of the callback function vx_kernel_image_valid_rectangle_f is undefined if one of the following is true:

- One of the input arguments of a user node is a pyramid or an array of images.
- Either input or output argument of a user node is an array of pyramids.

Parameters

| in, out | node                     | The handle to the node that is being validated. |
| in      | index                    | The index of the output parameter for which a valid region should be set. |
| in      | input_valid              | A pointer to an array of valid regions of input images or images contained in image container (e.g. pyramids). They are provided in same order as the parameter list of the kernel's declaration. |
| out     | output_valid             | An array of valid regions that should be set for the output images or image containers (e.g. pyramid) after graph processing. The length of the array should be equal to the size of the image container (e.g. number of levels in the pyramid). For a simple output image the array size is always one. Each rectangle supplies the valid region for one image. The array memory allocation is managed by the framework. |

Returns

An error code describing the validation status on parameters.

Definition at line 1832 of file vx_types.h.

3.94.3 Enumeration Type Documentation
def vx_meta_valid_rect_attribute_e

The meta valid rectangle attributes.

Enumerator

VX_VALID_RECT_CALLBACK Valid rectangle callback during output parameter validation. Write-only.

Definition at line 1183 of file vx_types.h.

3.94.4 Function Documentation
def vx_status VX_API_CALL vxAllocateUserKernelId (vx_context context, vx_enum * pKernelEnumId)

Allocates and registers user-defined kernel enumeration to a context. The allocated enumeration is from available pool of 4096 enumerations reserved for dynamic allocation from VX_KERNEL_BASE(VX_ID_USER,0).

Parameters

| in | context | The reference to the implementation context. |
| out | pKernelEnumId | pointer to return vx_enum for user-defined kernel. |
Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>If the context is not a valid vx_context reference.</td>
</tr>
<tr>
<td>VX_ERROR_NO_RESOURCES</td>
<td>The enumerations has been exhausted.</td>
</tr>
</tbody>
</table>

vx_status VX_API_CALL vxAllocateUserKernelLibraryId ( vx_context context, vx_enum * pLibraryId )

Allocates and registers user-defined kernel library ID to a context.
The allocated library ID is from available pool of library IDs (1..255) reserved for dynamic allocation. The
returned libraryId can be used by user-kernel library developer to specify individual kernel enum IDs in a header
file, shown below:

00001 #define MY_KERNEL_ID1(libraryId) (VX_KERNEL_BASE(VX_ID_USER,libraryId) + 0);
00002 #define MY_KERNEL_ID2(libraryId) (VX_KERNEL_BASE(VX_ID_USER,libraryId) + 1);
00003 #define MY_KERNEL_ID3(libraryId) (VX_KERNEL_BASE(VX_ID_USER,libraryId) + 2);

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the implementation context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>p←LibraryId</td>
<td>pointer to vx_enum for user-kernel libraryId.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_NO_RESOURCES</td>
<td>The enumerations has been exhausted.</td>
</tr>
</tbody>
</table>

vx_status VX_API_CALL vxLoadKernels ( vx_context context, const vx_char * module )

Loads a library of kernels, called module, into a context.
The module must be a dynamic library with by convention, two exported functions named vxPublishKernels and vxUnpublishKernels.
vxPublishKernels must have type vx_publish_kernels_f, and must add kernels to the context by calling vxAddUserKernel for each new kernel. vxPublishKernels is called by vxLoadKernels.
vxUnpublishKernels must have type vx_unpublish_kernels_f, and must remove kernels from the context by calling vxRemoveKernel for each kernel the vxPublishKernels has added. vxUnpublishKernels is called by vxUnloadKernels.

Note
When all references to loaded kernels are released, the module may be automatically unloaded.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the context the kernels must be added to.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>module</td>
<td>The short name of the module to load. On systems where there are specific naming conventions for modules, the name passed should ignore such conventions. For example: libxyz.so should be passed as just xyz and the implementation will do the right thing that the platform requires.</td>
</tr>
</tbody>
</table>
Note

This API uses the system pre-defined paths for modules.

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th><code>vx_status_e</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Context is not a valid <code>vx_context</code> reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>If any of the other parameters are incorrect.</td>
</tr>
</tbody>
</table>

See also

`vxGetKernelByName`

`vx_status VX_API_CALL vxUnloadKernels ( vx_context context, const vx_char * module )`

Unloads all kernels from the OpenVX context that had been loaded from the module using the `vxLoadKernels` function.

The kernel unloading is performed by calling the `vxUnpublishKernels` exported function of the module.

Note

`vxUnpublishKernels` is defined in the description of `vxLoadKernels`.

Parameters

| in | context | The reference to the context the kernels must be removed from. |
| in | module | The short name of the module to unload. On systems where there are specific naming conventions for modules, the name passed should ignore such conventions. For example: `libxyz.so` should be passed as just `xyz` and the implementation will do the right thing that the platform requires. |

Note

This API uses the system pre-defined paths for modules.

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th><code>vx_status_e</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Context is not a valid <code>vx_context</code> reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>If any of the other parameters are incorrect.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

See also

vxLoadKernels

vx_kernel VX_API_CALL vxAddUserKernel ( vx_context context, const vx_char name[ VX_MAX_KERNEL_NAME ], vx_enum enumeration, vx_kernel_f func_ptr, vx_uint32 numParams, vx_kernel_validate_f validate, vx_kernel_initialize_f init, vx_kernel_deinitialize_f deinit )

Allows users to add custom kernels to a context at run-time.

Parameters

| in | context | The reference to the context the kernel must be added to. |
| in | name | The string to use to match the kernel. |
| in | enumeration | The enumerated value of the kernel to be used by clients. |
| in | func_ptr | The process-local function pointer to be invoked. |
| in | numParams | The number of parameters for this kernel. |
| in | validate | The pointer to vx_kernel_validate_f, which validates parameters to this kernel. |
| in | init | The kernel initialization function. |
| in | deinit | The kernel de-initialization function. |

Returns

A vx_kernel reference. Any possible errors preventing a successful creation should be checked using vxGetStatus.

vx_status VX_API_CALL vxFinalizeKernel ( vx_kernel kernel )

This API is called after all parameters have been added to the kernel and the kernel is ready to be used. Notice that the reference to the kernel created by vxAddUserKernel is still valid after the call to vxFinalizeKernel. If an error occurs, the kernel is not available for usage by the clients of OpenVX. Typically this is due to a mismatch between the number of parameters requested and given.

Parameters

| in | kernel | The reference to the loaded kernel from vxAddUserKernel. |

Returns

A vx_status_e enumeration.

Return values

| VX_SUCCESS | No errors; any other value indicates failure. |
| VX_ERROR_INVALID_REFERENCE | kernel is not a valid vx_kernel reference. |

Precondition

vxAddUserKernel and vxAddParameterToKernel

vx_status VX_API_CALL vxAddParameterToKernel ( vx_kernel kernel, vx_uint32 index, vx_enum dir, vx_enum data_type, vx_enum state )

Allows users to set the signatures of the custom kernel.
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **in** *kernel* | The reference to the kernel added with `vxAddUserKernel`.
| **in** *index* | The index of the parameter to add. |
| **in** *dir* | The direction of the parameter. This must be either `VX_INPUT` or `VX_OUTPUT`. `VX_BIDIRECTIONAL` is not supported for this function. |
| **in** *data_type* | The type of parameter. This must be a value from `vx_type_e`.
| **in** *state* | The state of the parameter (required or not). This must be a value from `vx_parameter_state_e`.

**Returns**

A `vx_status_e` enumerated value.

**Return values**

<table>
<thead>
<tr>
<th><code>vx_status_e</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>VX_SUCCESS</code></td>
<td>Parameter is successfully set on kernel; any other value indicates failure.</td>
</tr>
<tr>
<td><code>VX_ERROR_INVALID_REFERENCE</code></td>
<td>Kernel is not a valid <code>vx_kernel</code> reference.</td>
</tr>
<tr>
<td><code>VX_ERROR_INVALID_PARAMETERS</code></td>
<td>If the parameter is not valid for any reason.</td>
</tr>
</tbody>
</table>

**Precondition**

`vxAddUserKernel`

**vx_status VX_API_CALL vxRemoveKernel ( vx_kernel kernel )**

Removes a custom kernel from its context and releases it.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **in** *kernel* | The reference to the kernel to remove. Returned from `vxAddUserKernel`.

**Note**

Any kernel enumerated in the base standard cannot be removed; only kernels added through `vxAddUserKernel` can be removed.

**Returns**

A `vx_status_e` enumeration. The function returns to the application full control over the memory resources provided at the kernel creation time.

**Return values**

<table>
<thead>
<tr>
<th><code>vx_status_e</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>VX_SUCCESS</code></td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td><code>VX_ERROR_INVALID_REFERENCE</code></td>
<td>Kernel is not a valid <code>vx_kernel</code> reference.</td>
</tr>
<tr>
<td><code>VX_ERROR_INVALID_PARAMETERS</code></td>
<td>If a base kernel is passed in.</td>
</tr>
<tr>
<td><code>VX_FAILURE</code></td>
<td>If the application has not released all references to the kernel object OR if the application has not released all references to a node that is using this kernel OR if the application has not released all references to a graph which has nodes that is using this kernel.</td>
</tr>
</tbody>
</table>
vx_status VX_API_CALL vxSetKernelAttribute ( vx_kernel kernel, vx_enum attribute, const void *ptr, vx_size size )

Sets kernel attributes.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel</td>
<td>The reference to the kernel.</td>
</tr>
<tr>
<td>attribute</td>
<td>The enumeration of the attributes. See vx_kernel_attribute_e.</td>
</tr>
<tr>
<td>ptr</td>
<td>The pointer to the location from which to read the attribute.</td>
</tr>
<tr>
<td>size</td>
<td>The size in bytes of the data area indicated by ptr in bytes.</td>
</tr>
</tbody>
</table>

Note

After a kernel has been passed to vxFinalizeKernel, no attributes can be altered.

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>kernel is not a valid vx_kernel reference.</td>
</tr>
</tbody>
</table>

vx_status VX_API_CALL vxSetMetaFormatAttribute ( vx_meta_format meta, vx_enum attribute, const void *ptr, vx_size size )

This function allows a user to set the attributes of a vx_meta_format object in a kernel output validator.

The vx_meta_format object contains two types of information: data object meta data and some specific information that defines how the valid region of an image changes.

The meta data attributes that can be set are identified by this list:

- vx_image : VX_IMAGE_FORMAT, VX_IMAGE_HEIGHT, VX_IMAGE_WIDTH
- vx_array : VX_ARRAY_CAPACITY, VX_ARRAY_ITEMTYPE
- vx_pyramid : VX_PYRAMID_FORMAT, VX_PYRAMID_HEIGHT, VX_PYRAMID_WIDTH, VX_PYRAMID_LEVELS, VX_PYRAMID_SCALE
- vx_scalar : VX_SCALAR_TYPE
- vx_matrix : VX_MATRIX_TYPE, VX_MATRIX_ROWS, VX_MATRIX_COLUMNS
- vx_distribution : VX_DISTRIBUTION_BINS, VX_DISTRIBUTION_OFFSET, VX_DISTRIBUTION_RANGE
- vx_remap : VX_REMAP_SOURCE_WIDTH, VX_REMAP_SOURCE_HEIGHT, VX_REMAP_DESTINATION_WIDTH, VX_REMAP_DESTINATION_HEIGHT
- vx_lut : VX_LUT_TYPE, VX_LUT_COUNT
- vx_threshold : VX_THRESHOLD_TYPE, VX_THRESHOLD_INPUT_FORMAT, VX_THRESHOLD_INPUT_FORMAT
- vx_object_array : VX_OBJECT_ARRAY_NUMITEMS, VX_OBJECT_ARRAY_ITEMTYPE
- vx_tensor : VX_TENSOR_NUMBER_OF_DIMS, VX_TENSOR_DIMS, VX_TENSOR_DATA_TYPE, VX_TENSOR_NUMBER_OF_DIMS, VX_TENSOR_DIMS, VX_TENSOR_DATA_TYPE, VX_TENSOR_DTYPE, VX_TENSOR_FIXED_POINT_POSITION
- VX_VALID_RECT_CALLBACK
Note

For vx_image, a specific attribute can be used to specify the valid region evolution. This information is not a meta data.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>meta</strong></td>
<td>The reference to the <code>vx_meta_format</code> struct to set</td>
</tr>
<tr>
<td><strong>attribute</strong></td>
<td>Use the subset of data object attributes that define the meta data of this object or attributes from <code>vx_meta_format</code>.</td>
</tr>
<tr>
<td><strong>ptr</strong></td>
<td>The input pointer of the value to set on the meta format object.</td>
</tr>
<tr>
<td><strong>size</strong></td>
<td>The size in bytes of the object to which <code>ptr</code> points.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>The attribute was set; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Meta is not a valid <code>vx_meta_format</code> reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>Size was not correct for the type needed.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUPPORTED</td>
<td>The object attribute was not supported on the meta format object.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_TYPE</td>
<td>Attribute type did not match known meta format type.</td>
</tr>
</tbody>
</table>

`vx_status VX_API_CALL vxSetMetaFormatFromReference ( vx_meta_format meta, vx_reference exemplar )`

Set a meta format object from an exemplar data object reference.

