The OpenVX™ [Provisional] Specification

Version 1.0

Document Revision: r26495
Generated on Fri May 2 2014 16:13:28

Khronos Vision Working Group

Editor: Susheel Gautam
Editor: Erik Rainey

Copyright ©2014 The Khronos Group Inc.
# Contents

1 Introduction
   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 Requirements Language .............................. 3
   1.7 Typographical Conventions .......................... 3
      1.7.1 Naming Conventions ........................... 3
   1.8 Glossary and Acronyms ............................. 4
   1.9 Acknowledgements .................................. 4

2 Design Overview ....................................... 6
   2.1 Software Landscape .................................. 6
   2.2 Design Objectives .................................. 7
      2.2.1 Hardware Optimizations ........................ 7
      2.2.2 Hardware Limitations ........................... 7
   2.3 Assumptions ......................................... 7
      2.3.1 Portability ..................................... 7
      2.3.2 Opaqueness ..................................... 7
   2.4 Object Oriented Behaviors ........................... 7
   2.5 OpenVX Framework Objects ........................... 8
   2.6 OpenVX Data Objects ................................ 8
   2.7 Error Objects ....................................... 9
   2.8 Graphs Concepts .................................... 9
      2.8.1 Linking Nodes .................................... 9
      2.8.2 Virtual Data Objects ............................ 9
      2.8.3 Node Parameters ................................ 10
      2.8.4 Graph Parameters ................................ 10
      2.8.5 Execution Model ................................ 10
         Asynchronous Mode .................................. 10
      2.8.6 Graph Formalisms ................................ 10
      2.8.7 Node Execution Independence .................... 11
      2.8.8 Verification ...................................... 12
   2.9 Callbacks ........................................... 13
   2.10 Client Defined Functions (CDF) ....................... 13
      2.10.1 Parameter Validation ............................ 14
         The Meta Format Object .............................. 14
      2.10.2 Client Defined Function Naming Conventions .... 15
   2.11 Immediate Mode Functions ............................ 15
   2.12 Base Vision Functions ................................ 15
      2.12.1 Inputs ........................................... 15
      2.12.2 Outputs ......................................... 17
   2.13 Lifecycles ........................................... 18
      2.13.1 OpenVX Context Lifecycle ....................... 18
      2.13.2 Graph Lifecycle ................................ 18
      2.13.3 Data Object Lifecycle ........................... 19
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.14</td>
<td>Host Memory Data Object Access Patterns</td>
<td>19</td>
</tr>
<tr>
<td>2.14.1</td>
<td>Matrix Access Example</td>
<td>20</td>
</tr>
<tr>
<td>2.14.2</td>
<td>Image Access Example</td>
<td>20</td>
</tr>
<tr>
<td>2.14.3</td>
<td>Array Access Example</td>
<td>20</td>
</tr>
<tr>
<td>2.15</td>
<td>Extending OpenVX</td>
<td>21</td>
</tr>
<tr>
<td>2.15.1</td>
<td>Extending Attributes</td>
<td>21</td>
</tr>
<tr>
<td>2.15.2</td>
<td>Vendor Custom Kernels</td>
<td>22</td>
</tr>
<tr>
<td>2.15.3</td>
<td>Vendor Custom Extensions</td>
<td>22</td>
</tr>
<tr>
<td>2.15.4</td>
<td>Hinting</td>
<td>22</td>
</tr>
<tr>
<td>2.15.5</td>
<td>Directives</td>
<td>22</td>
</tr>
<tr>
<td>2.16</td>
<td>Known Extensions to OpenVX</td>
<td>23</td>
</tr>
<tr>
<td>2.16.1</td>
<td>User Node Tiling</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>Module Documentation</td>
<td>24</td>
</tr>
<tr>
<td>3.1</td>
<td>Vision Functions</td>
<td>24</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Detailed Description</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Function: Absolute Difference</td>
<td>27</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Detailed Description</td>
<td>27</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Function Documentation</td>
<td>27</td>
</tr>
<tr>
<td>3.3</td>
<td>Function: Accumulate</td>
<td>28</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Detailed Description</td>
<td>28</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Function Documentation</td>
<td>28</td>
</tr>
<tr>
<td>3.4</td>
<td>Function: Accumulate Squared</td>
<td>29</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Detailed Description</td>
<td>29</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Function Documentation</td>
<td>29</td>
</tr>
<tr>
<td>3.5</td>
<td>Function: Accumulate Weighted</td>
<td>31</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Detailed Description</td>
<td>31</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Function Documentation</td>
<td>31</td>
</tr>
<tr>
<td>3.6</td>
<td>Function: Arithmetic Addition</td>
<td>33</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Detailed Description</td>
<td>33</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Function Documentation</td>
<td>33</td>
</tr>
<tr>
<td>3.7</td>
<td>Function: Arithmetic Subtraction</td>
<td>35</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Detailed Description</td>
<td>35</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Function Documentation</td>
<td>35</td>
</tr>
<tr>
<td>3.8</td>
<td>Function: Bitwise And</td>
<td>37</td>
</tr>
<tr>
<td>3.8.1</td>
<td>Detailed Description</td>
<td>37</td>
</tr>
<tr>
<td>3.8.2</td>
<td>Function Documentation</td>
<td>37</td>
</tr>
<tr>
<td>3.9</td>
<td>Function: Bitwise Exclusive Or</td>
<td>39</td>
</tr>
<tr>
<td>3.9.1</td>
<td>Detailed Description</td>
<td>39</td>
</tr>
<tr>
<td>3.9.2</td>
<td>Function Documentation</td>
<td>39</td>
</tr>
<tr>
<td>3.10</td>
<td>Function: Bitwise Inclusive Or</td>
<td>41</td>
</tr>
</tbody>
</table>
3.10.1 Detailed Description ................................................................. 41
3.10.2 Function Documentation ............................................................ 41
vxOrNode .................................................................................. 41
vxuOr .................................................................................... 41
3.11 Function: Bitwise Not ................................................................. 43
3.11.1 Detailed Description ................................................................. 43
3.11.2 Function Documentation ............................................................ 43
vxuNotNode ................................................................................ 43
vxuNot ................................................................................... 43
3.12 Function: Box Filter ................................................................... 44
3.12.1 Detailed Description ................................................................. 44
3.12.2 Function Documentation ............................................................ 44
vxBox3x3Node ........................................................................... 44
vxuBox3x3 ............................................................................ 44
3.13 Function: Canny Edge Detector ..................................................... 45
3.13.1 Detailed Description ................................................................. 45
3.13.2 Enumeration Type Documentation ............................................. 46
vx_norm_type_e ....................................................................... 46
3.13.3 Function Documentation ............................................................ 46
vxuCannyEdgeDetectorNode ........................................................... 46
vxuCannyEdgeDetector ................................................................. 46
3.14 Function: Channel Combine .......................................................... 48
3.14.1 Detailed Description ................................................................. 48
3.14.2 Function Documentation ............................................................ 48
vxChannelCombineNode ................................................................. 48
vxuChannelCombine ................................................................. 48
3.15 Function: Channel Extract ............................................................. 50
3.15.1 Detailed Description ................................................................. 50
3.15.2 Function Documentation ............................................................ 50
vxChannelExtractNode ................................................................. 50
vxuChannelExtract ................................................................. 50
3.16 Function: Color Convert ............................................................... 52
3.16.1 Detailed Description ................................................................. 52
3.16.2 Function Documentation ............................................................ 52
vxColorConvertNode ................................................................. 54
vxuColorConvert ................................................................. 54
3.17 Function: Convert Bit depth ............................................................ 56
3.17.1 Detailed Description ................................................................. 56
3.17.2 Function Documentation ............................................................ 56
vxConvertDepthNode ................................................................. 56
vxuConvertDepth ................................................................. 57
3.18 Function: Custom Convolution ....................................................... 58
3.18.1 Detailed Description ................................................................. 58
3.18.2 Function Documentation ............................................................ 58
vxConvolveNode ................................................................. 58
vxuConvolve ................................................................. 59
3.19 Function: Dilate Image ................................................................. 60
3.19.1 Detailed Description ................................................................. 60
3.19.2 Function Documentation ............................................................ 60
vxDilate3x3Node ................................................................. 60
vxuDilate3x3 ........................................................................ 60
3.20 Function: Equalize Histogram ......................................................... 61
3.20.1 Detailed Description ................................................................. 61
3.20.2 Function Documentation ............................................................ 61
vxEqualizeHistNode ................................................................. 61
vxuEqualizeHist ................................................................. 61
3.21 Function: Erode Image ................................................................. 62
3.21.1 Detailed Description ................................................................. 62
CONTENTS

3.21.2 Function Documentation .......................................................... 62
vxErode3x3Node ................................................................. 62
vxuErode3x3 ................................................................. 62
3.22 Function: Fast Corners ............................................................. 63
3.22.1 Detailed Description .......................................................... 63
3.22.2 Segment Test Detector ....................................................... 63
3.22.3 Function Documentation ....................................................... 64
vxFastCornersNode .............................................................. 64
vxuFastCorners ............................................................... 64
3.23 Function: Gaussian Filter ......................................................... 66
3.23.1 Detailed Description .......................................................... 66
3.23.2 Function Documentation ....................................................... 66
vxGaussian3x3Node .............................................................. 66
vxuGaussian3x3 ............................................................... 66
3.24 Function: Harris Corners ......................................................... 67
3.24.1 Detailed Description .......................................................... 67
3.24.2 Function Documentation ....................................................... 68
vxHarrisCornersNode ............................................................. 68
vxuHarrisCorners .............................................................. 68
3.25 Function: Histogram ............................................................... 70
3.25.1 Detailed Description .......................................................... 70
3.25.2 Function Documentation ....................................................... 70
vxHistogramNode ............................................................... 70
vxuHistogram ............................................................... 70
3.26 Function: Gaussian Image Pyramid ............................................. 71
3.26.1 Detailed Description .......................................................... 71
3.26.2 Function Documentation ....................................................... 71
vxGaussianPyramidNode ......................................................... 71
vxuGaussianPyramid ......................................................... 71
3.27 Function: Integral Image .......................................................... 73
3.27.1 Detailed Description .......................................................... 73
3.27.2 Function Documentation ....................................................... 73
vxIntegralImageNode ............................................................ 73
vxuIntegralImage ............................................................ 73
3.28 Function: Magnitude ............................................................... 74
3.28.1 Detailed Description .......................................................... 74
3.28.2 Function Documentation ....................................................... 74
vxMagnitudeNode ............................................................... 74
vxuMagnitude ............................................................... 74
3.29 Function: Mean and Standard Deviation ....................................... 76
3.29.1 Detailed Description .......................................................... 76
3.29.2 Function Documentation ....................................................... 76
vxMeanStdDevNode .............................................................. 76
vxuMeanStdDev .............................................................. 76
3.30 Function: Median Filter ........................................................... 78
3.30.1 Detailed Description .......................................................... 78
3.30.2 Function Documentation ....................................................... 78
vxMedian3x3Node ............................................................... 78
vxuMedian3x3 ............................................................... 78
3.31 Function: Min, Max Location ..................................................... 79
3.31.1 Detailed Description .......................................................... 79
3.31.2 Function Documentation ....................................................... 79
vxMinMaxLocNode .............................................................. 79
vxuMinMaxLoc .............................................................. 79
3.32 Function: Optical Flow Pyramid (LK) ....................................... 81
3.32.1 Detailed Description .......................................................... 81
3.32.2 Function Documentation ....................................................... 82
vxOpticalFlowPyrLKNode ......................................................... 82
Contents

3.33 Function: Phase .................................................. 85
  3.33.1 Detailed Description ................................... 85
  3.33.2 Function Documentation
    vxPhaseNode .................................................. 85
    vxPhase ...................................................... 85

3.34 Function: Pixel-wise Multiplication .................. 87
  3.34.1 Detailed Description ................................ 87
  3.34.2 Function Documentation
    vxMultiplyNode ............................................... 87
    vxuMultiply .................................................. 88

3.35 Function: Remap ................................................. 89
  3.35.1 Detailed Description ................................ 89
  3.35.2 Function Documentation
    vxRemapNode ................................................ 89
    vxuRemap .................................................... 89

3.36 Function: Scale Image ........................................ 91
  3.36.1 Detailed Description ................................ 91
  3.36.2 Function Documentation
    vxScaleImageNode ........................................... 91
    vxuHalfScaleGaussian3x3 ................................... 91
    vxuScaleImage ................................................. 92

3.37 Function: Sobel 3x3 ............................................. 93
  3.37.1 Detailed Description ................................ 93
  3.37.2 Function Documentation
    vxSobel3x3Node ............................................... 93
    vxuSobel3x3 .................................................. 93

3.38 Function: TableLookup .......................................... 95
  3.38.1 Detailed Description ................................ 95
  3.38.2 Function Documentation
    vxTableLookupNode .......................................... 95
    vxuTableLookup ................................................ 95

3.39 Function: Thresholding ....................................... 96
  3.39.1 Detailed Description ................................ 96
  3.39.2 Function Documentation
    vxThresholdNode .............................................. 96
    vxuThreshold ................................................ 96

3.40 Function: Warp Affine ....................................... 98
  3.40.1 Detailed Description ................................ 98
  3.40.2 Function Documentation
    vxuWarpAffine ............................................... 98
    vxuWarpAffineNode .......................................... 98