This function sets a `vx_meta_format` object from the meta data of the exemplar.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>meta</strong></td>
<td>The meta format object to set</td>
</tr>
<tr>
<td><strong>exemplar</strong></td>
<td>The exemplar data object.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>The meta format was correctly set; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>Meta is not a valid <code>vx_meta_format</code> reference, or exemplar is not a valid <code>vx_reference</code> reference.</td>
</tr>
</tbody>
</table>
3.95 Framework: Graph Parameters

3.95.1 Detailed Description

Defines the Graph Parameter API. Graph parameters allow Clients to create graphs with Client settable parameters. Clients can then create Graph creation methods (a.k.a. Graph Factories). When creating these factories, the client will typically not be able to use the standard Node creator functions such as \texttt{vxSobel3x3Node} but instead will use the manual method via \texttt{vxCreateGenericNode}.

```c
vx_graph vx_cornersGraphFactory(vx_context context)
{
    vx_status status = VX_SUCCESS;
    vx_uint32 i;
    vx_float32 strength_thresh = 10000.0f;
    vx_float32 r = 1.5f;
    vx_float32 sensitivity = 0.14f;
    vx_int32 window_size = 3;
    vx_int32 block_size = 3;
    vx_enum channel = VX_CHANNEL_Y;
    vx_graph graph = vxCreateGraph(context);
    // (vxGetStatus((vx_reference)graph) == VX_SUCCESS)
    {
        vx_image virts[] = {
            vxCreateVirtualImage(graph, 0, 0,
                VX_DF_IMAGE_VIRT),
            vxCreateVirtualImage(graph, 0, 0,
                VX_DF_IMAGE_VIRT),
        };
        vx_kernel kernels[] = {
            vxGetKernelByEnum(context,
                VX_KERNEL_CHANNEL_EXTRACT),
            vxGetKernelByEnum(context, VX_KERNEL_MEDIAN_3x3),
            vxGetKernelByEnum(context, VX_KERNEL_HARRIS_CORNERS),
        };
        vx_node nodes[dimof(kernels)] = {
            vxCreateGenericNode(graph, kernels[0]),
            vxCreateGenericNode(graph, kernels[1]),
            vxCreateGenericNode(graph, kernels[2]),
        };
        vx_scalar scalars[] = {
            vxCreateScalar(context, VX_TYPE_ENUM, &channel),
            vxCreateScalar(context, VX_TYPE_FLOAT32, &strength_thresh),
            vxCreateScalar(context, VX_TYPE_FLOAT32, &r),
            vxCreateScalar(context, VX_TYPE_FLOAT32, &sensitivity),
            vxCreateScalar(context, VX_TYPE_INT32, &window_size),
            vxCreateScalar(context, VX_TYPE_INT32, &block_size),
        };
        vx_parameter parameters[] = {
            vxGetParameterByIndex(nodes[0], 0),
            vxGetParameterByIndex(nodes[2], 6)
        };
        // Channel Extract
        status |= vxAddParameterToGraph(graph, parameters[0]);
        status |= vxSetParameterByIndex(nodes[0], 1, (vx_reference)scalars[0]);
        status |= vxSetParameterByIndex(nodes[0], 2, (vx_reference)virts[0]);
        // Median Filter
        status |= vxSetParameterByIndex(nodes[1], 0, (vx_reference)virts[0]);
        status |= vxSetParameterByIndex(nodes[1], 1, (vx_reference)virts[1]);
        // Harris Corners
        status |= vxSetParameterByIndex(nodes[2], 0, (vx_reference)virts[1]);
        status |= vxSetParameterByIndex(nodes[2], 1, (vx_reference)scalars[1]);
        status |= vxSetParameterByIndex(nodes[2], 2, (vx_reference)scalars[2]);
        status |= vxSetParameterByIndex(nodes[2], 3, (vx_reference)scalars[3]);
        status |= vxSetParameterByIndex(nodes[2], 4, (vx_reference)scalars[4]);
        status |= vxSetParameterByIndex(nodes[2], 5, (vx_reference)scalars[5]);
    }
    for (i = 0; i < dimof(scalars); i++)
    {
        vxReleaseScalar(&scalars[i]);
    }
    for (i = 0; i < dimof(virts); i++)
    {
        // Additional code...
    }
}
```
Some data are contained in these Graphs and do not become exposed to Clients of the factory. This allows ISVs or Vendors to create custom IP or IP-sensitive factories that Clients can use but may not be able to determine what is inside the factory. As the graph contains internal references to the data, the objects will not be freed until the graph itself is released.

Functions

- `vx_status VX_API_CALL vxAddParameterToGraph (vx_graph graph, vx_parameter parameter)`
  Adds the given parameter extracted from a `vx_node` to the graph.
- `vx_parameter VX_API_CALL vxGetGraphParameterByIndex (vx_graph graph, vx_uint32 index)`
  Retrieves a `vx_parameter` from a `vx_graph`.
- `vx_status VX_API_CALL vxSetGraphParameterByIndex (vx_graph graph, vx_uint32 index, vx_reference value)`
  Sets a reference to the parameter on the graph. The implementation must set this parameter on the originating node as well.

3.95.2 Function Documentation

`vx_status VX_API_CALL vxAddParameterToGraph (vx_graph graph, vx_parameter parameter)`

Adds the given parameter extracted from a `vx_node` to the graph.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The graph reference that contains the node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>parameter</td>
<td>The parameter reference to add to the graph from the node.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th><code>VX_SUCCESS</code></th>
<th>Parameter added to Graph; any other value indicates failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>VX_ERROR_INVALID_REFERENCE</code></td>
<td>graph is not a valid <code>vx_graph</code> reference or parameter is not a valid <code>vx_parameter</code> reference.</td>
</tr>
<tr>
<td><code>VX_ERROR_INVALID_PARAMETERS</code></td>
<td>The parameter is of a node not in this graph.</td>
</tr>
</tbody>
</table>
vx_status VX_API_CALL vxSetGraphParameterByIndex ( vx_graph graph, vx_uint32 index, vx_reference value )

Sets a reference to the parameter on the graph. The implementation must set this parameter on the originating node as well.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The graph reference.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>index</td>
<td>The parameter index.</td>
</tr>
<tr>
<td>in</td>
<td>value</td>
<td>The reference to set to the parameter.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>vx_status_e</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Parameter set to Graph; any other value indicates failure.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>graph is not a valid vx_graph reference or value is not a valid vx_reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>The parameter index is out of bounds or the dir parameter is incorrect.</td>
</tr>
</tbody>
</table>

vx_parameter VX_API_CALL vxGetGraphParameterByIndex ( vx_graph graph, vx_uint32 index )

Retrieves a vx_parameter from a vx_graph.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>index</td>
<td>The index of the parameter.</td>
</tr>
</tbody>
</table>