3.41 Function: Warp Perspective ............................... 100
  3.41.1 Detailed Description ................................ 100
  3.41.2 Function Documentation
    vxuWarpPerspective ......................................... 100
    vxuWarpPerspectiveNode ..................................... 101

3.42 Basic Features .................................................. 102
  3.42.1 Detailed Description ................................ 102
  3.42.2 Data Structure Documentation
    struct vx_coordinates2d_t ................................ 107
    struct vx_coordinates3d_t ................................ 107
    struct vx_keypoint_t ...................................... 107
    struct vx_rectangle_t ..................................... 108

3.42.3 Macro Definition Documentation
    VX_ENUM_BASE .................................................. 108
    VX_FMT_REF ................................................... 108
    VX_FMT_SIZE ................................................... 108
CONTENTS

3.42.4 Typedef Documentation
vx_enum ....................................................... 109
vx_status ...................................................... 109
3.42.5 Enumeration Type Documentation
vx_bool ......................................................... 109
vx_channel_e .................................................. 109
vx_convert_policy_e .......................................... 110
vx_enum_e ....................................................... 110
vx_fourcc_e .................................................... 111
vx_interpolation_type_e ..................................... 111
vx_status_e ..................................................... 112
vx_type_e ....................................................... 113
vx_vendor_id_e ................................................ 114
3.42.6 Function Documentation
vxGetStatus ................................................... 115

3.43 Objects ..................................................... 116
3.43.1 Detailed Description ...................................... 116
3.44 Object: Reference ........................................... 117
3.44.1 Detailed Description ...................................... 117
3.44.2 Typedef Documentation ................................... 117
3.44.3 Enumeration Type Documentation
vx_reference ..................................................... 117
vx_reference_attribute_e ...................................... 117
3.44.4 Function Documentation
vxQueryReference ............................................... 117
3.45 Object: Context .............................................. 119
3.45.1 Detailed Description ...................................... 119
3.45.2 Typedef Documentation ................................... 120
3.45.3 Enumeration Type Documentation
vx_accessor_e .................................................... 120
vx_context_attribute_e ........................................ 121
vx_import_type_e ............................................... 121
vx_round_policy_e .............................................. 122
vx_termination_criteria_e ..................................... 122
3.45.4 Function Documentation
vxCreateContext ............................................... 122
vxGetContext ................................................... 122
vxQueryContext ................................................ 123
vxReleaseContext .............................................. 123
vxSetContextAttribute ........................................ 123
3.46 Object: Graph ................................................. 125
3.46.1 Detailed Description ...................................... 125
3.46.2 Typedef Documentation ................................... 126
3.46.3 Enumeration Type Documentation
vx_graph .......................................................... 126
3.46.4 Function Documentation
vxCreateGraph .................................................. 126
vxIsGraphVerified .............................................. 126
vxProcessGraph ................................................ 127
vxQueryGraph ................................................... 127

VX_FOURCC .......................................................... 108
VX_SCALE UNITY .................................................. 108
VX_TYPE MASK ..................................................... 108
VX_VERSION ........................................................ 109
VX_VERSION_MAJOR ............................................... 109
VX_VERSION_MINOR ............................................... 109
CONTENTS

3.51.4 Enumeration Type Documentation
vx_channel_range_e .......................... 149
vx_color_space_e ................................ 149
vx_image_attribute_e ............................ 150

3.51.5 Function Documentation
vxAccessImagePatch ................................ 150
vxCommitImagePatch ................................ 152
vxComputeImagePatchSize .......................... 154
vxCreateImage .................................... 154
vxCreateImageFromHandle ......................... 155
vxCreateImageFromROI ............................. 156
vxCreateUniformImage ............................. 156
vxCreateVirtualImage ............................ 157
vxFormatImagePatchAddress1d .................... 157
vxFormatImagePatchAddress2d .................... 159
vxGetValidRegionImage ........................... 160
vxQueryImage ..................................... 161
vxReleaseImage ................................... 161
vxSetImageAttribute ................................ 162

3.52 Object: LUT ........................................ 163
3.52.1 Detailed Description ......................... 163
3.52.2 Enumeration Type Documentation
vx_lut_attribute_e ................................ 163

3.52.3 Function Documentation
vxAccessLUT ...................................... 163
vxCommitLUT ...................................... 164
vxCreateLUT ....................................... 164
vxQueryLUT ........................................ 165
vxReleaseLUT ...................................... 166

3.53 Object: Matrix ..................................... 167
3.53.1 Detailed Description ......................... 167
3.53.2 Enumeration Type Documentation
vx_matrix_attribute_e ............................. 167

3.53.3 Function Documentation
vxAccessMatrix ................................... 167
vxCommitMatrix ................................... 168
vxCreateMatrix ................................... 168
vxQueryMatrix .................................... 168
vxReleaseMatrix .................................. 169

3.54 Object: Pyramid .................................. 170
3.54.1 Detailed Description .......................... 170
3.54.2 Enumeration Type Documentation
vx_pyramid_attribute_e ............................ 171

3.54.3 Function Documentation
vxCreatePyramid .................................. 171
vxCreateVirtualPyramid ........................... 171
vxGetPyramidLevel ................................ 172
vxQueryPyramid ................................... 172
vxReleasePyramid .................................. 172

3.55 Object: Remap ..................................... 174
3.55.1 Detailed Description ......................... 174
3.55.2 Enumeration Type Documentation
vx_remap_attribute_e ................................ 174

3.55.3 Function Documentation
vxCreateRemap .................................... 175
vxGetRemapPoint ................................... 176
vxQueryRemap ..................................... 176
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.56</td>
<td>Object: Scalar</td>
<td>178</td>
</tr>
<tr>
<td>3.56.1</td>
<td>Detailed Description</td>
<td>178</td>
</tr>
<tr>
<td>3.56.2</td>
<td>Typedef Documentation</td>
<td>178</td>
</tr>
<tr>
<td>3.56.3</td>
<td>Enumeration Type Documentation</td>
<td>178</td>
</tr>
<tr>
<td>3.56.4</td>
<td>Function Documentation</td>
<td>179</td>
</tr>
<tr>
<td></td>
<td>vx_scalar</td>
<td>178</td>
</tr>
<tr>
<td></td>
<td>vx_scalar_attribute_e</td>
<td>178</td>
</tr>
<tr>
<td>3.57</td>
<td>Object: Threshold</td>
<td>181</td>
</tr>
<tr>
<td>3.57.1</td>
<td>Detailed Description</td>
<td>181</td>
</tr>
<tr>
<td>3.57.2</td>
<td>Enumeration Type Documentation</td>
<td>181</td>
</tr>
<tr>
<td>3.57.3</td>
<td>Function Documentation</td>
<td>182</td>
</tr>
<tr>
<td></td>
<td>vxCreateThreshold</td>
<td>182</td>
</tr>
<tr>
<td></td>
<td>vxQueryThreshold</td>
<td>182</td>
</tr>
<tr>
<td></td>
<td>vxReleaseThreshold</td>
<td>182</td>
</tr>
<tr>
<td>3.58</td>
<td>Basic Framework</td>
<td>184</td>
</tr>
<tr>
<td>3.58.1</td>
<td>Detailed Description</td>
<td>184</td>
</tr>
<tr>
<td>3.59</td>
<td>Framework: Node Callbacks</td>
<td>185</td>
</tr>
<tr>
<td>3.59.1</td>
<td>Detailed Description</td>
<td>185</td>
</tr>
<tr>
<td>3.59.2</td>
<td>Typedef Documentation</td>
<td>187</td>
</tr>
<tr>
<td>3.59.3</td>
<td>Enumeration Type Documentation</td>
<td>187</td>
</tr>
<tr>
<td>3.59.4</td>
<td>Function Documentation</td>
<td>187</td>
</tr>
<tr>
<td></td>
<td>vxAssignNodeCallback</td>
<td>187</td>
</tr>
<tr>
<td></td>
<td>vxRetrieveNodeCallback</td>
<td>188</td>
</tr>
<tr>
<td>3.60</td>
<td>Framework: Performance Measurement</td>
<td>189</td>
</tr>
<tr>
<td>3.60.1</td>
<td>Detailed Description</td>
<td>189</td>
</tr>
<tr>
<td>3.60.2</td>
<td>Data Structure Documentation</td>
<td>189</td>
</tr>
<tr>
<td></td>
<td>struct vx_perf_t</td>
<td>189</td>
</tr>
<tr>
<td>3.61</td>
<td>Advanced Features</td>
<td>190</td>
</tr>
<tr>
<td>3.61.1</td>
<td>Detailed Description</td>
<td>190</td>
</tr>
<tr>
<td>3.62</td>
<td>Advanced Objects</td>
<td>191</td>
</tr>
<tr>
<td>3.62.1</td>
<td>Detailed Description</td>
<td>191</td>
</tr>
<tr>
<td>3.63</td>
<td>Object: Array (Advanced)</td>
<td>192</td>
</tr>
<tr>
<td>3.63.1</td>
<td>Detailed Description</td>
<td>192</td>
</tr>
<tr>
<td>3.63.2</td>
<td>Function Documentation</td>
<td>192</td>
</tr>
<tr>
<td></td>
<td>vxRegisterUserStruct</td>
<td>192</td>
</tr>
<tr>
<td>3.64</td>
<td>Object: Node (Advanced)</td>
<td>193</td>
</tr>
<tr>
<td>3.64.1</td>
<td>Detailed Description</td>
<td>193</td>
</tr>
<tr>
<td>3.64.2</td>
<td>Function Documentation</td>
<td>193</td>
</tr>
<tr>
<td></td>
<td>vxCreateNode</td>
<td>193</td>
</tr>
<tr>
<td>3.65</td>
<td>Node: Border Modes</td>
<td>194</td>
</tr>
<tr>
<td>3.65.1</td>
<td>Detailed Description</td>
<td>194</td>
</tr>
<tr>
<td>3.65.2</td>
<td>Data Structure Documentation</td>
<td>194</td>
</tr>
<tr>
<td></td>
<td>struct vx_border_mode_t</td>
<td>194</td>
</tr>
<tr>
<td>3.65.3</td>
<td>Enumeration Type Documentation</td>
<td>194</td>
</tr>
<tr>
<td></td>
<td>vx_border_mode_e</td>
<td>194</td>
</tr>
</tbody>
</table>
3.66 Object: Delay .......................................................... 195
  3.66.1 Detailed Description ........................................... 195
  3.66.2 Typedef Documentation ........................................ 195
  vx_delay .......................................................... 195
  3.66.3 Enumeration Type Documentation ............................. 196
  vx_delay_attribute_e ............................................. 196
  3.66.4 Function Documentation ...................................... 196
  vxAgeDelay ...................................................... 196
  vxAssociateDelayWithNode ....................................... 196
  vxCreateDelay .................................................. 197
  vxDissociateDelayFromNode ..................................... 197
  vxGetReferenceFromDelay ........................................ 198
  vxQueryDelay ................................................... 198
  vxReleaseDelay ................................................ 198

3.67 Object: Kernel ................................................... 199
  3.67.1 Detailed Description ........................................ 199
  3.67.2 Data Structure Documentation ............................... 201
  struct vx_kernel_info_t ......................................... 201
  3.67.3 Typedef Documentation ...................................... 201
  vx_kernel ........................................................ 201
  3.67.4 Enumeration Type Documentation ........................... 201
  vx_kernel_attribute_e .......................................... 201
  vx_kernel_e ...................................................... 202
  3.67.5 Function Documentation .................................... 210
  vxGetKernelByEnum ............................................... 210
  vxGetKernelByName ............................................... 211
  vxQueryKernel .................................................. 211
  vxReleaseKernel ................................................. 212

3.68 Object: Parameter ............................................... 213
  3.68.1 Detailed Description ........................................ 213
  3.68.2 Typedef Documentation ...................................... 214
  vx_parameter .................................................... 214
  3.68.3 Enumeration Type Documentation ........................... 214
  vx_direction_e .................................................. 214
  vx_parameter_attribute_e ....................................... 214
  vx_parameter_state_e .......................................... 214
  3.68.4 Function Documentation .................................... 214
  vxGetParameterByIndex .......................................... 215
  vxQueryParameter ................................................ 215
  vxReleaseParameter ............................................. 215
  vxSetParameterByIndex ......................................... 215
  vxSetParameterByReference ..................................... 215

3.69 Advanced Framework API ........................................ 217
  3.69.1 Detailed Description ....................................... 217

3.70 Framework: Log ................................................ 218
  3.70.1 Detailed Description ....................................... 218
  3.70.2 Function Documentation .................................... 218
  vxAddLogEntry ................................................... 218
  vxRegisterLogCallback .......................................... 218

3.71 Framework: Hints ............................................... 219
  3.71.1 Detailed Description ....................................... 219
  3.71.2 Enumeration Type Documentation ......................... 219
  vx_hint_e ......................................................... 219
  3.71.3 Function Documentation .................................... 219
  vxHint .......................................................... 219

3.72 Framework: Directives ........................................ 221
  3.72.1 Detailed Description ....................................... 221
  3.72.2 Enumeration Type Documentation ........................... 221
CONTENTS

vx_directive_e ............................................. 221
3.72.3 Function Documentation .................................. 221
vxDirective .................................................. 221
3.73 Framework: Client Defined Functions ......................... 223
3.73.1 Detailed Description .................................... 223
3.73.2 Typedef Documentation ................................ 225
vx_kernel_deinit_f ........................................... 225
vx_kernel_f ................................................ 225
vx_kernel_init_f ........................................... 226
vx_kernel_input_validate_f ................................ 226
vx_kernel_output_validate_f ................................ 226
vx_publish_kernels_f ....................................... 227
3.73.3 Function Documentation ................................ 227
vxAddKernel ................................................. 227
vxAddParameterToKernel ..................................... 228
vxFinalizeKernel ........................................... 228
vxLoadKernels .............................................. 228
vxRemoveKernel ............................................ 229
vxSetKernelAttribute ........................................ 229
3.74 Framework: Graph Parameters ............................... 231
3.74.1 Detailed Description ................................... 231
3.74.2 Function Documentation ................................ 232
vxAddParameterToGraph ....................................... 232
vxGetGraphParameterByIndex ................................ 232
vxSetGraphParameterByIndex ................................ 233
Chapter 1

Introduction

1.1 Abstract

OpenVX is a low level programming framework for the Computer Vision domain. OpenVX has been designed for supporting modern hardware systems such as mobile and embedded SoCs, and desktop systems. These systems are typically parallel and heterogeneous. They can contain a combination of multi-core CPUs, DSP subsystems, GPUs, dedicated vision computing fabrics, and hardwired functions. Their memory hierarchy can be complex, distributed, and not fully consistent across the system.