Returns

vx_parameter reference. Any possible errors preventing a successful function completion should be checked using vxGetStatus.
Bibliography
Absolute Difference, 35
  vxAbsDiffNode, 35
  vxuAbsDiff, 35
Accumulate, 37
  vxAccumulateImageNode, 37
  vxuAccumulateImage, 37
Accumulate Squared, 39
  vxAccumulateSquareImageNode, 39
  vxuAccumulateSquareImage, 39
Accumulate Weighted, 43
  vxAccumulateWeightedImageNode, 43
  vxuAccumulateWeightedImage, 43
Administrative Features, 286
Advanced Framework API, 314
Advanced Objects, 287
Arithmetic Addition, 48
  vxAddNode, 48
  vxuAdd, 48
Arithmetic Subtraction, 50
  vxSubtractNode, 50
  vxuSubtract, 50
Basic Features, 169
  VX_CHANNEL_0, 183
  VX_CHANNEL_1, 183
  VX_CHANNEL_2, 183
  VX_CHANNEL_3, 183
  VX_CHANNEL_A, 183
  VX_CHANNEL_B, 183
  VX_CHANNEL_G, 183
  VX_CHANNEL_R, 183
  VX_CHANNEL_U, 183
  VX_CHANNEL_V, 183
  VX_CHANNEL_Y, 183
  VX.ConvertPolicy_Saturate, 182
  VX.ConvertPolicy_Wrap, 182
  VX_DF_IMAGE_IYUV, 182
  VX_DF_IMAGE_NV12, 182
  VX_DF_IMAGE_NV21, 182
  VX_DF_IMAGE_RGBX, 182
  VX_DF_IMAGE_RGB, 182
  VX_DF_IMAGE_S16, 182
  VX_DF_IMAGE_S32, 182
  VX_DF_IMAGE_U16, 182
  VX_DF_IMAGE_U32, 182
  VX_DF_IMAGE_U8, 182
  VX_DF_IMAGE_UYYV, 182
  VX_DF_IMAGE_VIRT, 182
  VX_DF_IMAGE_YUV4, 182
  VX_DF_IMAGE_YUVV, 182
  VX_ENUM_ACCESSOR, 181
  VX_ENUM_ACTION, 181
  VX_ENUM_BASE, 177
  VX_ENUM_BORDER_POLICY, 181
  VX_ENUM_BORDER, 181
  VX_ENUM_CHANNEL, 181
  VX_ENUM_COLOR_RANGE, 181
  VX_ENUM_COLOR_SPACE, 181
  VX_ENUM_COMP_METRIC, 181
  VX_ENUM_COMPARISON, 181
  VX_ENUM_CONVERT_POLICY, 181
  VX_ENUM_DIRECTIONS, 181
  VX_ENUM_DIR, 181
  VX_ENUM_GRAPH_STATE, 181
  VX_ENUM_HINST, 181
  VX_ENUM_INTERPOLATION, 181
  VX_ENUM_LBP_FORMAT, 181
  VX_ENUM_MEMORY_TYPE, 181
  VX_ENUM_NONLINEAR, 181
  VX_ENUM_NORM_TYPE, 181
  VX_ENUM_OVERFLOW, 181
  VX_ENUM_PARAMETER_STATE, 181
  VX_ENUM_PATTERN, 181
  VX_ENUM_ROUND_POLICY, 181
  VX_ENUM_SCALAR_OPERATION, 181
  VX_ENUM_TARGET, 181
  VX_ENUM_TERM_CRITERIA, 181
  VX_ENUM_THRESHOLD_TYPE, 181
  VX_ERROR_GRAPH_ABANDONED, 180
  VX_ERROR_GRAPH_SCHEDULED, 180
  VX_ERROR_INVALID_FORMAT, 180
  VX_ERROR_INVALID_GRAPH, 180
  VX_ERROR_INVALID_LINK, 180
  VX_ERROR_INVALID_MODULE, 180
  VX_ERROR_INVALID_PARAMETERS, 180
  VX_ERROR_INVALID_PARAMETER_STATE, 180
  VX_ERROR_INVALID_PARAMETERS, 180
  VX_ERROR_INVALID_SCOPE, 180
  VX_ERROR_INVALID_VALUE, 180
  VX_ERROR_MULTIPLE_WRITERS, 180
  VX_ERROR_NO_MEMORY, 180
  VX_ERROR_NO_RESOURCES, 180
  VX_ERROR_NOTALLOCATED, 180
  VX_ERROR_NOT_COMPATIBLE, 180
  VX_ERROR_NOT_IMPLEMENTED, 181
  VX_ERROR_NOT_SUFFICIENT, 180
<table>
<thead>
<tr>
<th>VX_ERROR_NOT_SUPPORTED</th>
<th>VX_TYPE_DELAY</th>
<th>VX_TYPE_DF_IMAGE</th>
<th>VX_TYPE_DISTRIBUTION</th>
<th>VX_TYPE_ENUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_OPTIMIZED_AWAY</td>
<td>VX_TYPE_ERROR</td>
<td>VX_TYPE_FLOAT32</td>
<td>VX_TYPE_FLOAT64</td>
<td>VX_TYPE_ERROR</td>
</tr>
<tr>
<td>VX_ERROR_REFERENCE_NONZERO</td>
<td>VX_TYPE_FLOAT64</td>
<td>VX_TYPE_GRAPH</td>
<td>VX_TYPE_HOG_PARAMS</td>
<td>VX_TYPE_IMAGE</td>
</tr>
<tr>
<td>VX_FAILURE</td>
<td>VX_TYPE_HOUGH_LINES_PARAMS</td>
<td>VX_TYPE_IMAGE</td>
<td>VX_TYPE_INVALID</td>
<td>VX_TYPE_KERNEL</td>
</tr>
<tr>
<td>VX_ID_AMD</td>
<td>VX_TYPE_KEYPOINT</td>
<td>VX_TYPE_OBJECT_ARRAY</td>
<td>VX_TYPE_KEYPOINT</td>
<td>VX_TYPE_KHRONOS</td>
</tr>
<tr>
<td>VX_ID_ARM</td>
<td>VX_TYPE_VENDOR_OBJECT_END</td>
<td>VX_TYPE_KHRONOS_OBJECT_END</td>
<td>VX_TYPE_VENDOR_OBJECT_END</td>
<td>VX_TYPE_LUT</td>
</tr>
<tr>
<td>VX_ID_AXIS</td>
<td>VX_TYPE_VENDOR_OBJECT_START</td>
<td>VX_TYPE_LUT</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_LUT</td>
</tr>
<tr>
<td>VX_ID_BDTI</td>
<td>VX_TYPE_VENDOR_STRUCT_START</td>
<td>VX_TYPE_LINE_2D</td>
<td>VX_TYPE_VENDOR_STRUCT_START</td>
<td>VX_TYPE_LUT</td>
</tr>
<tr>
<td>VX_ID_BROADCOM</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_CADENCE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_CEVA</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_DEFAULT</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_Freescale</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_HUAWEI</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_IMAGINATION</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_INTEL</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_ITSEEZ</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_KHRONOS</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_MARVELL</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_MEDIATEK</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_MOVIDIUS</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_NVIDIA</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_NXP</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_QUALCOMM</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_RENESAS</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_SAMSUNG</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_SOCIONEXT</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_ST</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_TI</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_USER</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_VIDEANTIS</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_VIVANTE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_ID_XILINX</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_INTERPOLATION_AREA</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_INTERPOLATION_BILINEAR</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_INTERPOLATION_NEAREST_NEIGHBOR</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_NONLINEAR_FILTER_MAX</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_NONLINEAR_FILTER_MEDIAN</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_NONLINEAR_FILTER_MIN</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERNS_BOOLEAN</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERNS_CHOICE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERNS_COMPLEX</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERNS_ENUM</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERNS_FALSE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERNS_FUNCTION</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERNS_TRUE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERNS_VARIABLE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERNS_VARY</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERNS_WHEN</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERNS_WHILE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERN_BOX</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERN_CROSS</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERN_DISK</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERN_LINE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERN_POINTS</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERN_SQR</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
<tr>
<td>VX_PATTERN_LINES</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
<td>VX_TYPE_VENDOR_STRUCT_END</td>
<td>VX_TYPE_MAKESPACE</td>
</tr>
</tbody>
</table>
INDEX

vx_non_linear_filter_e, 184
vx_pattern_e, 184
vx_status, 177
vx_status_e, 179
vx_target_e, 182
vx_true_e, 178
vx_type_e, 178
vx_vendor_id_e, 184
vxGetStatus, 185

Bilateral Filter, 139
vxBilateralFilterNode, 139
vxuBilateralFilter, 140

Bitwise AND, 52
vxAndNode, 52
vxuAnd, 52

Bitwise EXCLUSIVE OR, 54
vxXorNode, 54
vxuXor, 54

Bitwise INCLUSIVE OR, 56
vxOrNode, 56
vxuOr, 56

Bitwise NOT, 58
vxNotNode, 58
vxuNot, 58

Box Filter, 60
vxBox3x3Node, 60
vxuBox3x3, 60

equalize

Canny Edge Detector, 64
VX_NORM_L1, 65
VX_NORM_L2, 65
vx_norm_type_e, 65
vxCannyEdgeDetectorNode, 65
vxuCannyEdgeDetector, 66

Channel Combine, 67
vxChannelCombineNode, 67
vxuChannelCombine, 67

Channel Extract, 69
vxChannelExtractNode, 69
vxuChannelExtract, 69

Color Convert, 71
vxColorConvertNode, 73
vxuColorConvert, 73

Control Flow, 45
vxScalarOperationNode, 46
vxSelectNode, 46

Convert Bit depth, 75
vxConvertDepthNode, 75
vxuConvertDepth, 76

Custom Convolution, 77
vxConvolveNode, 77
vxuConvolve, 78

Data Object Copy, 41
vxCopyNode, 41
vxuCopy, 41

Dilate Image, 79
vxDilate3x3Node, 79
vxuDilate3x3, 79

Equalize Histogram, 81
vxEqualizeHistNode, 81
vxuEqualizeHist, 81

Erode Image, 82
vxErode3x3Node, 82
vxuErode3x3, 82

Fast Corners, 84
vxFastCornersNode, 85
vxuFastCorners, 85

Framework: Directives, 324
VX_DIRECTIVE_DISABLE_LOGGING, 324
VX_DIRECTIVE_DISABLE_PERFORMANCE, 324
VX_DIRECTIVE_ENABLE_LOGGING, 324
VX_DIRECTIVE_ENABLE_PERFORMANCE, 324
vx_directive_e, 324
vxDirective, 324

Framework: Graph Parameters, 339
vxAddParameterToGraph, 340
vxGetGraphParameterByIndex, 341
vxSetGraphParameterByIndex, 340

Framework: Hints, 322
VX_HINT_PERFORMANCE_DEFAULT, 322
VX_HINT_PERFORMANCE_HIGH_SPEED, 322
VX_HINT_PERFORMANCE_LOW_POWER, 322
vx_hint_e, 322
vxHint, 322

Framework: Log, 320
vxAddLogEntry, 320
vxRegisterLogCallback, 321

Framework: Node Callbacks, 315
VX_ACTION_ABANDON, 317
VX_ACTION_CONTINUE, 317
vx_action, 317
vx_action_e, 317
vx_nodecomplete_f, 317
vxAssignNodeCallback, 317
vxRetrieveNodeCallback, 318

Framework: Performance Measurement, 319

Framework: User Kernels, 326
VX_VALID_RECT_CALLBACK, 332
vx_kernel_deinitialize_f, 331
vx_kernel_f, 330
vx_kernel_image_valid_rectangle_f, 331
vx_kernel_initialize_f, 331
vx_kernel_validate_f, 331
vx_meta_format, 330
vx_meta_valid_rect_attribute_e, 332
vx_publish_kernels_f, 330
vx_unpublish_kernels_f, 330
vxAddParameterToKernel, 335
vxAllocateUserKernelId, 332
vxAllocateUserKernelLibraryId, 333
vxFinalizeKernel, 335
vxLoadKernels, 333
vxRemoveKernel, 336
vxSetKernelAttribute, 336
vxSetMetaFormatAttribute, 337
INDEX

vxSetMetaFormatFromReference, 338
vxUnloadKernels, 334

Gaussian Filter, 87
vxGaussian3x3Node, 87
vxuGaussian3x3, 87

Gaussian Image Pyramid, 96
vxGaussianPyramidNode, 96
vxuGaussianPyramid, 96

HOG, 146
vxHOGCellsNode, 147
vxHOGFeaturesNode, 148
vxuHOGCells, 149
vxuHOGFeatures, 150

Harris Corners, 91
vxHarrisCornersNode, 92
vxuHarrisCorners, 93

Histogram, 94
vxHistogramNode, 94
vxuHistogram, 94

HoughLinesP, 152
vxHoughLinesPNode, 153
vxuHoughLinesP, 153

Integral Image, 102
vxIntegralImageNode, 102
vxuIntegralImage, 102

LBP, 143

vxLBPNode, 145
vxuLBP, 145

Laplacian Image Pyramid, 98
vxLaplacianPyramidNode, 98
vxuLaplacianPyramid, 99

Magnitude, 104
vxMagnitudeNode, 104
vxuMagnitude, 104

MatchTemplate, 141

vxMatchTemplateNode, 142
vxuMatchTemplate, 142

Max, 114
vxMaxNode, 114
vxuMax, 114

Mean and Standard Deviation, 106
vxMeanStdDevNode, 106
vxuMeanStdDev, 106

Median Filter, 108
vxMedian3x3Node, 108
vxuMedian3x3, 108

Min, 112
vxMinNode, 112
vxuMin, 112

Min, Max Location, 110
vxMinMaxLocNode, 110
vxuMinMaxLoc, 110

Node: Border Modes, 290

vx_border_e, 290
vx_border_policy_e, 290

Non Linear Filter, 89

vxNonLinearFilterNode, 89
vxuNonLinearFilter, 89

Non-Maxima Suppression, 62
vxNonMaxSuppressionNode, 62
vxuNonMaxSuppression, 62

Object: Array, 210

VX_ARRAY_CAPACITY, 211
VX_ARRAY_ITEMSIZE, 211
VX_ARRAY_NUMITEMS, 211
vx_array_attribute_e, 211
vxAddArrayItems, 213
vxArrayItem, 211
vxCopyArrayRange, 214
vxCreateArray, 211
vxCreateVirtualArray, 212
vxFormatArrayPointer, 211
vxMapArrayRange, 215
vxQueryArray, 213
vxReleaseArray, 212
vxTruncateArray, 214
vxUnmapArrayRange, 216

Object: Array (Advanced), 288
vxRegisterUserStruct, 288

Object: Context, 191

VX_CONTEXT_IMMEDIATE_BORDER, 193
VX_CONTEXT_IMMEDIATE_BORDER_POLICY, 193
VX_CONTEXT_MAX_TENSOR_DIMS, 193
VX_CONTEXT_NONLINEAR_MAX_DIMENSION, 193
VX_CONTEXT_CONVOLUTION_MAX_DIMENSION, 193
VX_CONTEXT_EXTENSIONS, 193
VX_CONTEXT_EXTENSIONS_SIZE, 193
VX_CONTEXT_EXTENSIONS, 193
VX_CONTEXT_CONVOLUTION_MAX_DIMENSION, 193
vxCreateImageFromROI, 232  
vxCreateUniformImage, 233  
vxCreateVirtualImage, 233  
vxFormatImagePatchAddress1d, 237  
vxFormatImagePatchAddress2d, 238  
vxGetValidRegionImage, 238  
vxMapImagePatch, 240  
vxQueryImage, 236  
vxReleaseImage, 237  
vxSetImageAttribute, 236  
vxSetImagePixelValues, 236  
vxSetImageValidRectangle, 242  
vxSwapImageHandle, 235  
vxUnmapImagePatch, 241