By the abstractions it provides, OpenVX intends to maximize performance portability across these hardware platforms and thus to provide high-level vision frameworks with a means to address, efficiently, current and future hardware systems with minimal impact on application source code. OpenVX contains

- a library of useful predefined or customizable vision functions,
- a graph-based execution model enabling task- and data-independent execution, as well as data tiling optimization extensions,
- a set of specific memory objects that abstract the physical memory layout and location.

Since the computer vision domain is still evolving fast, OpenVX provides an extensibility mechanism with client-defined functions that can be added to the application graph.

OpenVX consists of a C API for building, verifying, coordinating graphs execution, and for accessing memory objects. OpenVX also defines a vxu utility library which exposes each OpenVX predefined function as a C function that can be called directly, without creating a graph. The vxu does not benefit from the optimizations enabled by graphs, however, it can be used as a first, and simpler, optimization step by computer vision programmers.

1.2 Purpose

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

1.3 Scope of Specification

The scope of this document is to provide the standard by which an implementation of OpenVX will be judged to be conformant from an interface (API) point of view. This document does not contain the conformance standards for each vision function.

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 postconditions specified that are normative. If these normative conditions are not met, the behavior of the function is undefined.
1.5 Version/Change History

- 1.0 ALPHA - September 24, 2013 - Provisional Specification sent to Promoter Board for review.
- 1.0 BETA - November 30, 2013 - Provisional Specification Ratified.
- 1.0 - ??? 2014 - Final Specification Ratified.

1.6 Requirements Language

In this specification, 'shall' or 'must' is used to express a requirement that is binding, 'should' is used to express design goals or recommended actions, and 'may' is used to express an allowed behavior. All other text is explanatory or provided for information only.

1.7 Typographical Conventions

*Italics* are used in this specification to denote an emphasis on a particular concept or to denote an abstraction of a concept.

*Bold* words indicate warnings or strongly communicated concepts which are intended to draw attention to the text.

Throughout this specification, code examples may be given to highlight a particular issue. They will be given using 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/)

1.7.1 Naming Conventions

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 names 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 Programmer's Interfaces are named `vxSomeFunction()` with camel-casing 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 is done in order to minimize the possibility of name collisions and to promote 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 should follow existing international standards. For OpenVX-specified vision functions, the "function" section of the unique name is not camel-cased and uses underscores to separate words.
1.8 Glossary and Acronyms

- **FOURCC**: a 32-bit representation of an image format which is a combination of four 8-bit character codes.

- **Atomic**: The specification will occasionally mention "atomics" which is used to mean a C primitive data type. Usages which 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 term to describe a generic software abstraction in which users can override behaviors to produce application specific functionality.

- **Engine**: A term used to refer a purpose-specific software abstraction which is tunable by users.

- **Run-time**: A term used to refer to either 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.

1.9 Acknowledgements

Without the contributions from this partial list of the follow individuals from the Khronos Working Group and the companies which they represented at the time, this specification would not be possible:

- Erik Rainey - Texas Instruments, Inc.
- Susheel Gautam - QUALCOMM
- Victor Eruhimov - Itseez
- Doug Knisely - QUALCOMM
- Frank Brill - NVIDIA
- Kari Pulli - NVIDIA
- Tomer Schwartz - Broadcom Corporation
- Shervin Emami - NVIDIA
- Thierry Lepley - STMicroelectronics International NV
- Olivier Pothier - STMicroelectronics International NV
- Andy Kuzma - Intel
- Shorin Kyo - Renesas Electronics
- Renato Grottesi - ARM Limited
- Dave Schreiner - ARM Limited
- Chris Tseng - Texas Instruments, Inc.
- Daniel Laroche - Cognivue Corporation
- Andrew Garrard - Samsung Electronics
- Tomer Yanir - Samsung Electronics
- Erez Natan - Samsung Electronics
- Chang-Hyo Yu - Samsung Electronics
• Hans-Peter Nilsson - Axis Communications
• Stephen Neuendorffer - Xilinx, Inc.
• Amit Shoham - BDTi
• Paul Buxton - Imagination Technologies
• Yuki Kobayashi - Renesas Electronics
• Cormac Brick - Movidius Ltd
• Mikael Bourges-Sevenier - Aptina Imaging Corporation
• Tao Zhang - QUALCOMM
Chapter 2

Design Overview

2.1 Software Landscape

OpenVX is intended to be used either as purely the acceleration layer of many commonly-used Computer Vision Framework and/or Engines, or directly from applications when the need suits. In most cases existing APIs from various sources are already present and OpenVX would be accessed from within these existing APIs. Applications which are performance sensitive may want to consider refactoring their Computer Vision algorithms to accommodate using this API.
CHAPTER 2. DESIGN OVERVIEW

2.2 Design Objectives

This specification was designed as a framework of standardized Computer Vision functions designed to run on a wide variety of platforms which are intended to be accelerated by a Vendor's implementation on that platform. OpenVX is intended to improve the performance (in a variety of meanings) of vision applications by creating an abstraction for commonly-used vision functions, and an abstraction for aggregations of functions (a ‘graph’) and minimizing run-time overhead.

These required vision functions cover common use cases required by vision applications (e.g. object detection).

2.2.1 Hardware Optimizations

Vendors may choose to achieve this design objective through parallelism and/or specialized hardware offload techniques or any number of other methods. This specification makes no statements as to what methodology is required. This specification also makes no statement or requirements on a “level of performance” as this may vary wildly across platforms.

2.2.2 Hardware Limitations

The focus is on vision functions which are commonly known to lend themselves to an appreciable level of hardware based optimization, and are free of IP encumberance. Future versions of this specification may adopt more vision functions as part of the standard based on a broadly-available set of hardware able to accelerate said vision functions.

2.3 Assumptions

2.3.1 Portability

It is assumed that there is an upper limit to the portability of a framework across various platforms and environments. The intent is to obtain the most possible portability, while recognizing that this API is intended to be used on specific devices which have specific requirements. 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 bounds of the conformance tests.

2.3.2 Opaqueness

The API is designed to be opaque in order to not force hardware-specific requirements into any particular implementation. OpenVX is intended to address a very broad range of devices, platforms, and uses; everything from deeply embedded to desktop, and even to the distributed computing. The range of implementations are quite different and as such, the API shall only address all these spaces through opaqueness.

For example, the API does not want to dictate byte packing or alignment for structures on architectures which potentially may not be able to comply and thereby require the implementor to track two structures (one that maps to the hardware alignment and one that does not).

To avoid this issue, the API does not specify memory layout of opaque objects.

This specification does not dictate any requirements on memory allocation methods for opaque data objects. All data, except client-facing structures, are opaque and hidden behind a reference which 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 which is “imported” or “shared” from other APIs is not subsumed by OpenVX and is still maintained and accessible by the originator.

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 \texttt{vx_type_e} list. Any object may be down-cast to a \texttt{vx_reference} safely to be used in functions which require this, specifically \texttt{vxQueryReference} which can be used to get the \texttt{vx_type_e} value using an \texttt{vx_enum}.
2.5 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 is private in that the references which 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 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 which 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 which are unconnected to other sets of Nodes within the same Graph. See Graph Formalisms.

2.6 OpenVX Data Objects

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

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

- **Object: Convolution** - An opaque object which 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 which contains a manually control temporally-delayed list of objects.

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

- **Object: Image** - An opaque image object which may be some format in `vx_fourcc_e`.

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

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

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

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

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

- **Object: Threshold** - An opaque object which contains the thresholding configuration.
2.7  Error Objects

Error objects are specialized objects which may be returned from other object creator functions when serious platform issue occur (i.e. out of memory, out of handles). These are intended to be checked at the time of creation of these objects, but 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.

```text
vx_object obj = vxCreateObject(context, ...);
vx_status status = vxGetStatus((vx_reference)obj);
if (obj && 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 which are added to the graph through node creation functions. Graphs in OpenVX must be created ahead of processing time, verified by the implementation, then 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 will not need access to these intermediate data objects, they may be declared 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 some respects mentioned below.

- Inaccessible - No calls to an Access/Commit API shall succeed given a reference to an object created through a Virtual create function from a Graph external perspective. Calls to Access/Commit from within client-defined functions may succeed as they are Graph internal.
- Dimensionless or Formatless - Virtual may be declared to have no dimensions or format and they may return zeros or generic values for formats when queried.
- Scoped - Virtual data objects are scoped within the Graph they are created in. They can not be shared outside their scope.
- Intermediates - Virtual data objects should only be used for intermediate operations within Graphs since they are fundamentally inaccessible to clients of the API.
- Optimizations - Virtual data does not have to be created during Graph validation and execution and therefore may be of zero size.

These restrictions are in place to allow vendors the possibility to optimize some aspect of the data object or its usage. Some vendors may not allocate such objects, some may create intermediate sub-objects of the object, some may allocate the object on remote, inaccessible memories. OpenVX does not proscribe what 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 \texttt{vx\_int32}, \texttt{vx\_enum}, or as objects, such as \texttt{vx\_scalar}, \texttt{vx\_image}. The atomic variables of the Node creation functions shall be converted by the framework into \texttt{vx\_scalar} references for use by the Nodes. A node parameter of type \texttt{vx\_scalar} can be changed during the graph execution whereas a node parameter of an atomic type (\texttt{vx\_int32} etc) require at least a graph revalidation if changed. All node parameter objects may be modified by retrieving the reference to the \texttt{vx\_parameter} via \texttt{vx\_getParameterByIndex}, then passing that to \texttt{vx\_QueryParameter} to retrieve the reference to the object.

\begin{verbatim}
  vx\_parameter param = vx\_getParameterByIndex(node, p);
  vx\_reference ref;
  vx\_QueryParameter(param, VX\_PARAMETER\_ATTRIBUTE\_REF, &ref, sizeof(ref));
\end{verbatim}

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

\begin{verbatim}
  vx\_enum type;
  vx\_QueryParameter(param, VX\_PARAMETER\_ATTRIBUTE\_TYPE, &type, sizeof(type));
  /* cast the ref to the correct vx\_type. Atomics are now vx\_scalar */
\end{verbatim}

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 \texttt{vx\_AddParameterToGraph}. Graph parameters are used to communicate to the implementation that there are specific Node parameters which may be modified by the client between Graph executions. Additionally they are parameters which the client may set without the reference to the Node, but with the reference to the Graph using \texttt{vx\_SetGraphParameterByIndex}. This allows for the Graph authors to construct \texttt{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 \texttt{vx\_ProcessGraph} will block until the graph has completed) and in Asynchronous single-issue-per-reference mode (via \texttt{vx\_ScheduleGraph} and \texttt{vx\_WaitGraph}).

Asynchronous Mode

In asynchronous mode, Graphs must be single issue per reference. This means that given a constructed graph reference \texttt{G}, it may be scheduled multiple times but will only execute 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 series based on the behavior or the vendor’s implementation.

2.8.6 Graph Formalisms

In order to use graphs several rules must be put in place to allow deterministic execution of Graphs. The behavior of a \texttt{processGraph(G)} call is determined by the structure of the Processing Graph \texttt{G}. The Processing Graph is a bipartite graph consisting of a set of Nodes \texttt{N}_1 \ldots \texttt{N}_n and a set of data objects \texttt{D}_1 \ldots \texttt{D}_i. Each edge \texttt{(N}_x, \texttt{D}_y) in the graph represents a data object \texttt{D}_y that is written by Node \texttt{N}_x, and each edge \texttt{(D}_x, \texttt{N}_y) represents a data object \texttt{D}_x that is read by Node \texttt{N}_y. Each edge \texttt{e} has a name \texttt{Name(e)}, which gives the parameter name of the node that references the corresponding data object. Each Node Parameter also as 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 \texttt{(N}_x, \texttt{D}_y) requires Type\texttt{(N}_x, \texttt{Name(N}_x, \texttt{D}_y)) in \{OUTPUT, INOUT\}
2. (Input typing) Every input edge \texttt{(N}_x, \texttt{D}_y) requires Type\texttt{(N}_x, \texttt{Name(D}_x, \texttt{N}_y)) 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 $(D_x, N_y)$ 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 $D_y$ $(N_x, D_y)$

6. (Bidirectional data objects shall not be virtual) If Type$(N_x, \text{Name}(N_x, D_y))$ is INOUT implies $D_y$ is non-Virtual.

7. (Delay data objects shall not be virtual) If $D_x$ is Delay then it shall not be Virtual.

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 $(D_x, N_y)$ from non-Virtual objects $D_x$ are marked (parameters must be set).
- all input edges $(D_x, N_y)$ with an output edge $(N_z, D_x)$ are unmarked
- all input edges $(D_x, N_y)$ where $D_x$ 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 $(N_x, D_y)$ where $D_y$ is not a Delay results in marking all of the input edges $(D_y, N_x)$. 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 $N_1 ... N_x$, and an edge $(N_x, N_z)$ for every pair of edges $(N_x, D_y)$ and $(D_y, N_z)$ where $D_y$ is not a delay. Then unconditionally fire each node according to any topological sort of $P$.

Following assertions should be verified:

- $P$ is a 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.

### 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 $\text{vxMagnitudeNode}$ and $\text{vxPhaseNode}$ 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 vendor of the OpenVX implementation.
The code to construct such a graph can be seen below.