Object: Kernel, 296
  VX_KERNEL_ABSDIFF, 301  
  VX_KERNEL_ACCUMULATE_SQUARE, 302  
  VX_KERNEL_ACCUMULATE_WEIGHTED, 302  
  VX_KERNEL_ACCUMULATE, 302  
  VX_KERNEL_ADD, 303  
  VX_KERNEL_AND, 302  
  VX_KERNEL_BILATERAL_FILTER, 305  
  VX_KERNEL_BOX_3x3, 302  
  VX_KERNEL_CANNY_EDGE_DETECTOR, 302  
  VX_KERNEL_CHANNEL_COMBINE, 301  
  VX_KERNEL_CHANNEL_EXTRACT, 301  
  VX_KERNEL_COLOR_CONVERT, 300  
  VX_KERNEL_CONVERTDEPTH, 302  
  VX_KERNEL_COPY, 304  
  VX_KERNEL_CUSTOM_CONVOLUTION, 302  
  VX_KERNEL_DILATE_3x3, 302  
  VX_KERNEL_ENUM, 305  
  VX_KERNEL_EQUALIZE_HISTOGRAM, 301  
  VX_KERNEL_ERODE_3x3, 302  
  VX_KERNEL_FAST_CORNERS, 303  
  VX_KERNEL_GAUSSIAN_3x3, 302  
  VX_KERNEL_GAUSSIAN_PYRAMID, 302  
  VX_KERNEL_HALFSCALE_GAUSSIAN, 303  
  VX_KERNEL_HARRIS_CORNERS, 303  
  VX_KERNEL_HISTOGRAM, 301  
  VX_KERNEL_HOG_CELLS, 305  
  VX_KERNEL_HOG_FEATURES, 305  
  VX_KERNEL_HOUGH_LINES_P, 304  
  VX_KERNEL_INTEGRAL_IMAGE, 301  
  VX_KERNEL_LAPLACIAN_PYRAMID, 303  
  VX_KERNEL_LAPLACIAN_RECONSTRUCT, 304  
  VX_KERNEL_LBP, 304  
  VX_KERNEL_LOCAL_DATA_SIZE, 305  
  VX_KERNEL_MAGNITUDE, 301  
  VX_KERNEL_MATCH_TEMPLATE, 304  
  VX_KERNEL_MAX, 305  
  VX_KERNEL_MEAN_STDDEV, 301  
  VX_KERNEL_MEDIAN_3x3, 302  
  VX_KERNEL_MINMAXLOC, 302  
  VX_KERNEL_MIN, 305  
  VX_KERNEL_MULTIPLY, 303  
  VX_KERNEL_NAME, 305  
  VX_KERNEL_NON_LINEAR_FILTER, 304  
  VX_KERNEL_NON_MAX_SUPPRESSION, 304  
  VX_KERNEL_NOT, 303  
  VX_KERNEL_OPTICAL_FLOW_PYR_LK, 303  
  VX_KERNEL_OR, 303  
  VX_KERNEL_PARAMETERS, 305  
  VX_KERNEL_PHASE, 301  
  VX_KERNEL_REMAP, 303  
  VX_KERNEL_SCALAR_OPERATION, 305  
  VX_KERNEL_SCALE_IMAGE, 301  
  VX_KERNEL_SELECT, 305  
  VX_KERNEL_SOBEL_3x3, 301  
  VX_KERNEL_SUBTRACT, 303  
  VX_KERNEL_TABLE_LOOKUP, 301  
  VX_KERNEL_TENSOR_ADD, 304  
  VX_KERNEL_TENSOR_CONVERT_DEPTH, 304  
  VX_KERNEL_TENSOR_MATRIX_MULTIPLY, 304  
  VX_KERNEL_TENSOR_MULTIPLY, 304  
  VX_KERNEL_TENSOR_SUBTRACT, 304  
  VX_KERNEL_TENSOR_TABLE_LOOKUP, 304  
  VX_KERNEL_TENSOR_TRANSPOSE, 304  
  VX_KERNEL_THRESHOLD, 301  
  VX_KERNEL_WARP_AFFINE, 303  
  VX_KERNEL_WARP_PERSPECTIVE, 303  
  VX_KERNEL_XOR, 303  
  VX_LIBRARY_KHR_BASE, 300

Object: LUT, 244
  VX_LUT_COUNT, 244  
  VX_LUT_OFFSET, 244  
  VX_LUT_SIZE, 244  
  VX_LUT_TYPE, 244  
  vx_lut_attribute_e, 244  
  vxCopyLUT, 246  
  vxCreateLUT, 245  
  vxCreateVirtualLUT, 245  
  vxMapLUT, 247  
  vxQueryLUT, 246  
  vxReleaseLUT, 245  
  vxUnmapLUT, 248

Object: Matrix, 250
  VX_MATRIX_COLUMNS, 250  
  VX_MATRIX_ORIGIN, 250  
  VX_MATRIX_PATTERN, 251  
  VX_MATRIX_ROWS, 250  
  VX_MATRIX_SIZE, 250  
  VX_MATRIX_TYPE, 250  
  vx_matrix_attribute_e, 250  
  vxCopyMatrix, 252  
  vxCreateMatrix, 251  
  vxCreateMatrixFromPattern, 253  
  vxCreateMatrixFromPatternAndOrigin, 253
vxCreateVirtualMatrix, 251
vxQueryMatrix, 252
vxReleaseMatrix, 251
Object: Node, 205
    VX_NODE_BORDER, 206
    VX_NODE_IS_REPLICATED, 206
    VX_NODE_LOCAL_DATA_PTR, 206
    VX_NODE_LOCAL_DATA_SIZE, 206
    VX_NODE_PARAMETERS, 206
    VX_NODE_PERFORMANCE, 206
    VX_NODE_STATUS, 206
    VX_NODE_VALID_RECT_RESET, 206
    vx_node, 206
    vx_node_attribute_e, 206
    vxQueryNode, 206
    vxReleaseNode, 207
    vxRemoveNode, 208
    vxReplicateNode, 208
    vxSetNodeAttribute, 207
    vxSetNodeTarget, 208
Object: Node (Advanced), 289
    vxCreateGenericNode, 289
Object: ObjectArray, 278
    VX_OBJECT_ARRAY_ITEMTYPE, 278
    VX_OBJECT_ARRAY_NUMITEMS, 278
    vx_object_array_attribute_e, 278
    vxCreateObjectArray, 279
    vxCreateVirtualObjectArray, 279
    vxGetObjectArrayItem, 279
    vxQueryObjectArray, 280
    vxReleaseObjectArray, 279
Object: Parameter, 309
    VX_BIDIRECTIONAL, 310
    VX_INPUT, 310
    VX_OUTPUT, 310
    VX_PARAMETER_DIRECTION, 310
    VX_PARAMETER_INDEX, 310
    VX_PARAMETER_REF, 310
    VX_PARAMETER_STATE_OPTIONAL, 310
    VX_PARAMETER_STATE_REQUIRED, 310
    VX_PARAMETER_STATE, 310
    VX_PARAMETER_TYPE, 310
    vx_direction_e, 310
    vx_parameter, 310
    vx_parameter_attribute_e, 310
    vx_parameter_state_e, 310
    vxGetKernelParameterByIndex, 310
    vxGetParameterByIndex, 311
    vxQueryParameter, 312
    vxReleaseParameter, 311
    vxSetParameterByIndex, 311
    vxSetParameterByReference, 312
Object: Pyramid, 255
    VX_PYRAMID_FORMAT, 256
    VX_PYRAMID_HEIGHT, 256
    VX_PYRAMID_LEVELS, 256
    VX_PYRAMID_SCALE, 256
    VX_PYRAMID_WIDTH, 256
    vx_pyramid_attribute_e, 256
    vxCreatePyramid, 256
    vxCreateVirtualPyramid, 256
    vxGetPyramidLevel, 258
    vxQueryPyramid, 257
    vxReleasePyramid, 257
Object: Reference, 187
    VX_MAX_REFERENCE_NAME, 187
    VX_REFERENCE_COUNT, 188
    VX_REFERENCE_NAME, 188
    VX_REFERENCE_TYPE, 188
    vx_reference, 188
    vx_reference_attribute_e, 188
    vxQueryReference, 188
    vxReleaseReference, 188
    vxRetainReference, 189
    vxSetReferenceName, 189
Object: Remap, 259
    VX_REMAP_DESTINATION_HEIGHT, 259
    VX_REMAP_DESTINATION_WIDTH, 259
    VX_REMAP_SOURCE_HEIGHT, 259
    VX_REMAP_SOURCE_WIDTH, 259
    vx_remap_attribute_e, 259
    vxCopyRemapPatch, 263
    vxCreateRemap, 260
    vxCreateVirtualRemap, 260
    vxMapRemapPatch, 261
    vxQueryRemap, 264
    vxReleaseRemap, 260
    vxUnmapRemapPatch, 262
Object: Scalar, 265
    VX_SCALAR_OP_ADD, 267
    VX_SCALAR_OP_AND, 266
    VX_SCALAR_OP_DIVIDE, 267
    VX_SCALAR_OP_EQUAL, 266
    VX_SCALAR_OP_GREATEREQ, 267
    VX_SCALAR_OP_GREATER, 266
    VX_SCALAR_OP_LESSEQ, 266
    VX_SCALAR_OP_LESS, 266
    VX_SCALAR_OP_MAX, 267
    VX_SCALAR_OP_MIN, 267
    VX_SCALAR_OP_MODULUS, 267
    VX_SCALAR_OP_MULTIPLY, 267
    VX_SCALAR_OP_NAND, 266
    VX_SCALAR_OP_NOTEQUAL, 266
    VX_SCALAR_OP_OR, 266
    VX_SCALAR_OP_SUBTRACT, 267
    VX_SCALAR_OP_XOR, 266
    VX_SCALAR_TYPE, 266
    vx_scalar, 266
    vx_scalar_attribute_e, 266
    vx_scalar_operation_e, 266
    vxCopyScalar, 269
    vxCopyScalarWithSize, 269
    vxCreateScalar, 267
    vxCreateScalarWithSize, 267
    vxCreateVirtualScalar, 267
vxQueryScalar, 268
vxReleaseScalar, 268

Object: Tensor, 281
  VX_TENSOR_DATA_TYPE, 282
  VX_TENSOR_DIMS, 282
  VX_TENSOR_FIXED_POINT_POSITION, 282
  VX_TENSOR_NUMBER_OF_DIMS, 282
  vx_tensor, 282
  vx_tensor_attribute_e, 282
  vxCopyTensorPatch, 284
  vxCreateImageObjectArrayFromTensor, 283
  vxCreateTensor, 282
  vxCreateTensorFromView, 283
  vxCreateVirtualTensor, 283
  vxQueryTensor, 285
  vxReleaseTensor, 285

Object: Threshold, 271
  VX_THRESHOLD_INPUT_FORMAT, 272
  VX_THRESHOLD_OUTPUT_FORMAT, 272
  VX_THRESHOLD_TYPE_BINARY, 272
  VX_THRESHOLD_TYPE_RANGE, 272
  VX_THRESHOLD_TYPE, 272
  vx_threshold_attribute_e, 272
  vx_threshold_type_e, 272
  vxCopyThresholdOutput, 275
  vxCopyThresholdRange, 274
  vxCopyThresholdValue, 273
  vxCreateThresholdForImage, 272
  vxCreateVirtualThresholdForImage, 273
  vxQueryThreshold, 277
  vxReleaseThreshold, 276
  vxSetThresholdAttribute, 276