```c
vx_context context = vxCreateContext();
vx_image images[] = {
    vxCreateImage(context, 640, 480, FOURCC_U8),
    vxCreateImage(context, 640, 480, FOURCC_U8),
    vxCreateImage(context, 640, 480, FOURCC_U8),
};

vx_graph graph = vxCreateGraph(context);
vx_image virts[] = {
    vxCreateVirtualImage(graph, 0, 0, FOURCC_VIRT),
    vxCreateVirtualImage(graph, 0, 0, FOURCC_VIRT),
    vxCreateVirtualImage(graph, 0, 0, FOURCC_VIRT),
    vxCreateVirtualImage(graph, 0, 0, FOURCC_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 */
```

## 2.8.8 Verification

Graphs within OpenVX must go through a rigorous validation process before execution in order to satisfy the design concept of eliminating run-time overhead (parameter checking) which will guarantee 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 which has algorithmic significance must be verified. In the case of a scalar value, it may need to be ranged checked (e.g. \(0.5 \leq k \leq 1.0\)). 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 DAG (Directed, Acyclic Graph). 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.

### 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 in unspecified. Nodes which are arranged in a serial fashion due to data dependencies will perform callbacks in order. The callback function may use the node reference to extract parameters from the node, then extract the data references. Data outputs of Nodes with callbacks shall be available (via Access/Commit methods) when the callback is called.

### 2.10 Client Defined Functions (CDF)

OpenVX supports the concept of *client-defined functions* which 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, the graph does not have to be halted to execute client-supplied functions 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 nodes must aid in the Graph Verification effort by providing explicit validation functions for each vision function they implement. Each parameter passed to the instanced Node of a Client Defined Function will be validated using the client-supplied validation functions. The client must check these attributes and/or values of each parameter:

- If an attribute or value of the parameter has algorithmic significance, it 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 (within the output validator).

Input validators will execute before output validators. This allows any or all inputs to be used as dependents of output parameter validation.

The Meta Format Object

The Meta Format Object is a 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 Client Defined Function Naming Conventions

Client Defined Functions must be export 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 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_KHRONOS, 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.

In order 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 (optional, but very helpful for long enumerations).

2.11 Immediate Mode Functions

OpenVX also contains an interface defined within `<VX/vxu.h>` which 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, notably vxuHalf-ScaleGaussian3x3. 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 Base Vision Functions

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

2.12.1 Inputs

<table>
<thead>
<tr>
<th>Vision Function</th>
<th>U8</th>
<th>U16</th>
<th>S16</th>
<th>S32</th>
<th>U32</th>
<th>F32</th>
<th>4CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsDiff</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accumulate</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accumulate-Squared</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accumulate-Weighted</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>1.0</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Box3x3</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Canny-Edge-Detector</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
###CHAPTER 2. DESIGN OVERVIEW

<table>
<thead>
<tr>
<th>Function</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel-Combine</td>
<td>1.0</td>
</tr>
<tr>
<td>Channel-Extract</td>
<td>1.0</td>
</tr>
<tr>
<td>Channel-ExtractRef</td>
<td>1.0</td>
</tr>
<tr>
<td>Color-Convert</td>
<td>1.0</td>
</tr>
<tr>
<td>Convert-Image-Depth</td>
<td>1.0</td>
</tr>
<tr>
<td>Convolve</td>
<td>1.0</td>
</tr>
<tr>
<td>Dilate3x3</td>
<td>1.0</td>
</tr>
<tr>
<td>Equalize-Histogram</td>
<td>1.0</td>
</tr>
<tr>
<td>Erode3x3</td>
<td>1.0</td>
</tr>
<tr>
<td>Fast-Corners</td>
<td>1.0</td>
</tr>
<tr>
<td>Gaussian3x3</td>
<td>1.0</td>
</tr>
<tr>
<td>Harris-Corners</td>
<td>1.0</td>
</tr>
<tr>
<td>HalfScale-Gaussian3x3</td>
<td>1.0</td>
</tr>
<tr>
<td>Histogram</td>
<td>1.0</td>
</tr>
<tr>
<td>Integral-Image</td>
<td>1.0</td>
</tr>
<tr>
<td>Inv</td>
<td>1.0</td>
</tr>
<tr>
<td>Table-Lookup</td>
<td>1.0</td>
</tr>
<tr>
<td>Magnitude</td>
<td>1.0</td>
</tr>
<tr>
<td>MeanStd-Dev</td>
<td>1.0</td>
</tr>
<tr>
<td>Median3x3</td>
<td>1.0</td>
</tr>
<tr>
<td>MinMax-Loc</td>
<td>1.0</td>
</tr>
<tr>
<td>Multiply</td>
<td>1.0</td>
</tr>
<tr>
<td>Optical-FlowLK</td>
<td>1.0</td>
</tr>
<tr>
<td>Or</td>
<td>1.0</td>
</tr>
<tr>
<td>Phase</td>
<td>1.0</td>
</tr>
<tr>
<td>Pyramid</td>
<td>1.0</td>
</tr>
<tr>
<td>Remap</td>
<td>1.0</td>
</tr>
<tr>
<td>Scale-Image</td>
<td>1.0</td>
</tr>
<tr>
<td>Sobel3x3</td>
<td>1.0</td>
</tr>
<tr>
<td>Subtract</td>
<td>1.0</td>
</tr>
<tr>
<td>Threshold</td>
<td>1.0</td>
</tr>
<tr>
<td>Undistort</td>
<td>1.0</td>
</tr>
<tr>
<td>WarpAffine</td>
<td>1.0</td>
</tr>
<tr>
<td>Warp-Perspective</td>
<td>1.0</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Vision Function</th>
<th>U8</th>
<th>U16</th>
<th>S16</th>
<th>U32</th>
<th>S32</th>
<th>F32</th>
<th>4CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsDiff</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accumulate</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accumulate-Squared</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accumulate-Weighted</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>1.0</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Box3x3</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Canny-Edge-Detector</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel-Combine</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Channel-Extract</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel-ExtractRef</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convert</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Convert-Image-Depth</td>
<td>1.0</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>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dilate3x3</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equalize-Histogram</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Erode3x3</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fast-Corners</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaussian3x3</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harris-Corners</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HalfScale-Gaussian3x3</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Histogram</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>Integral-Image</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>Inv</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Table-Lookup</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.13 Lifecycles

2.13.1 OpenVX Context Lifecycle

The lifecycle of the context is very simple.

![Figure 2.4: The lifecycle model for an OpenVX Context.]

2.13.2 Graph Lifecycle

OpenVX has four main phases of graph lifecycle:

- **Construction** - Graphs are created via `vxCreateGraph`, Nodes are connected together by with data objects.

- **Verification** - The graphs are checked for consistency, correctness and other conditions. Memory allocation may occur.

- **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 2.5: Graph Lifecycle**

### 2.13.3 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 via `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.

![Image Lifecycle Diagram](image)

**Figure 2.6: Image Object Lifecycle**
2.14 Host Memory Data Object Access Patterns

For objects which are retrieved from OpenVX which are 2D in nature such as `vx_image`, `vx_matrix`, and `vx convolution`, the manner in which the host-side has access to these memory region is well defined. OpenVX uses a row-major storage (that is each unit in a column is memory adjacent to it's row adjacent unit). Two dimensional objects are always declared in width (columns) by height (rows) notation (vxCreateImage or vxCreateMatrix). Therefore when accessing these structures in 'C' with arrays of declared size, users must declare dimensions in the reverse order. 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.

2.14.1 Matrix Access Example

```c
const vx_size columns = 3;
const vx_size rows = 4;
vx_matrix matrix = vxCreateMatrix(context,
    VX_TYPE_FLOAT32, columns, rows);
if (matrix) {
    #if defined(OPENVX_USE_C99)
        vx_float32 mat[rows][columns]; /* note: row major */
    #else
        vx_float32 *mat = (vx_float32 *)malloc(rows*columns*sizeof(vx_float32));
    #endif
    if (vxAccessMatrix(matrix, mat) == VX_SUCCESS) {
        for (j = 0; j < rows; j++)
            for (i = 0; i < columns; i++)
                #if defined(OPENVX_USE_C99)
                    mat[j][i] = (vx_float32)rand()/(vx_float32)RAND_MAX;
                #else
                    mat[j*columns + i] = (vx_float32)rand()/(vx_float32)RAND_MAX;
                #endif
        vxCommitMatrix(matrix, mat);
    }
    #if !defined(OPENVX_USE_C99)
        free(mat);
    #endif
}
```

2.14.2 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, FOURCC_U8);
vx_rectangle_2 rect;
vx_imagepatch_addressing t addr;
rect.start.x = rect.start.y = 0;
rect.end.x = rect.end.y = PATCH_DIM;
status = vxAccessImagePatch(image, &rect, plane, &addr, &base_ptr,
    VX_READ_AND_WRITE);
if (status == VX_SUCCESS) {
    vx_uint8 *ptr2 = vxFormatImagePatchAddress2d(base_ptr, x, y, &addr);
    *ptr2 = pixel;
}
```
CHAPTER 2. DESIGN OVERVIEW

2.14.3 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 = 0UL;
void *base = NULL;
/* access entire array at once */
vxAccessArrayRange(array, 0, num_items, &stride, &base, VX_READ_AND_WRITE);
for (i = 0; i < num_items; i++)
{
    vxArrayItem(mystruct, base, i, stride).some_uint += i;
    vxArrayItem(mystruct, base, i, stride).some_double = 3.14f;
}
vxCommitArrayRange(array, 0, num_items, base);
```

Access/Commit 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;
    vxAccessArrayRange(array, i, i+1, &stride, (void **)&myptr, VX_READ_AND_WRITE);
    myptr->some_uint += 1;
    myptr->some_double = 3.14f;
    vxCommitArrayRange(array, i, i+1, (void *)myptr);
}
```

2.15 Extending OpenVX

Beyond Client Defined Functions (CDF) there are other mechanisms for vendors to extend features OpenVX. The mechanisms are not available to CDFs.

2.15.1 Extending Attributes

When extending attributes, vendors must use their assigned id from `vx_vendor_id` 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:
2.15.2 Vendor Custom Kernels

Vendors will also undoubtedly add more kernels to the base set supplied to OpenVX. They should provide a header of the form

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

which 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_KHRONOS, VX_LIBRARY_XYZ) + 0x0,
    // up to 0xFFF kernel enums can be created.
  }
  ```

- **A new VXU Function per function.**

  ```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 sort of extensions do not require linking to new objects to facilitate usage.

2.15.3 Vendor Custom Extensions

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

```c
vx_char *tmp, *extensions = NULL;
vx_size size = 0ul;
vxQueryContext(context, VX_CONTEXT_ATTRIBUTE_EXTENSIONS_SIZE,
               &size, sizeof(size));
extensions = malloc(size);
vxQueryContext(context, VX_CONTEXT_ATTRIBUTE_EXTENSIONS,
               extensions, size);
```

An extension in this list may have a header and new kernels or framework feature or data objects or may not, they are dependent on the extension itself. The common feature is that they are implemented and supported by the vendor of the implementation.

2.15.4 Hinting

The specification defines a Hinting API which allows Clients to fed 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 vendor of the implementation of OpenVX for information on vendor-specific extensions.

2.15.5 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.
2.16 Known Extensions to OpenVX

2.16.1 User Node Tiling

The User Node Tiling facility enables optimizations of the user nodes (e.g. locality of execution or parallelism) when performing computation on the image data. Modern processors have a diverse memory hierarchy that varies from relatively small but fast and expensive memory to relatively large but slow and inexpensive memory. Image data is typically too large to fit into the fast but small memory. The ability to break the image data into smaller sized units allows for optimized computation on these smaller units with fast memory access or parallel execution of a user node on multiple image tiles simultaneously. The OpenVX Graph Manager possesses the knowledge about the memory hierarchy of the platform and is hence in a position to break the image data into smaller units for memory optimization. Knowledge of the memory access pattern of an algorithm is key for the graph manager to enable optimizations.

The Khronos OpenVX Working Group will include this extension as part of the OpenVX 1.1 specification, contingent on community feedback.
Chapter 3
Module Documentation

3.1 Vision Functions

3.1.1 Detailed Description

These are the base vision functions supported in OpenVX 1.0. These functions were chosen as a subset of a larger pool of possible functions which fall under the following criteria:

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

Modules

- Function: Absolute Difference
  Computes the absolute difference between two images.
- Function: Accumulate
  Accumulates an input image into output image.
- Function: Accumulate Squared
  Accumulates a squared value from an input image to an output image.
- Function: Accumulate Weighted
  Accumulates a weighted value from an input image to an output image.
- Function: Arithmetic Addition
  Performs addition between two images.
- Function: Arithmetic Subtraction
  Performs subtraction between two images.
- Function: Bitwise And
  Performs bitwise "and" between two \texttt{FOURCC\_U8} images.
- Function: Bitwise Exclusive Or
  Performs bitwise "exclusive or" between two \texttt{FOURCC\_U8} images.
- Function: Bitwise Inclusive Or
  Performs bitwise "inclusive or" between two \texttt{FOURCC\_U8} images.
- Function: Bitwise Not
  Performs bitwise "not" on a \texttt{FOURCC\_U8} input image.
- Function: Box Filter
  Compute a Box filter over a window of the input image.
- Function: Canny Edge Detector
  Canny Edge detection kernel.
- Function: Channel Combine
  The Channel Combine Kernel.
• Function: Channel Extract
  The Channel Extraction Kernel.
• Function: Color Convert
  The Color Conversion Kernel.
• Function: Convert Bit depth
  Converts image bit depth.
• Function: Custom Convolution
  Convolves the input with the client supplied convolution matrix.
• Function: Dilate Image
  Dilation "grows" the white space in a \texttt{FOURCC} \texttt{U8} "bool" image.
• Function: Equalize Histogram
  Equalizes the histogram of a grayscale image.
• Function: Erode Image
  Erosion "shrinks" the white space in a \texttt{FOURCC} \texttt{U8} "bool" image.
• Function: Fast Corners
  Computes the corners in an image using FAST algorithm.
• Function: Gaussian Filter
  Computes a Gaussian filter over a window of the input image.
• Function: Gaussian Image Pyramid
  Computes a Gaussian Image Pyramid from an input image.
• Function: Harris Corners
  Computes the Harris Corners of an image.
• Function: Histogram
  Generates a distribution from an image.
• Function: Integral Image
  Computes the integral image of the input.
• Function: Magnitude
  The Gradient Magnitude Computation Kernel.
• Function: 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).
• Function: Median Filter
  Compute a median pixel value over a window of the input image.
• Function: Min, Max Location
  Finds the minimum and maximum values in an image and a location for each.
• Function: Optical Flow Pyramid (LK)
  Computes the optical flow using the Lucas-Kanade method between two pyramid images.
• Function: Phase
  The Gradient Phase Computation Kernel.
• Function: Pixel-wise Multiplication
  Performs element-wise multiplication between two images and a scalar value.
• Function: Remap
  Maps output pixels in an image from input pixels in an image.
• Function: Scale Image
  The Image Resizing Kernel.
• Function: Sobel 3x3
  The Sobel Image Filter Kernel.
• Function: TableLookup
  The Table Lookup Image Kernel.
• Function: Thresholding
  Thresholds an input image and produces an output boolean image.
• Function: Warp Affine
  Performs an affine transform on an image.
• Function: Warp Perspective
  Performs a perspective transform on an image.
3.2 Function: Absolute Difference

3.2.1 Detailed Description

Computes the absolute difference between two images. Absolute Difference is computed by:

\[ out(x,y) = |in1(x,y) - in2(x,y)| \]

Functions

- **VX_API vx_node vxAbsDiffNode (vx_graph graph, vx_image in1, vx_image in2, vx_image out)**
  
  [Graph] Creates an AbsDiff node.

- **VX_API vx_status 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

**VX_API vx_node 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>in1</td>
<td>in2</td>
<td>An input image in FOURCC_U8</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The output image in FOURCC_U8</td>
</tr>
</tbody>
</table>

**Returns**

vx_node

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>Node could not be created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Node Handle</td>
</tr>
</tbody>
</table>

**VX_API vx_status 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>in1</td>
<td>in2</td>
<td>An input image</td>
</tr>
<tr>
<td>out</td>
<td>out</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 vx_status_e.</td>
</tr>
</tbody>
</table>
3.3 Function: Accumulate

3.3.1 Detailed Description

Accumulates an input image into output image. Accumulation is computed by:

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

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

Functions

- \text{VX\_API vx\_node vxAccumulateImageNode (vx\_graph graph, vx\_image input, vx\_image accum)}
  
  [Graph] Creates an accumulate node.

- \text{VX\_API vx\_status vxuAccumulateImage (vx\_context context, vx\_image input, vx\_image accum)}
  
  [Immediate] Creates an accumulate node.

3.3.2 Function Documentation

\text{VX\_API vx\_node vxAccumulateImageNode (vx\_graph graph, vx\_image input, vx\_image accum)}

[Graph] Creates an accumulate node.

Parameters

- \text{in graph} The reference to the graph.
- \text{in input} The input \text{FOURCC\_U8} image.
- \text{in,out accum} The accumulation image in \text{FOURCC\_S16}.

Returns

- \text{vx\_node}

Return values

\[\begin{array}{|c|l|}
\hline
0 & \text{Node could not be created.} \\
\hline
\ast & \text{Node Handle} \\
\hline
\end{array}\]

\text{VX\_API vx\_status vxuAccumulateImage (vx\_context context, vx\_image input, vx\_image accum)}

[Immediate] Creates an accumulate node.

Parameters

- \text{in context} The reference to the overall context.
- \text{in input} The input \text{FOURCC\_U8} image.
- \text{in,out accum} The accumulation image in \text{FOURCC\_S16}.

Returns

- A \text{vx\_status\_e} enumeration.

Return values

\[\begin{array}{|c|l|}
\hline
\text{VX\_SUCCESS} & \text{Success} \\
\hline
\ast & \text{An error occurred. See \text{vx\_status\_e}.} \\
\hline
\end{array}\]
3.4 Function: Accumulate Squared

3.4.1 Detailed Description
Accumulates a squared value from an input image to an output image. Accumulate squares is computed by:

\[
\text{accum}(x, y) = \text{accum}(x, y) + \text{scale} \times \text{input}(x, y)^2
\]

Where \(0 \leq \text{scale}\)
The overflow policy used is \texttt{VX_CONVERT_POLICY_SATURATE}

Functions

- \texttt{VX\_API vx\_node vxAccumulateSquareImageNode (vx\_graph \textit{graph}, vx\_image \textit{input}, vx\_scalar \textit{scalar}, vx\_image \textit{accum})}
  \texttt{[Graph]} Creates an accumulate square node.
- \texttt{VX\_API vx\_status vxuAccumulateSquareImage (vx\_context \textit{context}, vx\_image \textit{input}, vx\_float32 \textit{scale}, vx\_image \textit{accum})}
  \texttt{[Immediate]} Creates an accumulate square node.

3.4.2 Function Documentation

\texttt{VX\_API vx\_node vxAccumulateSquareImageNode (vx\_graph \textit{graph}, vx\_image \textit{input}, vx\_scalar \textit{scalar}, vx\_image \textit{accum})}
\texttt{[Graph]} Creates an accumulate square node.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{graph}</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>\textit{input}</td>
<td>The input \texttt{FOURCC_U8} image.</td>
</tr>
<tr>
<td>\textit{scalar}</td>
<td>scalar The input \texttt{VX_TYPE_FLOAT32} scalar with the range (0.0f \leq \text{scalar} \leq 1.0f).</td>
</tr>
<tr>
<td>\textit{accum}</td>
<td>The accumulation image in \texttt{FOURCC_S16}.</td>
</tr>
</tbody>
</table>

Returns \texttt{vx\_node}

Return values

<table>
<thead>
<tr>
<th>Return 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>

\texttt{VX\_API vx\_status vxuAccumulateSquareImage (vx\_context \textit{context}, vx\_image \textit{input}, vx\_float32 \textit{scale}, vx\_image \textit{accum})}
\texttt{[Immediate]} Creates an accumulate square node.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{context}</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>\textit{input}</td>
<td>The input \texttt{FOURCC_U8} image.</td>
</tr>
<tr>
<td>\textit{scale}</td>
<td>The input scalar with the range (0.0 \leq \text{scale} \leq 1.0).</td>
</tr>
<tr>
<td>\textit{accum}</td>
<td>The accumulation image in \texttt{FOURCC_S16}.</td>
</tr>
</tbody>
</table>

Returns \texttt{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>
3.5 Function: Accumulate Weighted

3.5.1 Detailed Description

Accumulates a weighted value from an input image to an output image. Weighted accumulation is computed by:

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

Where \(0 \leq \alpha \leq 1\) Conceptually the rounding for this is defined as: \(\text{output}(x,y) = \text{uint8}( (1 \ast \alpha) \ast \text{float}( \text{int32}(\text{output}(x,y))) + \alpha \ast \text{float}( \text{int32}(\text{input}(x,y))) )\)

Functions

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

- **VX_API vx_status vxuAccumulateWeightedImage (vx_context context, vx_image input, vx_float32 alpha, vx_image accum)**
  - [Immediate] Creates a weighted accumulate node.

3.5.2 Function Documentation

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

[Graph] Creates a weighted accumulate 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 input FOURCC_U8 image.</td>
</tr>
<tr>
<td>in alpha</td>
<td>The input VX_TYPE_FLOAT32 alpha value with the range (0.0 \leq \alpha \leq 1.0).</td>
</tr>
<tr>
<td>in,out accum</td>
<td>The FOURCC_U8 accumulation image.</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_API vx_status vxuAccumulateWeightedImage ( vx_context context, vx_image input, vx_float32 alpha, vx_image accum )**

[Immediate] Creates a weighted accumulate node.

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 FOURCC_U8 image.</td>
</tr>
<tr>
<td>in alpha</td>
<td>The input alpha value with the range (0.0 \leq \alpha \leq 1.0).</td>
</tr>
<tr>
<td>in,out accum</td>
<td>The FOURCC_U8 accumulation 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>
3.6 Function: Arithmetic Addition

3.6.1 Detailed Description

Performs addition between two images. Arithmetic addition is performed between the pixel values in two \texttt{FOURCC\_U8} or \texttt{FOURCC\_S16} images. The output image can be \texttt{FOURCC\_U8} only if both source images are \texttt{FOURCC\_U8} and the output image is explicitly set to \texttt{FOURCC\_U8}. It is otherwise \texttt{FOURCC\_S16}. If one of the input images is of type \texttt{FOURCC\_S16}, all values are converted to \texttt{FOURCC\_S16}. The overflow handling is controlled by an overflow-policy parameter. For each pixel value in the two input images:

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

Functions

- \texttt{VX\_API vx\_node vxAddNode (vx\_graph graph, vx\_image in1, vx\_image in2, vx\_enum policy, vx\_image out)}
  
  [Graph] Creates an arithmetic addition node.

- \texttt{VX\_API vx\_status 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.

3.6.2 Function Documentation

\texttt{VX\_API vx\_node 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{FOURCC_U8} or \texttt{FOURCC_S16}.</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>An input image, \texttt{FOURCC_U8} or \texttt{FOURCC_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{FOURCC_U8} or \texttt{FOURCC_S16} image.</td>
</tr>
</tbody>
</table>

Returns

\texttt{vx\_node}

Return values

| 0  | Node could not be created. |
| #  | Node Handle               |

\texttt{VX\_API vx\_status 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

<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 \texttt{FOURCC_U8} or \texttt{FOURCC_S16} input image.</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>A \texttt{FOURCC_U8} or \texttt{FOURCC_S16} input image.</td>
</tr>
<tr>
<td>in</td>
<td>policy</td>
<td>A \texttt{vx_convert_policy_e} enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The output image in \texttt{FOURCC_U8} or \texttt{FOURCC_S16} format.</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 vx_status_e.</td>
</tr>
</tbody>
</table>
3.7 Function: Arithmetic Subtraction

3.7.1 Detailed Description

Performs subtraction between two images. Arithmetic subtraction is performed between the pixel values in two FOURCC_U8 or FOURCC_S16 images. The output image can be FOURCC_U8 only if both source images are FOURCC_U8 and the output image is explicitly set to FOURCC_U8. It is otherwise FOURCC_S16. If one of the input images is of type FOURCC_S16, all values are converted to FOURCC_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_API vx_node vxSubtractNode** (vx_graph graph, vx_image in1, vx_image in2, vx_enum policy, vx_image out)
  
  [Graph] Creates an arithmetic subtraction node.

- **VX_API vx_status 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.

3.7.2 Function Documentation

**VX_API vx_node 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, FOURCC_U8 or FOURCC_S16, the minuend.</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>An input image, FOURCC_U8 or FOURCC_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 FOURCC_U8 or FOURCC_S16 image.</td>
</tr>
</tbody>
</table>

Returns

vx_node

Return values

| 0 | Node could not be created. |
| # | Node Handle |

**VX_API vx_status 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**

<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 FOURCC_U8 or FOURCC_S16 input image, the minuend.</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>A FOURCC_U8 or FOURCC_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 FOURCC_U8 or FOURCC_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 \texttt{vx_status_e}.</td>
</tr>
</tbody>
</table>
3.8 Function: Bitwise And

3.8.1 Detailed Description

Performs bitwise “and” between two FOURCC_U8 images. Bitwise “and” is computed by, for each bit in each pixel in the input images:

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

Or expressed as “C” code:

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

Functions

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

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

  [Immediate] Computes the bitwise and between two images.