Objects, 186
Optical Flow Pyramid (LK), 116
  vxOpticalFlowPyrLKNode, 117
  vxuOpticalFlowPyrLK, 117

Phase, 119
  vxPhaseNode, 119
  vxuPhase, 119

Pixel-wise Multiplication, 121
  vxMultiplyNode, 121
  vxuMultiply, 122

Reconstruction from a Laplacian Image Pyramid, 100
  vxLaplacianReconstructNode, 100
  vxuLaplacianReconstruct, 100

Remap, 123
  vxRemapNode, 123
  vxuRemap, 123

Scale Image, 125
  vxHalfScaleGaussianNode, 127
  vxScaleImageNode, 127
  vxuHalfScaleGaussian, 128
  vxuScaleImage, 128

Sobel 3x3, 129
  vxSobel3x3Node, 129
  vxuSobel3x3, 129

TableLookup, 131
  vxTableLookupNode, 131
  vxuTableLookup, 131

Tensor Add, 157
  vxTensorAddNode, 157
  vxuTensorAdd, 157

Tensor Convert Bit-Depth, 165
  vxTensorConvertDepthNode, 165
  vxuTensorConvertDepth, 165

Tensor Multiply, 155
  vxTensorMultiplyNode, 155
  vxuTensorMultiply, 155

Tensor Subtract, 159
  vxTensorSubtractNode, 159
  vxuTensorSubtract, 159

Thresholding, 133
  vxThresholdNode, 133
  vxuThreshold, 134

VX_ACTION_ABANDON
  Framework: Node Callbacks, 317

VX_ACTION_CONTINUE
  Framework: Node Callbacks, 317

VX_ARRAY_CAPACITY
  Object: Array, 211

VX_ARRAY_ITEMSIZE
  Object: Array, 211

VX_ARRAY_ITEMTYPE
  Object: Array, 211

VX_BIDIRECTIONAL
  Object: Parameter, 310

VX_BORDER_CONSTANT
  Node: Border Modes, 290

VX_BORDER_POLICY_DEFAULT_TO_UNDEFINED
  Node: Border Modes, 291

VX_BORDER_POLICY_RETURN_ERROR
  Node: Border Modes, 291

VX_CHANNEL_0
  Basic Features, 183

VX_CHANNEL_1
  Basic Features, 183

VX_CHANNEL_2
  Basic Features, 183

VX_CHANNEL_3
INDEX

Basic Features, 183
VX_CHANNEL_RANGE_FULL
Object: Image, 232
VX_CHANNEL_RANGE_RESTRICTED
Object: Image, 232
VX_CHANNEL_A
Basic Features, 183
VX_CHANNEL_B
Basic Features, 183
VX_CHANNEL_G
Basic Features, 183
VX_CHANNEL_R
Basic Features, 183
VX_CHANNEL_U
Basic Features, 183
VX_CHANNEL_V
Basic Features, 183
VX_CHANNEL_Y
Basic Features, 183
VX_COLOR_SPACE_BT601_525
Object: Image, 231
VX_COLOR_SPACE_BT601_625
Object: Image, 232
VX_COLOR_SPACE_BT709
Object: Image, 232
VX_COLOR_SPACE_DEFAULT
Object: Image, 232
VX_COLOR_SPACE_NONE
Object: Image, 231
VX_COMPARE_CCORR_NORM
MatchTemplate, 141
VX_COMPARE_CCORR
MatchTemplate, 141
VX_COMPARE_HAMMING
MatchTemplate, 141
VX_COMPARE_L1
MatchTemplate, 141
VX_COMPARE_L2
MatchTemplate, 141
VX_COMPARE_L2_NORM
MatchTemplate, 141
VX_CONTEXT_CONVOLUTION_MAX_DIMENSION
Object: Context, 193
VX_CONTEXT_CONVOLUTION_COLUMNS
Object: Convolution, 218
VX_CONTEXT_CONVOLUTION_ROWS
Object: Convolution, 218
VX_CONTEXT_CONVOLUTION_SCALE
Object: Convolution, 218
VX_CONTEXT_CONVOLUTION_SIZE
Object: Convolution, 219
VX_DELAY_SLOTS
Object: Delay, 293
VX_DELAY_TYPE
Object: Delay, 293
VX_DF_IMAGE_IYUV
Basic Features, 182
VX_DF_IMAGE_NV12
Basic Features, 182
VX_DF_IMAGE_NV21
Basic Features, 182
VX_DF_IMAGE_RGBX
Basic Features, 182
VX_DF_IMAGE_RBG
Basic Features, 182
VX_DF_IMAGE_S16
Basic Features, 182
VX_DF_IMAGE_S32
Basic Features, 182
VX_DF_IMAGE_U16
Basic Features, 182
VX_DF_IMAGE_U32
Basic Features, 182
VX_DF_IMAGE_U8
Basic Features, 182
VX_DF_IMAGE_UYYV
Basic Features, 182
VX_DF_IMAGE_VIRT
Basic Features, 182
VX_DF_IMAGE_Y4U4
Basic Features, 182
VX_DF_IMAGE_YUV4
Basic Features, 182
VX_DF_IMAGE_YUVV
Basic Features, 182
INDEX

VX_DF_IMAGE
   Basic Features, 177

VX_DIRECTIVE_DISABLE_LOGGING
   Framework: Directives, 324

VX_DIRECTIVE_DISABLE_PERFORMANCE
   Framework: Directives, 324

VX_DIRECTIVE_ENABLE_LOGGING
   Framework: Directives, 324

VX_DIRECTIVE_ENABLE_PERFORMANCE
   Framework: Directives, 324

VX_DISTRIBUTION_BINS
   Object: Distribution, 223

VX_DISTRIBUTION_DIMENSIONS
   Object: Distribution, 223

VX_DISTRIBUTION_OFFSET
   Object: Distribution, 223

VX_DISTRIBUTION_RANGE
   Object: Distribution, 223

VX_DISTRIBUTION_SIZE
   Object: Distribution, 223

VX_DISTRIBUTION_WINDOW
   Object: Distribution, 223

VX_ENUM_ACCESSOR
   Basic Features, 181

VX_ENUM_ACTION
   Basic Features, 181

VX_ENUM_BASE
   Basic Features, 177

VX_ENUM_BORDER_POLICY
   Basic Features, 181

VX_ENUM_BORDER
   Basic Features, 181

VX_ENUM_CHANNEL
   Basic Features, 181

VX_ENUM_COLOR_RANGE
   Basic Features, 181

VX_ENUM_COLOR_SPACE
   Basic Features, 181

VX_ENUM_COMP_METRIC
   Basic Features, 181

VX_ENUM_COMPARISON
   Basic Features, 181

VX_ENUM_CONVERT_POLICY
   Basic Features, 181

VX_ENUM_DIRECTION
   Basic Features, 181

VX_ENUM_DIRECTIVE
   Basic Features, 181

VX_ENUM_GRAPH_STATE
   Basic Features, 181

VX_ENUM_HINT
   Basic Features, 181

VX_ENUM_INTERPOLATION
   Basic Features, 181

VX_ENUM_LBP_FORMAT
   Basic Features, 181

VX_ENUM_MEMORY_TYPE
   Basic Features, 181

VX_ENUM_NONLINEAR
   Basic Features, 181

VX_ENUM_Normal_Type
   Basic Features, 181

VX_ENUM_OVERFLOW
   Basic Features, 181

VX_ENUM_PARAMETER_STATE
   Basic Features, 181

VX_ENUM_PATTERN
   Basic Features, 181

VX_ENUM_ROUND_POLICY
   Basic Features, 181

VX_ENUM_SCALAR_OPERATION
   Basic Features, 181

VX_ENUM_TARGET
   Basic Features, 181

VX_ENUM_TERM_CRITERIA
   Basic Features, 181

VX_ENUM_THRESHOLD_TYPE
   Basic Features, 181

VX_ERROR_GRAPH_ABANDONED
   Basic Features, 180

VX_ERROR_GRAPH_SCHEDULED
   Basic Features, 180

VX_ERROR_INVALID_DIMENSION
   Basic Features, 180

VX_ERROR_INVALID_FORMAT
   Basic Features, 180

VX_ERROR_INVALID_GRAPH
   Basic Features, 180

VX_ERROR_INVALID_LINK
   Basic Features, 180

VX_ERROR_INVALID_MODULE
   Basic Features, 180

VX_ERROR_INVALID_NODE
   Basic Features, 180

VX_ERROR_INVALID_PARAMETERS
   Basic Features, 180

VX_ERROR_INVALID_REFERENCE
   Basic Features, 180

VX_ERROR_INVALID_SCOPE
   Basic Features, 180

VX_ERROR_INVALID_TYPE
   Basic Features, 180

VX_ERROR_INVALID_VALUE
   Basic Features, 180

VX_ERROR_MULTIPLE_WRITERS
   Basic Features, 180

VX_ERROR_NO_MEMORY
   Basic Features, 180

VX_ERROR_NO_RESOURCES
   Basic Features, 180

VX_ERROR_NOT_ALLOCATED
   Basic Features, 180

VX_ERROR_NOT_COMPATIBLE
   Basic Features, 180

VX_ERROR_NOT_IMPLEMENTED
   Basic Features, 181
VX_ERROR_NOT_SUFFICIENT
Basic Features, 180