3.8.2 Function Documentation

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

[Graph] Creates a bitwise-and 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>A FOURCC_U8 input image</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>A FOURCC_U8 input image</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The FOURCC_U8 output image.</td>
</tr>
</tbody>
</table>

Returns

vx_node

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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_API vx_status 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>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 FOURCC_U8 input image</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>A FOURCC_U8 input image</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The FOURCC_U8 output image.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Success</td>
</tr>
</tbody>
</table>
An error occurred. See $\texttt{vx\_status\_e}$. 

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>An error occurred. See $\texttt{vx_status_e}$.</td>
</tr>
</tbody>
</table>
3.9 Function: Bitwise Exclusive Or

3.9.1 Detailed Description
Performs bitwise "exclusive or" between two FOURCC_U8 images. Bitwise "exclusive or" is computed by, 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:

\[ out(x, y) = in_1(x, y) \text{ ^ } in_2(x, y) \]

Functions

- VX_API vx_status vxuXor (vx_context context, vx_image in1, vx_image in2, vx_image out)
  [Immediate] Computes the bitwise exclusive-or between two images.
- VX_API vx_node vxXorNode (vx_graph graph, vx_image in1, vx_image in2, vx_image out)
  [Graph] Creates a bitwise exclusive-or node.

3.9.2 Function Documentation

VX_API vx_status vxuXor ( vx_context context, vx_image in1, vx_image in2, vx_image out )

[Immediate] Computes the bitwise exclusive-or 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>in1</td>
<td>in1</td>
<td>A FOURCC_U8 input image</td>
</tr>
<tr>
<td>in2</td>
<td>in2</td>
<td>A FOURCC_U8 input image</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The FOURCC_U8 output image.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

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

VX_API vx_node vxXorNode ( vx_graph graph, vx_image in1, vx_image in2, vx_image out )

[Graph] Creates a bitwise exclusive-or node.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in1</td>
<td>in1</td>
<td>A FOURCC_U8 input image</td>
</tr>
<tr>
<td>in2</td>
<td>in2</td>
<td>A FOURCC_U8 input image</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The FOURCC_U8 output image.</td>
</tr>
</tbody>
</table>

Returns

vx_node

Return values

| 0   | Node could not be created. |
|   | Node Handle |
3.10 Function: Bitwise Inclusive Or

3.10.1 Detailed Description
Performs bitwise "inclusive or" between two FOURCC_U8 images. Bitwise "inclusive or" is computed by, for each bit in each pixel in the input images:

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

Or expressed as "C" code:

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

Functions
- **VX_API vx_node vxOrNode (vx_graph graph, vx_image in1, vx_image in2, vx_image out)**
  [Graph] Creates a bitwise inclusive-or node.
- **VX_API vx_status vxuOr (vx_context context, vx_image in1, vx_image in2, vx_image out)**
  [Immediate] Computes the bitwise inclusive-or between two images.

3.10.2 Function Documentation
**VX_API vx_node vxOrNode ( vx_graph graph, vx_image in1, vx_image in2, vx_image out )**
[Graph] Creates a bitwise inclusive-or node.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in1</td>
<td>A FOURCC_U8 input image</td>
</tr>
<tr>
<td>in2</td>
<td>A FOURCC_U8 input image</td>
</tr>
<tr>
<td>out</td>
<td>The FOURCC_U8 output image.</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_API vx_status vxuOr ( vx_context context, vx_image in1, vx_image in2, vx_image out )**
[Immediate] Computes the bitwise inclusive-or between two images.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>in1</td>
<td>A FOURCC_U8 input image</td>
</tr>
<tr>
<td>in2</td>
<td>A FOURCC_U8 input image</td>
</tr>
<tr>
<td>out</td>
<td>The FOURCC_U8 output image.</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>
</tbody>
</table>
An error occurred. See \texttt{vx\_status.e}. 
3.11 Function: Bitwise Not

3.11.1 Detailed Description

Performs bitwise "not" on a \texttt{FOURCC\_U8} input image. Bitwise "not" is computed by, for each bit in each pixel in the input image:

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

Functions

- \texttt{VX\_API vx\_node vxNotNode (vx\_graph graph, vx\_image input, vx\_image output)} 
  [Graph] Creates a bitwise-not node.
- \texttt{VX\_API vx\_status vxuNot (vx\_context context, vx\_image input, vx\_image output)} 
  [Immediate] Computes the bitwise not of an image.

3.11.2 Function Documentation

\texttt{VX\_API vx\_node 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 \texttt{graph}</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in \texttt{input}</td>
<td>A \texttt{FOURCC_U8} input image</td>
</tr>
<tr>
<td>out \texttt{output}</td>
<td>The \texttt{FOURCC_U8} output image.</td>
</tr>
</tbody>
</table>

Returns

\texttt{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>

\texttt{VX\_API vx\_status 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 \texttt{context}</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>in \texttt{input}</td>
<td>The \texttt{FOURCC_U8} input image</td>
</tr>
<tr>
<td>out \texttt{output}</td>
<td>The \texttt{FOURCC_U8} output image.</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>VX_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>*</td>
<td>An error occurred. See \texttt{vx_status_e}.</td>
</tr>
</tbody>
</table>
3.12 Function: Box Filter

3.12.1 Detailed Description

Compute a Box filter over a window of the input image. This filter uses the following convolution matrix:

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

Functions

- **VX_API vx_node vxBox3x3Node (vx_graph graph, vx_image input, vx_image output)**
  [Graph] Creates a Box Filter Node.

- **VX_API vx_status vxuBox3x3 (vx_context context, vx_image input, vx_image output)**
  [Immediate] Computes a box filter on the image by a 3x3 window.

3.12.2 Function Documentation

**VX_API vx_node vxBox3x3Node ( vx_graph graph, vx_image input, vx_image output )**

[Graph] Creates a Box 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 FOURCC_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in FOURCC_U8 format.</td>
</tr>
</tbody>
</table>

Returns

- vx_node

Return values

| 0   | Node could not be created. |
| *   | Node Handle |

**VX_API vx_status vxuBox3x3 ( vx_context context, vx_image input, vx_image output )**

[Immediate] Computes a box 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 FOURCC_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in FOURCC_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>
3.13 Function: Canny Edge Detector

3.13.1 Detailed Description

Canny Edge detection kernel. This function implements an edge detection algorithm similar to that described in [2]. 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:
    
    \[
    \text{sobel} = \begin{bmatrix}
    1 & 0 & -1 \\
    2 & 0 & -2 \\
    1 & 0 & 1 \\
    \end{bmatrix}
    \]
  
  - For gradient size 5:
    
    \[
    \text{sobel} = \begin{bmatrix}
    1 & 2 & 0 & -2 & -1 \\
    4 & 8 & 0 & -8 & -4 \\
    6 & 12 & 0 & -12 & -6 \\
    4 & 8 & 0 & -8 & -4 \\
    1 & 2 & 0 & -2 & -1 \\
    \end{bmatrix}
    \]
  
  - For gradient size 7:
    
    \[
    \text{sobel} = \begin{bmatrix}
    1 & 4 & 5 & 0 & -5 & -4 & -1 \\
    6 & 24 & 30 & 0 & -30 & -24 & -6 \\
    15 & 60 & 75 & 0 & -75 & -60 & -15 \\
    20 & 80 & 100 & 0 & -100 & -80 & -20 \\
    15 & 60 & 75 & 0 & -75 & -60 & -15 \\
    6 & 24 & 30 & 0 & -30 & -24 & -6 \\
    1 & 4 & 5 & 0 & -5 & -4 & -1 \\
    \end{bmatrix}
    \]

- **Non-Maximum Suppression**: This is then applied such that a pixel is retained as a potential edge pixel if an only if its magnitude is greater than 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. If two adjacent pixels are both local maxima according to the above criteria with the same value, the non-maximum-suppression algorithm should retain both of them as edge pixels.

- **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 {
  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_API vx_node 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_API vx_status 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.

3.13.2 Enumeration Type Documentation

`enum vx_norm_type_e`  
A normalization type.

See Also

Function: Canny Edge Detector

Enumerator

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

Definition at line 1082 of file `vx_types.h`.

3.13.3 Function Documentation

`VX_API vx_node 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

<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 FOURCC_U8 image.</td>
</tr>
<tr>
<td><code>hyst</code></td>
<td>The double threshold for hysteresis.</td>
</tr>
<tr>
<td><code>gradient_size</code></td>
<td>The size of the Sobel filter window, must support at least 3, 5 and 7.</td>
</tr>
<tr>
<td><code>norm_type</code></td>
<td>A flag indicating the norm used to compute the gradient, VX_NORM_L1 or VX_NORM_L2.</td>
</tr>
<tr>
<td><code>output</code></td>
<td>The output image in FOURCC_U8 format with values either 0 or 255.</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_API vx_status 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 FOURCC_U8 image.</td>
</tr>
<tr>
<td>in</td>
<td>hyst</td>
<td>The double threshold for hysteresis.</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 FOURCC_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>
3.14 Function: Channel Combine

3.14.1 Detailed Description

The Channel Combine Kernel. This kernel takes multiple FOURCC_U8 planes to recombine them into a multi-planar or interleaved format from vx_fourcc_e.

Functions

- **VX_API vx_node 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_API vx_status 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.

3.14.2 Function Documentation

**VX_API vx_node 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 which will form channel 0. Must be FOURCC_U8.</td>
</tr>
<tr>
<td>in</td>
<td>plane1</td>
<td>The plane which will form channel 1. Must be FOURCC_U8.</td>
</tr>
<tr>
<td>in</td>
<td>plane2</td>
<td>[optional] The plane which will form channel 2. Must be FOURCC_U8.</td>
</tr>
<tr>
<td>in</td>
<td>plane3</td>
<td>[optional] The plane which will form channel 3. Must be FOURCC_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.</td>
</tr>
</tbody>
</table>

See Also

**VX_KERNEL_CHANNEL_COMBINE**

Returns

vx_node

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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_API vx_status 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 which will form channel 0. Must be FOURCC_U8.</td>
</tr>
<tr>
<td>in</td>
<td>plane1</td>
<td>The plane which will form channel 1. Must be FOURCC_U8.</td>
</tr>
<tr>
<td>in</td>
<td>plane2</td>
<td>[optional] The plane which will form channel 2. Must be FOURCC_U8.</td>
</tr>
</tbody>
</table>
in | plane3 | [optional] The plane which will form channel 3. Must be FOURCC_U8.
out | output | The output image.

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>
### 3.15 Function: Channel Extract

#### 3.15.1 Detailed Description

The Channel Extraction Kernel. This kernel removes a single FOURCC_U8 channel (plane) from a multi-planar or interleaved image format from *vx_fourcc_e*.

### Functions

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

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

#### 3.15.2 Function Documentation

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

[Graph] Creates a channel extract 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. Must be one of the defined <em>vx_fourcc_e</em> multi-planar formats.</td>
</tr>
<tr>
<td>in</td>
<td>channel</td>
<td>The <em>vx_channel_e</em> channel to extract.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image. Must be <em>FOURCC_U8</em>.</td>
</tr>
</tbody>
</table>

**See Also**

*vx_kernel_channel_extract*

**Returns**

*vx_node*

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>Node could not be created.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Node Handle</td>
</tr>
</tbody>
</table>

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

[Immediate] Invokes an immediate Channel Extract.

**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. Must be one of the defined <em>vx_fourcc_e</em> multiplanar formats.</td>
</tr>
<tr>
<td>in</td>
<td>channel</td>
<td>The <em>vx_channel_e</em> enumeration to extract.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image. Must be <em>FOURCC_U8</em>.</td>
</tr>
</tbody>
</table>

**Returns**

* 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>
3.16 Function: Color Convert

3.16.1 Detailed Description

The Color Conversion Kernel. This kernel converts an image of a designated `vx_fourcc_e` format to another `vx_fourcc_e` format for those combinations listed in this table, where the columns are output types and rows are input types; the API version first supporting the conversion is listed:

<table>
<thead>
<tr>
<th>I/O</th>
<th>RGB</th>
<th>RGBX</th>
<th>NV12</th>
<th>NV21</th>
<th>UYVV</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>UYVV</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_fourcc_e` encoding, held in the `VX_IMAGE_ATTRIBUTE_FORMAT` attribute, describes the data layout. The interpretation of the colors is determined by the `VX_IMAGE_ATTRIBUTE_SPACE` (see `vx_color_space_e`) and `VX_IMAGE_ATTRIBUTE_RANGE` (see `vx_channel_range_e`) attributes of the image. OpenVX 1.0 implementations are only required to support images of `VX_COLOR_SPACE_BT709` and `VX_CHANNEL_RANGE_RESTRICTED`.

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:

\[
\text{value}_\text{real} = \frac{\text{value}_\text{integer}}{256.0}
\]

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

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

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

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

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:

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

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

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

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

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

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

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

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

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

\[
Y' = (K_r \times R') + (K_b \times B') + (1 - K_r - K_b)G'
\]

\[
Pb = \frac{B'}{2} - \frac{(R' \times K_r) + G'(1 - K_r - K_b)}{2(1 - K_b)}
\]

\[
Pr = \frac{R'}{2} - \frac{(B' \times K_b) + G'(1 - K_r - K_b)}{2(1 - K_r)}
\]
The means of reconstructing Pb and Pr values from chroma-downsampled formats is implementation-defined. In \texttt{VX\_COLOR\_SPACE\_BT601\_525} or \texttt{VX\_COLOR\_SPACE\_BT601\_625}:

\[
K_r = 0.299 \\
K_b = 0.114
\]

In \texttt{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 \( B' \), \( Y' \), \( Pb \) and \( Pr \).