VX_ERROR_NOT_SUPPORTED
Basic Features, 181

VX_ERROR_OPTIMIZED_AWAY
Basic Features, 180

VX_ERROR_REFERENCE_NONZERO
Basic Features, 180

VX_FAILURE
Basic Features, 181

VX_GRAPH_NUMNODES
Object: Graph, 200

VX_GRAPH_NUMPARAMETERS
Object: Graph, 200

VX_GRAPH_PERFORMANCE
Object: Graph, 200

VX_GRAPH_STATE_ABANDONED
Object: Graph, 200

VX_GRAPH_STATE_COMPLETED
Object: Graph, 200

VX_GRAPH_STATE_RUNNING
Object: Graph, 199

VX_GRAPH_STATE_UNVERIFIED
Object: Graph, 199

VX_GRAPH_STATE_VERIFIED
Object: Graph, 199

VX_GRAPH_STATE
Object: Graph, 200

VX_HINT_PERFORMANCE_DEFAULT
Framework: Hints, 322

VX_HINT_PERFORMANCE_HIGH_SPEED
Framework: Hints, 322

VX_HINT_PERFORMANCE_LOW_POWER
Framework: Hints, 322

VX_ID_AMD
Basic Features, 184

VX_ID_ARM
Basic Features, 184

VX_ID_AXIS
Basic Features, 184

VX_ID_BDTI
Basic Features, 184

VX_ID_BROADCOM
Basic Features, 184

VX_ID_CADENCE
Basic Features, 185

VX_ID_CEVA
Basic Features, 185

VX_ID_DEFAULT
Basic Features, 185

VX_ID_FREESCALE
Basic Features, 184

VX_ID_HUAWEI
Basic Features, 185

VX_ID_IMAGINATION
Basic Features, 185

VX_ID_INTEL
Basic Features, 184

VX_ID_ITSEEZ
Basic Features, 185

VX_ID_KHRONOS
Basic Features, 184

VX_ID_MARVELL
Basic Features, 184

VX_ID_MEDIATEK
Basic Features, 185

VX_ID_MOVIDIUS
Basic Features, 184

VX_ID_MEDIATEK
Basic Features, 185

VX_ID_NVIDIA
Basic Features, 184

VX_ID_NXP
Basic Features, 185

VX_ID_QUALCOMM
Basic Features, 184

VX_ID_RENESAS
Basic Features, 184

VX_ID_SAMSUNG
Basic Features, 184

VX_ID_SOCIONEXT
Basic Features, 185

VX_ID_SYNOPSYS
Basic Features, 185

VX_ID_ST
Basic Features, 185

VX_ID_TI
Basic Features, 184

VX_ID_USER
Basic Features, 185

VX_ID_VIDEANTIS
Basic Features, 185

VX_ID_VIVANTE
Basic Features, 184

VX_ID_XILINX
Basic Features, 184

VX_IMAGE_FORMAT
Object: Image, 231

VX_IMAGE_HEIGHT
Object: Image, 231

VX_IMAGE_IS_UNIFORM
Object: Image, 231

VX_IMAGE_MEMORY_TYPE
Object: Image, 231

VX_IMAGE_PLANES
Object: Image, 231

VX_IMAGE_RANGE
Object: Image, 231

VX_IMAGE_SPACE
Object: Image, 231

VX_IMAGE_UNIFORM_VALUE
Object: Image, 231

VX_IMAGE_WIDTH
Object: Image, 231

VX_INTERPOLATION_AREA
Basic Features, 184
INDEX

VX_INTERPOLATION_BILINEAR
Basic Features, 183

VX_INTERPOLATION_NEAREST_NEIGHBOR
Basic Features, 183

VX_KERNEL_ABSDIFF
Object: Kernel, 301

VX_KERNEL_ACCUMULATE_SQUARE
Object: Kernel, 302

VX_KERNEL_ACCUMULATE_WEIGHTED
Object: Kernel, 302

VX_KERNEL_ACCUMULATE
Object: Kernel, 302

VX_KERNEL_ADD
Object: Kernel, 303

VX_KERNEL_AND
Object: Kernel, 303

VX_KERNEL_BILATERAL_FILTER
Object: Kernel, 305

VX_KERNEL_BOX_3x3
Object: Kernel, 302

VX_KERNEL_CANNY_EDGE_DETECTOR
Object: Kernel, 302

VX_KERNEL_CHANNEL_COMBINE
Object: Kernel, 301

VX_KERNEL_CHANNEL_EXTRACT
Object: Kernel, 301

VX_KERNEL_COLOR_CONVERT
Object: Kernel, 300

VX_KERNEL_CONVERTDEPTH
Object: Kernel, 302

VX_KERNEL_COPY
Object: Kernel, 304

VX_KERNEL_CUSTOM_CONVOLUTION
Object: Kernel, 302

VX_KERNEL_DILATE_3x3
Object: Kernel, 302

VX_KERNEL_ENUM
Object: Kernel, 305

VX_KERNEL_EQUALIZE_HISTOGRAM
Object: Kernel, 301

VX_KERNEL_ERODE_3x3
Object: Kernel, 302

VX_KERNEL_ENUM
Object: Kernel, 305

VX_KERNEL_EQUATE_PYRAMID
Object: Kernel, 302

VX_KERNEL_FAST_CORNERS
Object: Kernel, 303

VX_KERNEL_GREYSCALE
Object: Kernel, 303

VX_KERNEL_GAUSSIAN_3x3
Object: Kernel, 302

VX_KERNEL_HALFSCALE_GAUSSIAN
Object: Kernel, 303

VX_KERNEL_HARRIS_CORNERS
Object: Kernel, 303

VX_KERNEL_HISTOGRAM
Object: Kernel, 301

VX_KERNEL_HOG_CELLS
Object: Kernel, 305

VX_KERNEL_HOG_FEATURES
Object: Kernel, 305

VX_KERNEL_HOUGH_LINES_P
Object: Kernel, 304

VX_KERNEL_INTEGRAL_IMAGE
Object: Kernel, 301

VX_KERNEL_LAPLACIAN_PYRAMID
Object: Kernel, 303

VX_KERNEL_LAPLACIAN_RECONSTRUCT
Object: Kernel, 304

VX_KERNEL_LBP
Object: Kernel, 304

VX_KERNEL_LOCAL_DATA_SIZE
Object: Kernel, 305

VX_KERNEL_MAGNITUDE
Object: Kernel, 301

VX_KERNEL_MATCH_TEMPLATE
Object: Kernel, 304

VX_KERNEL_MAX
Object: Kernel, 305

VX_KERNEL_MEAN_STDDEV
Object: Kernel, 301

VX_KERNEL_MEDIAN_3x3
Object: Kernel, 302

VX_KERNEL_MINMAXLOC
Object: Kernel, 302

VX_KERNEL_MIN
Object: Kernel, 305

VX_KERNEL_MULTIPLY
Object: Kernel, 303

VX_KERNEL_NAME
Object: Kernel, 305

VX_KERNEL_NON_LINEAR_FILTER
Object: Kernel, 304

VX_KERNEL_NON_MAX_SUPPRESSION
Object: Kernel, 304

VX_KERNEL_NOT
Object: Kernel, 303

VX_KERNEL_OPTICAL_FLOW_PYR_LK
Object: Kernel, 303

VX_KERNEL_OR
Object: Kernel, 303

VX_KERNEL_PARAMETERS
Object: Kernel, 305

VX_KERNEL_PHASE
Object: Kernel, 301

VX_KERNEL_REMAP
Object: Kernel, 303

VX_KERNEL_SCALAR_OPERATION
Object: Kernel, 305

VX_KERNEL_SCALE_IMAGE
Object: Kernel, 301

VX_KERNEL_SELECT
Object: Kernel, 305

VX_KERNEL_SOBE_3x3
Object: Kernel, 301

VX_KERNEL_SUBTRACT
Object: Kernel, 303

VX_KERNEL_TABLE_LOOKUP
Object: Kernel, 301
INDEX

VX_KERNEL_TENSOR_ADD
   Object: Kernel, 304
VX_KERNEL_TENSOR_CONVERT_DEPTH
   Object: Kernel, 304
VX_KERNEL_TENSOR_MATRIX_MULTIPLY
   Object: Kernel, 304
VX_KERNEL_TENSOR_MULTIPLY
   Object: Kernel, 304
VX_KERNEL_TENSOR_SUBTRACT
   Object: Kernel, 304
VX_KERNEL_TENSOR_TABLE_LOOKUP
   Object: Kernel, 304
VX_KERNEL_TENSOR_TRANSPOSE
   Object: Kernel, 304
VX_KERNEL_THRESHOLD
   Object: Kernel, 301
VX_KERNEL_WARP_AFFINE
   Object: Kernel, 303
VX_KERNEL_WARP_PERSPECTIVE
   Object: Kernel, 303
VX_KERNEL_XOR
   Object: Kernel, 303
VX_LBP
   LBP, 144
VX_LIBRARY_KHR_BASE
   Object: Kernel, 300
VX_LUT_COUNT
   Object: LUT, 244
VX_LUT_OFFSET
   Object: LUT, 244
VX_LUT_SIZE
   Object: LUT, 244
VX_LUT_TYPE
   Object: LUT, 244
VX_MATRIX_COLUMNS
   Object: Matrix, 250
VX_MATRIX_ORIGIN
   Object: Matrix, 250
VX_MATRIX_PATTERN
   Object: Matrix, 251
VX_MATRIX_ROWS
   Object: Matrix, 250
VX_MATRIX_SIZE
   Object: Matrix, 250
VX_MATRIX_TYPE
   Object: Matrix, 250
VX_MAX_REFERENCE_NAME
   Object: Reference, 187
VX_MEMORY_TYPE_HOST
   Object: Context, 194
VX_MEMORY_TYPE_NONE
   Object: Context, 194
VX_MLBP
   LBP, 144
VX_NODE_BORDER
   Object: Node, 206
VX_NODE_IS_REPLICATED
   Object: Node, 206
VX_NODE_LOCAL_DATA_PTR
   Object: Node, 206
VX_NODE_LOCAL_DATA_SIZE
   Object: Node, 206
VX_NODE_PARAMETERS
   Object: Node, 206
VX_NODE_PERFORMANCE
   Object: Node, 206
VX_NODE_REPLICATE_FLAGS
   Object: Node, 206
VX_NODE_STATUS
   Object: Node, 206
VX_NODE_VALID_RECT_RESET
   Object: Node, 206
VX_NOGAP_X
   Object: Image, 232
VX_NONLINEAR_FILTER_MAX
   Basic Features, 184
VX_NONLINEAR_FILTER_MEDIAN
   Basic Features, 184
VX_NONLINEAR_FILTER_MIN
   Basic Features, 184
VX_NORM_L1
   Canny Edge Detector, 65
VX_NORM_L2
   Canny Edge Detector, 65
VX_OBJECT_ARRAY_ITEMTYPE
   Object: ObjectArray, 278
VX_OBJECT_ARRAY_NUMITEMS
   Object: ObjectArray, 278
VX_OUTPUT
   Object: Parameter, 310
VX_PARAMETER_DIRECTION
   Object: Parameter, 310
VX_PARAMETER_INDEX
   Object: Parameter, 310
VX_PARAMETER_REF
   Object: Parameter, 310
VX_PARAMETER_STATE_OPTIONAL
   Object: Parameter, 310
VX_PARAMETER_STATE_REQUIRED
   Object: Parameter, 310
VX_PARAMETER_STATE
   Object: Parameter, 310
VX_PATTERN_BOX
   Basic Features, 184
VX_PATTERN_CROSS
   Basic Features, 184
VX_PATTERN_DISK
   Basic Features, 184
VX_PATTERN_OTHER
   Basic Features, 184
VX_PYRAMID_FORMAT
   Object: Pyramid, 256
VX_PYRAMID_HEIGHT
   Object: Pyramid, 256
INDEX