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

\[
\text{value}_{\text{nonlinear}} = 1.099 \times \text{value}_{\text{linear}}^{0.45} - 0.099 \quad \text{for} \quad 1 \geq \text{value}_{\text{linear}} \geq 0.018
\]

\[
\text{value}_{\text{nonlinear}} = 4.500 \times \text{value}_{\text{linear}} \quad \text{for} \quad 0.018 > \text{value}_{\text{linear}} \geq 0
\]

\[
\text{value}_{\text{linear}} = \left( \frac{\text{value}_{\text{nonlinear}} + 0.099}{1.099} \right)^{\frac{1}{0.45}} \quad \text{for} \quad 1 \geq \text{value}_{\text{nonlinear}} > 0.081
\]

\[
\text{value}_{\text{linear}} = \frac{\text{value}_{\text{nonlinear}}}{4.5} \quad \text{for} \quad 0.081 \geq \text{value}_{\text{nonlinear}} \geq 0
\]

Since 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:

\[
R_{BT601.525} = R_{BT601.625} \times 1.112302 + G_{BT601.625} \times -0.102441 + B_{BT601.625} \times -0.009860
\]

\[
G_{BT601.525} = R_{BT601.625} \times -0.020497 + G_{BT601.625} \times 1.037030 + B_{BT601.625} \times -0.016533
\]

\[
R_{BT601.525} = R_{BT601.625} \times 0.001704 + G_{BT601.625} \times 0.016063 + B_{BT601.625} \times 0.982233
\]

\[
R_{BT709.525} = R_{BT709} \times 1.065379 + G_{BT709} \times -0.055401 + B_{BT709} \times -0.009978
\]

\[
G_{BT709.525} = R_{BT709} \times -0.019633 + G_{BT709} \times 1.036363 + B_{BT709} \times -0.016731
\]

\[
R_{BT709.525} = R_{BT709} \times 0.001632 + G_{BT709} \times 0.004412 + B_{BT709} \times 0.993956
\]

\[
R_{BT601.625} = R_{BT601.525} \times 0.900657 + G_{BT601.525} \times 0.088807 + B_{BT601.525} \times 0.010536
\]

\[
G_{BT601.625} = R_{BT601.525} \times 0.017772 + G_{BT601.525} \times 0.965793 + B_{BT601.525} \times 0.016435
\]

\[
R_{BT601.625} = R_{BT601.525} \times -0.001853 + G_{BT601.525} \times -0.015948 + B_{BT601.525} \times 1.017801
\]

\[
R_{BT601.625} = R_{BT709} \times 0.957815 + G_{BT709} \times 0.042185
\]

\[
G_{BT601.625} = G_{BT709}
\]

\[
B_{BT601.625} = G_{BT709} \times -0.011934 + B_{BT709} \times 1.011934
\]

\[
R_{BT709} = R_{BT601.525} \times 0.939542 + G_{BT601.525} \times 0.050181 + B_{BT601.525} \times 0.010277
\]

\[
G_{BT709} = R_{BT601.525} \times 0.017772 + G_{BT601.525} \times 0.965793 + B_{BT601.525} \times 0.016435
\]

\[
R_{BT709} = R_{BT601.525} \times -0.001622 + G_{BT601.525} \times -0.004370 + B_{BT601.525} \times 1.005991
\]

\[
R_{BT709} = R_{BT601.525} \times 1.044043 + G_{BT601.525} \times -0.044043
\]

\[
G_{BT709} = G_{BT601.625}
\]

\[
B_{BT709} = G_{BT601.625} \times 0.011793 + B_{BT601.625} \times 0.988207
\]

A conversion between one YUV color space and another may therefore consist of the following transformations:
1. Convert quantized Y’CbCr ("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 Y’PbPr
7. Convert continuous Y’PbPr 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_API vx_node vxColorConvertNode (vx_graph graph, vx_image input, vx_image output)**
  [Graph] Creates a color conversion node.

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

### 3.16.2 Function Documentation

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

[Graph] Creates a color 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 to convert from.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image to convert into.</td>
</tr>
</tbody>
</table>

**See Also**

**VX_KERNEL_COLOR_CONVERT**

**Returns**

vx_node

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>Node could not be created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Node Handle</td>
</tr>
</tbody>
</table>

**VX_API vx_status vxuColorConvert ( vx_context context, vx_image input, vx_image output )**

[Immediate] Invokes an immediate Color 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 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>
3.17 Function: Convert Bit depth

3.17.1 Detailed Description

Converts image bit depth. This kernel converts an image from some source bit-depth to another bit-depth as described by the table below. The columns are output types and rows are input types and the conversion has the API version on which it is supported listed (An ‘X’ implies 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>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U16</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S16</td>
<td>1.0</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>U32</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>S32</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Down-conversions with VX_CONVERT_POLICY_TRUNCATE follow this equation:

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

Down-conversions with VX_CONVERT_POLICY_SATURATE follow this equation:

\[
\text{INTYPE value} = \text{input}(x,y) \gg \text{shift};
\]
\[
\text{value} = (\text{value} < \text{OUTTYPE_MIN} ? \text{OUTTYPE_MIN} : \text{value});
\]
\[
\text{value} = (\text{value} > \text{OUTTYPE_MAX} ? \text{OUTTYPE_MAX} : \text{value});
\]
\[
\text{output}(x,y) = (\text{OUTTYPE})\text{value};
\]

Up-conversions ignore the policy and perform this operation:

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

Functions

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

- VX_API vx_status 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.

3.17.2 Function Documentation

VX_API vx_node 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.</td>
</tr>
<tr>
<td>in</td>
<td>policy</td>
<td>A scalar containing a VX_TYPE_ENUM of the vx_convert_policy_t 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Node could not be created.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

Node Handle

VX_API vx_status 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_convert_policy_e enumeration.</td>
</tr>
<tr>
<td>in</td>
<td>shift</td>
<td>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>
3.18 Function: Custom Convolution

3.18.1 Detailed Description

Convolves the input with the client supplied convolution matrix. The client can supply a \( \text{vx\_int16} \) typed convolution matrix \( C_{m,n} \). Outputs will be in the \( \text{FOURCC\_S16} \) format unless a \( \text{FOURCC\_U8} \) image is explicitly provided. If values would have been out of range of U8 for \( \text{FOURCC\_U8} \), the values are clamped to 0 or 255.

\[
\begin{align*}
  k_0 &= \frac{m}{2} + 1 \\
  l_0 &= \frac{n}{2} + 1 \\
  \text{sum} &= \sum_{k=0,l=0}^{m,n} \text{input}(x+k_0-k, y+l_0-l)C_{k,l}
\end{align*}
\]  