VX_PYRAMID_LEVELS
Object: Pyramid, 256

VX_PYRAMID_SCALE
Object: Pyramid, 256

VX_PYRAMID_WIDTH
Object: Pyramid, 256

VX_READ_AND_WRITE
Object: Context, 194

VX_READ_ONLY
Object: Context, 194

VX_REFERENCE_COUNT
Object: Reference, 188

VX_REFERENCE_NAME
Object: Reference, 188

VX_REFERENCE_TYPE
Object: Reference, 188

VX_REMAP_DESTINATION_HEIGHT
Object: Remap, 259

VX_REMAP_DESTINATION_WIDTH
Object: Remap, 259

VX_REMAP_SOURCE_HEIGHT
Object: Remap, 259

VX_REMAP_SOURCE_WIDTH
Object: Remap, 259

VX_ROUND_POLICY_TO_NEAREST_EVEN
Object: Context, 194

VX_ROUND_POLICY_TO_ZERO
Object: Context, 194

VX_SCALAR_OP_ADD
Object: Scalar, 267

VX_SCALAR_OP_AND
Object: Scalar, 266

VX_SCALAR_OP_DIVIDE
Object: Scalar, 267

VX_SCALAR_OP_EQUAL
Object: Scalar, 266

VX_SCALAR_OP_GREATERREQ
Object: Scalar, 267

VX_SCALAR_OP_GREATER
Object: Scalar, 266

VX_SCALAR_OP_LESSEQ
Object: Scalar, 266

VX_SCALAR_OP_LESS
Object: Scalar, 266

VX_SCALAR_OP_MAX
Object: Scalar, 267

VX_SCALAR_OP_MIN
Object: Scalar, 267

VX_SCALAR_OP_MODULUS
Object: Scalar, 267

VX_SCALAR_OP_MULTIPLY
Object: Scalar, 267

VX_SCALAR_OP_NAND
Object: Scalar, 266

VX_SCALAR_OP_NOTEQUAL
Object: Scalar, 266

VX_SCALAR_OP_OR
Object: Scalar, 266

VX_SCALAR_OP_SUBTRACT
Object: Scalar, 267

VX_SCALAR_OP_XOR
Object: Scalar, 266

VX_SCALAR_TYPE
Object: Scalar, 266

VX_STATUS_MIN
Basic Features, 180

VX_SUCCESS
Basic Features, 181

VX_TARGET_ANY
Basic Features, 182

VX_TARGET_STRING
Basic Features, 182

VX_TARGET_VENDOR_BEGIN
Basic Features, 183

VX_TENSOR_DATA_TYPE
Object: Tensor, 282

VX_TENSOR_DIMS
Object: Tensor, 282

VX_TENSOR_FIXED_POINT_POSITION
Object: Tensor, 282

VX_TENSOR_NUMBER_OF_DIMS
Object: Tensor, 282

VX_TERM_CRITERIA_BOTH
Object: Context, 194

VX_TERM_CRITERIA_EPSILON
Object: Context, 194

VX_TERM_CRITERIA_ITERATIONS
Object: Context, 194

VX_THRESHOLD_INPUT_FORMAT
Object: Threshold, 272

VX_THRESHOLD_OUTPUT_FORMAT
Object: Threshold, 272

VX_THRESHOLD_TYPE_BINARY
Object: Threshold, 272

VX_THRESHOLD_TYPE_RANGE
Object: Threshold, 272

VX_THRESHOLD_TYPE
Object: Threshold, 272

VX_TYPE_ARRAY
Basic Features, 179

VX_TYPE_BOOL
Basic Features, 178

VX_TYPE_CHAR
Basic Features, 178

VX_TYPE_CONTEXT
Basic Features, 179

VX_TYPE_CONVOLUTION
Basic Features, 179

VX_TYPE_COORDINATES2DF
Basic Features, 179

VX_TYPE_COORDINATES2D
Basic Features, 178

VX_TYPE_COORDINATES3D
Basic Features, 178

VX_TYPE_DELAY
Basic Features, 179
INDEX

VX_TYPE_DF_IMAGE
  Basic Features, 178
VX_TYPE_DISTRIBUTION
  Basic Features, 179
VX_TYPE_ENUM
  Basic Features, 178
VX_TYPE_ERROR
  Basic Features, 179
VX_TYPE_FLOAT32
  Basic Features, 178
VX_TYPE_FLOAT64
  Basic Features, 178
VX_TYPE_GRAPH
  Basic Features, 179
VX_TYPE_HOG_PARAMS
  Basic Features, 179
VX_TYPE_HOUGH_LINES_PARAMS
  Basic Features, 179
VX_TYPE_IMAGE
  Basic Features, 179
VX_TYPE_INT16
  Basic Features, 178
VX_TYPE_INT32
  Basic Features, 178
VX_TYPE_INT64
  Basic Features, 178
VX_TYPE_INT8
  Basic Features, 178
VX_TYPE_INVALID
  Basic Features, 178
VX_TYPE_KERNEL
  Basic Features, 179
VX_TYPE_KEYPOINT
  Basic Features, 178
VX_TYPE_KHRONOS_OBJECT_END
  Basic Features, 179
VX_TYPE_KHRONOS_OBJECT_START
  Basic Features, 179
VX_TYPE_KHRONOS_STRUCT_MAX
  Basic Features, 179
VX_TYPE_LINE_2D
  Basic Features, 179
VX_TYPE_LUT
  Basic Features, 179
VX_TYPE_MASK
  Basic Features, 177
VX_TYPE_MATRIX
  Basic Features, 179
VX_TYPE_META_FORMAT
  Basic Features, 179
VX_TYPE_NODE
  Basic Features, 179
VX_TYPE_OBJECT_ARRAY
  Basic Features, 179
VX_TYPE_PARAMETER
  Basic Features, 179
VX_TYPE_PYRAMID
  Basic Features, 179
VX_TYPE_RECTANGLE
  Basic Features, 178
VX_TYPE_REFERENCE
  Basic Features, 179
VX_TYPE_REMAP
  Basic Features, 179
VX_TYPE_SCALAR
  Basic Features, 179
VX_TYPE_SIZE
  Basic Features, 178
VX_TYPE_TENSOR_MATRIX_MULTIPLY_PARAMS
  Basic Features, 179
VX_TYPE_TENSOR
  Basic Features, 179
VX_TYPE_THRESHOLD
  Basic Features, 179
VX_TYPE_UINT16
  Basic Features, 178
VX_TYPE_UINT32
  Basic Features, 178
VX_TYPE_UINT64
  Basic Features, 178
VX_TYPE_UINT8
  Basic Features, 178
VX_TYPE_USER_STRUCT_END
  Basic Features, 179
VX_TYPE_USER_STRUCT_START
  Basic Features, 179
VX_TYPE_VENDOR_OBJECT_END
  Basic Features, 179
VX_TYPE_VENDOR_OBJECT_START
  Basic Features, 179
VX_TYPE_VENDOR_STRUCT_END
  Basic Features, 179
VX_TYPE_VENDOR_STRUCT_START
  Basic Features, 179
VX_ULBP
  LBP, 144
VX_VALID_RECT_CALLBACK
  Framework: User Kernels, 332
VX_WRITE_ONLY
  Object: Context, 194
Vision Functions, 31
vx_accessor_e
  Object: Context, 194
vx_action
  Framework: Node Callbacks, 317
vx_action_e
  Framework: Node Callbacks, 317
vx_array_attribute_e
  Object: Array, 211
vx_bool
  Basic Features, 177
vx_bool_e
  Basic Features, 178
vx_border_e
  Node: Border Modes, 290
vx_border_policy_e
Node: Border Modes, 290
vx_border_t, 290
vx_channel_e
Basic Features, 183
vx_channel_range_e
Object: Image, 232
vx_color_space_e
Object: Image, 231
vx_comp_metric_e
MatchTemplate, 141
vx_context
Object: Context, 192
vx_context_attribute_e
Object: Context, 192
vx_convert_policy_e
Basic Features, 181
vx_convolution_attribute_e
Object: Convolution, 218
vx_coordinates2d_t, 175
vx_coordinates2df_t, 175
vx_coordinates3d_t, 176
vx_delay
Object: Delay, 292
vx_delay_attribute_e
Object: Delay, 293
vx_df_image_e
Basic Features, 182
vx_direction_e
Object: Parameter, 310
vx_directive_e
Framework: Directives, 324
vx_distribution_attribute_e
Object: Distribution, 222
vx_enum
Basic Features, 177
vx_enum_e
Basic Features, 181
vx_false_e
Basic Features, 178
vx_graph
Object: Graph, 199
vx_graph_attribute_e
Object: Graph, 200
vx_graph_state_e
Object: Graph, 199
vx_hint_e
Framework: Hints, 322
vx_hog_t, 146
vx_hough_lines_p_t, 152
vx_image
Object: Image, 231
vx_image_attribute_e
Object: Image, 231
vx_imagepatch_addressing_t, 230
vx_interpolation_type_e
Basic Features, 183
vx_kernel
Object: Kernel, 300
vx_kernel_attribute_e
Object: Kernel, 305
vx_kernel_deinititalize_f
Framework: User Kernels, 331
vx_kernel_e
Object: Kernel, 300
vx_kernel_f
Framework: User Kernels, 330
vx_kernel_image_valid_rectangle_f
Framework: User Kernels, 331
vx_kernel_info_t, 298
vx_kernel_initialize_f
Framework: User Kernels, 331
vx_kernel_validate_f
Framework: User Kernels, 331
vx_keypoint_t, 176
vx_lbp_format_e
LBP, 144
vx_library_e
Object: Kernel, 300
vx_line2d_t, 176
vx_lut_attribute_e
Object: LUT, 244
vx_map_flag_e
Object: Image, 232
vx_matrix_attribute_e
Object: Matrix, 250
vx_memory_type_e
Object: Context, 193
vx_meta_format
Framework: User Kernels, 330
vx_meta_valid_rect_attribute_e
Framework: User Kernels, 332
vx_node
Object: Node, 206
vx_node_attribute_e
Object: Node, 206
vx_nodecomplete_f
Framework: Node Callbacks, 317
vx_non_linear_filter_e
Basic Features, 184
vx_norm_type_e
Canny Edge Detector, 65
vx_object_array_attribute_e
Object: ObjectArray, 278
vx_parameter
Object: Parameter, 310
vx_parameter_attribute_e
Object: Parameter, 310
vx_parameter_state_e
Object: Parameter, 310
vx_pattern_e
Basic Features, 184
vx_perf_t, 319
vx_pixel_value_t, 230
vx_publish_kernels_f
Framework: User Kernels, 330
vx_pyramid_attribute_e
Object: Threshold, 273
vxCreateArray
Object: Array, 211
vxCreateContext
Object: Context, 195
vxCreateConvolution
Object: Convolution, 219
vxCreateDelay
Object: Delay, 294
vxCreateDistribution
Object: Distribution, 223
vxCreateGenericNode
Object: Node (Advanced), 289
vxCreateGraph
Object: Graph, 200
vxCreateImage
Object: Image, 232
vxCreateImageFromChannel
Object: Image, 242
vxCreateImageFromHandle
Object: Image, 234
vxCreateImageFromROI
Object: Image, 232
vxCreateImageObjectArrayFromTensor
Object: Tensor, 283
vxCreateLUT
Object: LUT, 245
vxCreateMatrix
Object: Matrix, 251
vxCreateMatrixFromPattern
Object: Matrix, 253
vxCreateMatrixFromPatternAndOrigin
Object: Matrix, 253
vxCreateObjectArray
Object: ObjectArray, 279
vxCreatePyramid
Object: Pyramid, 256
vxCreateRemap
Object: Remap, 260
vxCreateScalar
Object: Scalar, 267
vxCreateScalarWithSize
Object: Scalar, 267
vxCreateTensor
Object: Tensor, 282
vxCreateTensorFromView
Object: Tensor, 283
vxCreateThresholdForImage
Object: Threshold, 272
vxCreateUniformImage
Object: Image, 233
vxCreateVirtualArray
Object: Array, 212
vxCreateVirtualConvolution
Object: Convolution, 219
vxCreateVirtualDistribution
Object: Distribution, 223
vxCreateVirtualImage
Object: Image, 233
vxCreateVirtualMatrix
Object: Matrix, 251
vxCreateVirtualObjectArray
Object: ObjectArray, 279
vxCreateVirtualPyramid
Object: Pyramid, 256
vxCreateVirtualRemap
Object: Remap, 260
vxCreateVirtualScalar
Object: Scalar, 267
vxCreateVirtualTensor
Object: Tensor, 283
vxCreateVirtualThresholdForImage
Object: Threshold, 273
vxDilate3x3Node
Dilate Image, 79
vxDirective
Framework: Directives, 324
vxEqualizeHistNode
Equalize Histogram, 81
vxErode3x3Node
Erode Image, 82
vxFastCornersNode
Fast Corners, 85
vxFinalizeKernel
Framework: User Kernels, 335
vxFormatArrayPointer
Object: Array, 211
vxFormatImagePatchAddress1d
Object: Image, 237
vxFormatImagePatchAddress2d
Object: Image, 238
vxGaussian3x3Node
Gaussian Filter, 87
vxGaussianPyramidNode
Gaussian Image Pyramid, 96
vxGetContext
Object: Context, 195
vxGetGraphParameterByIndex
Framework: Graph Parameters, 341
vxGetKernelByEnum
Object: Kernel, 307
vxGetKernelByName
Object: Kernel, 306
vxGetKernelParameterByIndex
Object: Parameter, 310
vxGetObjectArrayItem
Object: ObjectArray, 279
vxGetParameterByIndex
Object: Parameter, 311
vxGetPyramidLevel
Object: Pyramid, 258
vxGetReferenceFromDelay
Object: Delay, 294
vxGetStatus
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vxNonMaxSuppressionNode</td>
<td>Non-Maxima Suppression, 62</td>
</tr>
<tr>
<td>vxNotNode</td>
<td>Bitwise NOT, 58</td>
</tr>
<tr>
<td>vxOpticalFlowPyrLKNode</td>
<td>Optical Flow Pyramid (LK), 117</td>
</tr>
<tr>
<td>vxOrNode</td>
<td>Bitwise INCLUSIVE OR, 56</td>
</tr>
<tr>
<td>vxPhaseNode</td>
<td>Phase, 119</td>
</tr>
<tr>
<td>vxProcessGraph</td>
<td>Object: Graph, 201</td>
</tr>
<tr>
<td>vxQueryArray</td>
<td>Object: Array, 213</td>
</tr>
<tr>
<td>vxQueryContext</td>
<td>Object: Context, 196</td>
</tr>
<tr>
<td>vxQueryConvolution</td>
<td>Object: Convolution, 220</td>
</tr>
<tr>
<td>vxQueryDelay</td>
<td>Object: Delay, 293</td>
</tr>
<tr>
<td>vxQueryDistribution</td>
<td>Object: Distribution, 224</td>
</tr>
<tr>
<td>vxQueryGraph</td>
<td>Object: Graph, 203</td>
</tr>
<tr>
<td>vxQueryImage</td>
<td>Object: Image, 236</td>
</tr>
<tr>
<td>vxQueryKernel</td>
<td>Object: Kernel, 307</td>
</tr>
<tr>
<td>vxQueryLUT</td>
<td>Object: LUT, 246</td>
</tr>
<tr>
<td>vxQueryMatrix</td>
<td>Object: Matrix, 252</td>
</tr>
<tr>
<td>vxQueryNode</td>
<td>Object: Node, 206</td>
</tr>
<tr>
<td>vxQueryObjectArray</td>
<td>Object: ObjectArray, 280</td>
</tr>
<tr>
<td>vxQueryParameter</td>
<td>Object: Parameter, 312</td>
</tr>
<tr>
<td>vxQueryPyramid</td>
<td>Object: Pyramid, 257</td>
</tr>
<tr>
<td>vxQueryReference</td>
<td>Object: Reference, 188</td>
</tr>
<tr>
<td>vxQueryRemap</td>
<td>Object: Remap, 264</td>
</tr>
<tr>
<td>vxQueryScalar</td>
<td>Object: Scalar, 268</td>
</tr>
<tr>
<td>vxQueryTensor</td>
<td>Object: Tensor, 285</td>
</tr>
<tr>
<td>vxQueryThreshold</td>
<td>Object: Threshold, 277</td>
</tr>
<tr>
<td>vxRegisterAutoAging</td>
<td>Object: Graph, 204</td>
</tr>
<tr>
<td>vxRegisterLogCallback</td>
<td>Framework: Log, 321</td>
</tr>
<tr>
<td>vxRegisterUserStruct</td>
<td>Object: Array (Advanced), 288</td>
</tr>
<tr>
<td>vxReleaseArray</td>
<td>Object: Array, 212</td>
</tr>
</tbody>
</table>
vXReleaseContext
  Object: Context, 195
vXReleaseConvolution
  Object: Convolution, 219
vXReleaseDelay
  Object: Delay, 293
vXReleaseDistribution
  Object: Distribution, 223
vXReleaseGraph
  Object: Graph, 200
vXReleaseImage
  Object: Image, 237
vXReleaseKernel
  Object: Kernel, 308
vXReleaseLUT
  Object: LUT, 245
vXReleaseMatrix
  Object: Matrix, 251
vXReleaseNode
  Object: Node, 207
vXReleaseObjectArray
  Object: ObjectArray, 279
vXReleaseParameter
  Object: Parameter, 311
vXReleasePyramid
  Object: Pyramid, 257
vXReleaseReference
  Object: Reference, 188
vXReleaseRemap
  Object: Remap, 260
vXReleaseScalar
  Object: Scalar, 268
vXReleaseTensor
  Object: Tensor, 285
vXReleaseThreshold
  Object: Threshold, 276
vXRemapNode
  Remap, 123
vXRemoveKernel
  Framework: User Kernels, 336
vXRemoveNode
  Object: Node, 208
vXReplicateNode
  Object: Node, 208
vXRetainReference
  Object: Reference, 189
vXReteiveNodeCallback
  Framework: Node Callbacks, 318
vXScalarOperationNode
  Control Flow, 46
vXScaleImageNode
  Scale Image, 127
vXScheduleGraph
  Object: Graph, 202
vXSelectNode
  Control Flow, 46
vXSetContextAttribute
  Object: Context, 196
vXSetConvolutionAttribute
  Object: Convolution, 220
vXSetGraphAttribute
  Object: Graph, 203
vXSetGraphParameterByIndex
  Framework: Graph Parameters, 340
vXSetImageAttribute
  Object: Image, 236
vXSetImagePixelValues
  Object: Image, 236
vXSetImageValidateRectangle
  Object: Image, 242
vXSetImmediateModeTarget
  Object: Context, 196
vXSetKernelAttribute
  Framework: User Kernels, 336
vXSetMetaFormatAttribute
  Framework: User Kernels, 337
vXSetMetaFormatFromReference
  Framework: User Kernels, 338
vXSetNodeAttribute
  Object: Node, 207
vXSetNodeTarget
  Object: Node, 208
vXSetParameterByIndex
  Object: Parameter, 311
vXSetParameterByReference
  Object: Parameter, 312
vXSetReferenceName
  Object: Reference, 189
vXSetThresholdAttribute
  Object: Threshold, 276
vXSetThreshold
  Arithmetic Subtraction, 50
vXSwapImageHandle
  Object: Image, 235
vXTableLookupNode
  TableLookup, 131
vXTensorAddNode
  Tensor Add, 157
vXTensorConvertDepthNode
  Tensor Convert Bit-Depth, 165
vXTensorMatrixMultiplyNode
  Tensor Multiply, 168
vXTensorMultiplyNode
  Tensor Multiply, 155
vXTensorSubtractNode
  Tensor Subtract, 159
vXTensorTableLookupNode
  Tensor TableLookUp, 161
vXTensorTransposeNode
  Tensor Transpose, 163
vXTensorThresholdNode
  Thresholding, 133
vXTruncateArray
  Object: Array, 214
vxUnloadKernels  
Framework: User Kernels, 334
vxUnmapArrayRange  
Object: Array, 216
vxUnmapDistribution  
Object: Distribution, 226
vxUnmapImagePatch  
Object: Image, 241
vxUnmapLUT  
Object: LUT, 248
vxUnmapRemapPatch  
Object: Remap, 262
vxVerifyGraph  
Object: Graph, 201
vxWaitGraph  
Object: Graph, 202
vxWarpAffineNode  
Warp Affine, 135
vxWarpPerspectiveNode  
Warp Perspective, 137
vxXorNode  
Bitwise EXCLUSIVE OR, 54
vxuAbsDiff  
Absolute Difference, 35
vxuAccumulateImage  
Accumulate, 37
vxuAccumulateSquareImage  
Accumulate Squared, 39
vxuAccumulateWeightedImage  
Accumulate Weighted, 43
vxuAdd  
Arithmetic Addition, 48
vxuAnd  
Bitwise AND, 52
vxuBilateralFilter  
Bilateral Filter, 140
vxuBox3x3  
Box Filter, 60
vxuCannyEdgeDetector  
Canny Edge Detector, 66
vxuChannelCombine  
Channel Combine, 67
vxuChannelExtract  
Channel Extract, 69
vxuColorConvert  
Color Convert, 73
vxuConvertDepth  
Convert Bit depth, 76
vxuConvolve  
Custom Convolution, 78
vxuCopy  
Data Object Copy, 41
vxuDilate3x3  
Dilate Image, 79
vxuEqualizeHist  
Equalize Histogram, 81
vxuErode3x3  
Erode Image, 82
vxuFastCorners  
Fast Corners, 85
vxuGaussian3x3  
Gaussian Filter, 87
vxuGaussianPyramid  
Gaussian Image Pyramid, 96
vxuHOGCells  
HOG, 149
vxuHOGFeatures  
HOG, 150
vxuHalfScaleGaussian  
Scale Image, 128
vxuHarrisCorners  
Harris Corners, 93
vxuHistogram  
Histogram, 94
vxuHoughLinesP  
HoughLinesP, 153
vxuIntegralImage  
Integral Image, 102
vxuLBP  
LBP, 145
vxuLaplacianPyramid  
Laplacian Image Pyramid, 99
vxuLaplacianReconstruct  
Reconstruction from a Laplacian Image Pyramid, 100
vxuMagnitude  
Magnitude, 104
vxuMatchTemplate  
MatchTemplate, 142
vxuMax  
Max, 114
vxuMeanStdDev  
Mean and Standard Deviation, 106
vxuMedian3x3  
Median Filter, 108
vxuMin  
Min, 112
vxuMinMaxLoc  
Min, Max Location, 110
vxuMultiply  
Pixel-wise Multiplication, 122
vxuNonLinearFilter  
Non Linear Filter, 89
vxuNonMaxSuppression  
Non-Maxima Suppression, 62
vxuNot  
Bitwise NOT, 58
vxuOpticalFlowPyrLK  
Optical Flow Pyramid (LK), 117
vxuOr  
Bitwise INCLUSIVE OR, 56
vxuPhase  
Phase, 119
vxuRemap  
Remap, 123
vxuScaleImage
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale Image</td>
<td>128</td>
</tr>
<tr>
<td>vxuSobel3x3</td>
<td>129</td>
</tr>
<tr>
<td>Sobel 3x3</td>
<td></td>
</tr>
<tr>
<td>vxuSubtract</td>
<td></td>
</tr>
<tr>
<td>Arithmetic Subtraction</td>
<td>50</td>
</tr>
<tr>
<td>vxuTableLookup</td>
<td>131</td>
</tr>
<tr>
<td>TableLookup</td>
<td></td>
</tr>
<tr>
<td>vxuTensorAdd</td>
<td>157</td>
</tr>
<tr>
<td>Tensor Add</td>
<td></td>
</tr>
<tr>
<td>vxuTensorConvertDepth</td>
<td>165</td>
</tr>
<tr>
<td>Tensor Convert Bit-Depth</td>
<td></td>
</tr>
<tr>
<td>vxuTensorMatrixMultiply</td>
<td>168</td>
</tr>
<tr>
<td>Tensor Matrix Multiply</td>
<td></td>
</tr>
<tr>
<td>vxuTensorMultiply</td>
<td>155</td>
</tr>
<tr>
<td>Tensor Multiply</td>
<td></td>
</tr>
<tr>
<td>vxuTensorSubtract</td>
<td>159</td>
</tr>
<tr>
<td>Tensor Subtract</td>
<td></td>
</tr>
<tr>
<td>vxuTensorTableLookup</td>
<td>161</td>
</tr>
<tr>
<td>Tensor Table LookUp</td>
<td></td>
</tr>
<tr>
<td>vxuTensorTranspose</td>
<td>163</td>
</tr>
<tr>
<td>Tensor Transpose</td>
<td></td>
</tr>
<tr>
<td>vxuThreshold</td>
<td>134</td>
</tr>
<tr>
<td>Thresholding</td>
<td></td>
</tr>
<tr>
<td>vxuWarpAffine</td>
<td>136</td>
</tr>
<tr>
<td>Warp Affine</td>
<td></td>
</tr>
<tr>
<td>vxuWarpPerspective</td>
<td>138</td>
</tr>
<tr>
<td>Warp Perspective</td>
<td></td>
</tr>
<tr>
<td>vxuXor</td>
<td>54</td>
</tr>
<tr>
<td>Bitwise EXCLUSIVE OR</td>
<td></td>
</tr>
<tr>
<td>Warp Affine</td>
<td>135</td>
</tr>
<tr>
<td>vxWarpAffineNode</td>
<td>135</td>
</tr>
<tr>
<td>vxuWarpAffine</td>
<td>136</td>
</tr>
<tr>
<td>Warp Perspective</td>
<td>137</td>
</tr>
<tr>
<td>vxWarpPerspectiveNode</td>
<td>137</td>
</tr>
<tr>
<td>vxuWarpPerspective</td>
<td>138</td>
</tr>
</tbody>
</table>