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.
\text{vx\_int16} \text{gx}[3][3] = {
    \{ 3, 0, -3 \},
    \{ 10, 0, -10 \},
    \{ 3, 0, -3 \},
};
\text{vx\_uint32} \text{scale} = 9;
\text{vx\_convolution \text{scharr\_x} = \text{vx\_create\_convolution}}(\text{context}, 3, 3);
\text{vx\_access\_convolution\_coefficients}(<\text{scharr\_x}, \text{null}1);
\text{vx\_commit\_convolution\_coefficients}(<\text{scharr\_x}, \{ \text{vx\_int16 *1} \}\text{gx});
\text{vx\_set\_convolution\_attribute}(\text{scharr\_x},
    \text{vx\_CONVOLUTION\_ATTRIBUTE\_SCALE}, &\text{scale}, \text{sizeof(scale)});
```

For \( \text{FOURCC\_U8} \) output, an additional step is taken:

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

For \( \text{FOURCC\_S16} \) output, the summation is simply set to the output

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

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

Functions

- \text{VX\_API \text{vx\_node \text{vx\_convolve\_node}}(\text{vx\_graph} \text{graph}, \text{vx\_image} \text{input}, \text{vx\_convolution} \text{conv}, \text{vx\_image} \text{output})}
  
  [Graph] Creates a custom convolution node.

- \text{VX\_API \text{vx\_status \text{vx\_u\_convolve}}(\text{vx\_context} \text{context}, \text{vx\_image} \text{input}, \text{vx\_convolution} \text{matrix}, \text{vx\_image} \text{output})}

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

3.18.2 Function Documentation

\text{VX\_API \text{vx\_node \text{vx\_convolve\_node}}(\text{vx\_graph} \text{graph}, \text{vx\_image} \text{input}, \text{vx\_convolution} \text{conv}, \text{vx\_image} \text{output})}

[Graph] Creates a custom convolution node.
### CHAPTER 3. MODULE DOCUMENTATION

**Parameters**

<table>
<thead>
<tr>
<th>param</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>in</td>
<td><strong>conv</strong></td>
</tr>
<tr>
<td>out</td>
<td><strong>output</strong></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_API vx_status vxuConvolve ( vx_context context, vx_image input, vx_convolution matrix, vx_image output )**

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

**Parameters**

<table>
<thead>
<tr>
<th>param</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>context</strong></td>
</tr>
<tr>
<td>in</td>
<td><strong>input</strong></td>
</tr>
<tr>
<td>in</td>
<td><strong>matrix</strong></td>
</tr>
<tr>
<td>out</td>
<td><strong>output</strong></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>
3.19  Function: Dilate Image

3.19.1  Detailed Description

Dilation "grows" the white space in a FOURCC_U8 "bool" image. This kernel uses a 3x3 box around the output pixel used to determine value.

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

Functions

- VX_API vx_node vxDilate3x3Node (vx_graph graph, vx_image input, vx_image output)
  [Graph] Creates an Dilation Image Node.
- VX_API vx_status vxuDilate3x3 (vx_context context, vx_image input, vx_image output)
  [Immediate] Dilates an image by a 3x3 window.

3.19.2  Function Documentation

VX_API vx_node vxDilate3x3Node ( vx_graph graph, vx_image input, vx_image output )

[Graph] Creates an 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 FOURCC_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in FOURCC_U8 format.</td>
</tr>
</tbody>
</table>

Returns

vx_node

Return values

| 0 | Node could not be created. |
|   | * Node Handle |

VX_API vx_status 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 FOURCC_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in FOURCC_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>
3.20 Function: Equalize Histogram

3.20.1 Detailed Description
Equalizes the histogram of a grayscale image. This kernel modifies the values of a grayscale image so that it will automatically have a standardized brightness and contrast, using Histogram Equalization.

Functions
- VX_API vx_node vxEqualizeHistNode (vx_graph graph, vx_image input, vx_image output)
  [Graph] Creates a Histogram Equalization node.
- VX_API vx_status vxuEqualizeHist (vx_context context, vx_image input, vx_image output)
  [Immediate] Equalizes the Histogram of a grayscale image.

3.20.2 Function Documentation
VX_API vx_node vxEqualizeHistNode ( vx_graph graph, vx_image input, vx_image output )
[Graph] Creates a Histogram Equalization 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 grayscale input image in FOURCC_U8.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The grayscale output image of type FOURCC_U8 with equalized brightness and contrast.</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_API vx_status vxuEqualizeHist ( vx_context context, vx_image input, vx_image output )
[Immediate] Equalizes the Histogram of a grayscale 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 grayscale input image in FOURCC_U8</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The grayscale output image of type FOURCC_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>
3.21 Function: Erode Image

3.21.1 Detailed Description

Erosion "shrinks" the white space in a FOURCC_U8 "bool" image. This kernel uses a 3x3 box around the output pixel used to determine value.

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

Functions

- VX_API vx_node vxErode3x3Node (vx_graph graph, vx_image input, vx_image output)
  [Graph] Creates an Erosion Image Node.
- VX_API vx_status vxuErode3x3 (vx_context context, vx_image input, vx_image output)
  [Immediate] Erodes an image by a 3x3 window.

3.21.2 Function Documentation

**VX_API vx_node 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 FOURCC_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in FOURCC_U8 format.</td>
</tr>
</tbody>
</table>

Returns

vx_node

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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_API vx_status 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 FOURCC_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in FOURCC_U8 format.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>
3.22 Function: Fast Corners

3.22.1 Detailed Description

Computes the corners in an image using FAST algorithm. The FAST (features from accelerated segment test) algorithm based on the FAST9 algorithm described in [3] and with some updates from [4]. 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.

3.22.2 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:

\[
I = \text{input image} \quad (3.4)
\]

\[
p = \text{candidate point position for a corner} \quad (3.5)
\]

\[
I_p = \text{image intensity of the candidate point in image} I \quad (3.6)
\]

\[
x = \text{pixel on the Bresenham circle around the candidate point} p \quad (3.7)
\]

\[
I_x = \text{image intensity of the candidate point} \quad (3.8)
\]

\[
t = \text{intensity difference threshold for a corner} \quad (3.9)
\]

\[
N = \text{minimum number of contiguous pixel to detect a corner} \quad (3.10)
\]

\[
S = \text{set of contiguous pixel on the Bresenham circle around the candidate point} \quad (3.11)
\]

\[
C_p = \text{corner response at corner location} p \quad (3.12)
\]

\[
(3.13)
\]

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 \( \text{strength_threshold} \) of type VX_TYPE_FLOAT32 must be within:

\[
\text{UINT8_MIN} < t < \text{UINT8_MAX}
\]

These limits are established due to the input data type FOURCC_U8.

Corner Strength Computation Once a corner has been detected, its strength (response, saliency or score) is computed. 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 nonmax_suppression flag is true, a non-maxima suppression step is applied on the detected corners. This step is only keeping the corner which has a response greater than the ones of its neighboring corners within the Bresenham circle of radius 3.

See Also

http://www.edwardrosten.com/work/fast.html
http://en.wikipedia.org/wiki/Features_from_accelerated_segment_test
CHAPTER 3. MODULE DOCUMENTATION

Functions

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

  [Graph] Creates a FAST Corners Node.

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

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

### 3.22.3 Function Documentation

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

[Graph] Creates a FAST 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 FOURCC_U8 image.</td>
</tr>
<tr>
<td>in</td>
<td>strength_thresh</td>
<td>Threshold on difference between intensity of the central pixel and pixels on Bresenham's circle of radius 3 (VX_TYPE_FLOAT32 scalar)</td>
</tr>
<tr>
<td>in</td>
<td>nonmax_supression</td>
<td>If true, non-maximum suppression is applied to detected corners before being placed in the vx_array of VX_TYPE_KEYPOINT objects.</td>
</tr>
<tr>
<td>out</td>
<td>corners</td>
<td>Output corner vx_array of VX_TYPE_KEYPOINT.</td>
</tr>
<tr>
<td>out</td>
<td>num_corners</td>
<td>The total number of detected corners in image (optional).</td>
</tr>
</tbody>
</table>

#### Returns

- **vx_node**

  Return values

| 0   | Node could not be created. |
| 1   | Node Handle |

**VX_API vx_status vxuFastCorners (vx_context context, vx_image input, vx_scalar strength_thresh, vx_bool nonmax_supression, 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>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 FOURCC_U8 image.</td>
</tr>
<tr>
<td>in</td>
<td>strength_thresh</td>
<td>Threshold on difference between intensity of the central pixel and pixels on Bresenham's circle of radius 3 (VX_TYPE_FLOAT32 scalar)</td>
</tr>
<tr>
<td>in</td>
<td>nonmax_supression</td>
<td>If true, non-maximum suppression is applied to detected corners before being placed in the vx_array of VX_TYPE_KEYPOINT structs.</td>
</tr>
<tr>
<td>out</td>
<td>corners</td>
<td>Output corner vx_array of VX_TYPE_KEYPOINT.</td>
</tr>
<tr>
<td>out</td>
<td>num_corners</td>
<td>The total number of detected corners in image (optional).</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>
### 3.23 Function: Gaussian Filter

#### 3.23.1 Detailed Description

Computes a Gaussian filter over a window of the input image. This filter uses the follow convolution matrix:

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

#### Functions

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

- **VX_API vx_status vpxGaussian3x3 (vx_context context, vx_image input, vx_image output)**
  
  [Immediate] Computes a gaussian filter on the image by a 3x3 window.

#### 3.23.2 Function Documentation

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

[Graph] Creates a Gaussian Filter Node.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Type</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 FOURCC_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in FOURCC_U8 format.</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_API vx_status vpxGaussian3x3 ( 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>Type</th>
<th>Type</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 FOURCC_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in FOURCC_U8 format.</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 Function: Harris Corners

3.24.1 Detailed Description

Computes the Harris Corners of an image. The Harris Corners are computed with several parameters:

\[ I = \text{input image} \]  
\[ T_c = \text{corner strength threshold} \]  
\[ r = \text{Euclidean radius} \]  
\[ k = \text{sensitivity threshold} \]  
\[ w = \text{window size} \]  
\[ b = \text{block size} \]

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

\[ G_x = \text{Sobel}_x(w, I) \]  
\[ G_y = \text{Sobel}_y(w, I) \]  
\[ A = \text{window}_{G_{xx}}(x - b/2, y - b/2, x + b/2, y + b/2) \]  
\[ \text{trace}(A) = \sum A G_{xx}^2 + \sum A G_{yy}^2 \]  
\[ \text{det}(A) = \sum A G_{xx}^2 \sum A G_{yy}^2 - \left( \sum A (G_x G_y) \right)^2 \]  
\[ M_c(x, y) = \text{det}(A) - k \times \text{trace}(A)^2 \]  
\[ V_c(x, y) = \begin{cases} 
M_c(x, y) & \text{if } M_c(x, y) > T_c \\
0 & \text{otherwise}
\end{cases} \]

where \( V_c \) is the thresholded corner value. \( V_c \) is then non-maximally suppressed within the Euclidean distance \( r \) and returned as a \( \text{vx_array} \) of \( \text{vx_keypoint_t} \) structs. The Sobel kernels used for the gradient computation shall be as shown below:

- For gradient size 3:
  \[
  \text{sobel} = \begin{bmatrix}
  1 & 0 & -1 \\
  2 & 0 & -2 \\
  1 & 0 & 1 
  \end{bmatrix}
  \]

- For gradient size 5:
  \[
  \text{sobel} = \begin{bmatrix}
  1 & 2 & 0 & -2 & -1 \\
  4 & 8 & 0 & -8 & -4 \\
  6 & 12 & 0 & -12 & -6 \\
  4 & 8 & 0 & -8 & -4 \\
  1 & 2 & 0 & -2 & -1 
  \end{bmatrix}
  \]

- For gradient size 7:
  \[
  \text{sobel} = \begin{bmatrix}
  1 & 4 & 5 & 0 & -5 & -4 & -1 \\
  6 & 24 & 30 & 0 & -30 & -24 & -6 \\
  15 & 60 & 75 & 0 & -75 & -60 & -15 \\
  20 & 80 & 100 & 0 & -100 & -80 & -20 \\
  15 & 60 & 75 & 0 & -75 & -60 & -15 \\
  6 & 24 & 30 & 0 & -30 & -24 & -6 \\
  1 & 4 & 5 & 0 & -5 & -4 & -1 
  \end{bmatrix}
  \]

Functions

- \( \text{VX_API vx_node vxHarrisCornersNode (vx_graph graph, vx_image input, vx_scalar strength_thresh, vx_scalar min_distance, vx_scalar sensitivity, vx_int32 gradient_size, vx_int32 block_size, vx_array corners, vx_scalar num_corners)} \)
CHAPTER 3. MODULE DOCUMENTATION

[Graph] Creates a Harris Corners Node.

- **VX_API vx_status 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.

### 3.24.2 Function Documentation

**VX_API vx_node 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 )**

[Graph] Creates a Harris Corners 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 the graph.</td>
</tr>
<tr>
<td><strong>in</strong> input</td>
<td>The input FOURCC_U8 image.</td>
</tr>
<tr>
<td><strong>in</strong> strength_thresh</td>
<td>The minimum threshold which to eliminate Harris Corner scores.</td>
</tr>
<tr>
<td><strong>in</strong> min_distance</td>
<td>The radial Euclidean distance for non-maximum suppression.</td>
</tr>
<tr>
<td><strong>in</strong> sensitivity</td>
<td>The VX_TYPE_FLOAT32 scalar sensitivity threshold ( k ) from the Harris--Stephens equation.</td>
</tr>
<tr>
<td><strong>in</strong> 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><strong>in</strong> 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><strong>out</strong> corners</td>
<td>The array of VX_TYPE_KEYPOINT objects.</td>
</tr>
<tr>
<td><strong>out</strong> num_corners</td>
<td>The total number of detected corners in image (optional).</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_API vx_status 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

<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> input</td>
<td>The input FOURCC_U8 image.</td>
</tr>
<tr>
<td><strong>in</strong> strength_thresh</td>
<td>The minimum threshold which to eliminate Harris Corner scores.</td>
</tr>
<tr>
<td><strong>in</strong> min_distance</td>
<td>The radial Euclidean distance for non-maximum suppression.</td>
</tr>
<tr>
<td><strong>in</strong> sensitivity</td>
<td>The VX_TYPE_FLOAT32 scalar sensitivity threshold ( k ) from the Harris--Stephens equation.</td>
</tr>
<tr>
<td><strong>in</strong> 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><strong>in</strong> 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>
</tbody>
</table>
out | corners | The array of VX_TYPE_KEYPOINT structs.
---|---|---
out | num_corners | The total number of detected corners in image (optional).

Returns

A vx_status_e enumeration.

Return values

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

3.25.1 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.

Functions
- **VX_API vx_node vxHistogramNode (vx_graph graph, vx_image input, vx_distribution distribution)**
  [Graph] Creates a Histogram node.
- **VX_API vx_status vxuHistogram (vx_context context, vx_image input, vx_distribution distribution)**
  [Immediate] Generates a distribution from an image.

3.25.2 Function Documentation

**VX_API vx_node 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 FOURCC_U8.</td>
</tr>
<tr>
<td>out</td>
<td>distribution</td>
<td>The output distribution.</td>
</tr>
</tbody>
</table>

Returns

`vx_node`

Return values

| 0  | Node could not be created. |
| *  | Node Handle |

**VX_API vx_status 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 FOURCC_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>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>
3.26 Function: Gaussian Image Pyramid

3.26.1 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_TYPE_NEAREST_NEIGHBOR. Level 0 shall always have the same resolution as the input image. For the gaussian pyramid level 0 shall be the same as the input image. The pyramids must be configured with one of the following level scaling:

- VX_SCALE_PYRAMID_HALF
- VX_SCALE_PYRAMID_ORB

Functions

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

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

3.26.2 Function Documentation

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

[Graph] Creates a node for a Gaussian 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 input</td>
<td>The input image in FOURCC_U8 format.</td>
</tr>
<tr>
<td>out gaussian</td>
<td>The gaussian pyramid with FOURCC_U8 to construct.</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_API vx_status 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 FOURCC_U8</td>
</tr>
<tr>
<td>out</td>
<td>gaussian</td>
<td>The gaussian pyramid 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 <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
3.27 Function: Integral Image

3.27.1 Detailed Description

Computes the integral image of the input. Each output pixel is the sum of all pixels above and to the left of itself.

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

Functions

- **VX_API vx_node vxIntegralImageNode (vx_graph graph, vx_image input, vx_image output)**
  - [Graph] Creates an Integral Image Node.

- **VX_API vx_status vxuIntegralImage (vx_context context, vx_image input, vx_image output)**
  - [Immediate] Computes the integral image of the input.

3.27.2 Function Documentation

**VX_API vx_node vxIntegralImageNode (vx_graph graph, vx_image input, vx_image output)**

[Graph] Creates an Integral 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 FOURCC_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in FOURCC_U32 format.</td>
</tr>
</tbody>
</table>

Returns

- vx_node

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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_API vx_status vxuIntegralImage (vx_context context, vx_image input, vx_image output)**

[Immediate] Computes the integral image of the input.

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 FOURCC_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in FOURCC_U32 format.</td>
</tr>
</tbody>
</table>

Returns

- A vx_status_e enumeration.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>
3.28 Function: Magnitude

3.28.1 Detailed Description

The Gradient Magnitude Computation Kernel. This kernel takes two gradients in FOURCC_S16 format and computes the FOURCC_S16 normalized magnitude. Magnitude is computed as:

$$\text{mag}(x, y) = \sqrt{\text{grad}_x(x, y)^2 + \text{grad}_y(x, y)^2}$$

The conceptual definition describing the overflow is given as: \(\text{uint16 } z = \text{uint16}( \sqrt{\text{double}( \text{uint32}(\text{int32}(x) \times \text{int32}(x)) + \text{uint32}(\text{int32}(y) \times \text{int32}(y)))}) \); \text{int16 } r = z > 32767 ? 32767 : z;

Functions

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

- **VX_API vx_status vxuMagnitude (vx_context context, vx_image grad_x, vx_image grad_y, vx_image output)**
  
  [Immediate] Invokes an immediate Magnitude.

3.28.2 Function Documentation

**VX_API vx_node 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>in</th>
<th>graph</th>
<th>The reference to the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>grad_x</td>
<td>The input x image. This should be in FOURCC_S16 format.</td>
</tr>
<tr>
<td>in</td>
<td>grad_y</td>
<td>The input y image. This should be in FOURCC_S16 format.</td>
</tr>
<tr>
<td>out</td>
<td>mag</td>
<td>The magnitude image. This will be in FOURCC_U16 format.</td>
</tr>
</tbody>
</table>

See Also

**VX_KERNEL_MAGNITUDE**

Returns

- **vx_node**

Return values

| 0 | Node could not be created. |
| * | Node Handle |

**VX_API vx_status vxuMagnitude ( vx_context context, vx_image grad_x, vx_image grad_y, vx_image output )**

[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 should be in FOURCC_S16 format.</td>
</tr>
<tr>
<td>in</td>
<td>grad_y</td>
<td>The input y image. This should be in FOURCC_S16 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The magnitude image. This will be in FOURCC_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>
3.29 Function: Mean and Standard Deviation.

3.29.1 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

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

The standard deviation is computed as

\[
\sigma = \sqrt{\frac{\sum_{y=0}^{h} \sum_{x=0}^{w} (\mu - src(x, y))^2}{(width \times height)}}
\]

Functions

- VX_API vx_node vxMeanStdDevNode (vx_graph graph, vx_image input, vx_scalar mean, vx_scalar stddev)
  [Graph] Creates a mean value and standard deviation node.

- VX_API vx_status vxuMeanStdDev (vx_context context, vx_image input, vx_float32 *mean, vx_float32 *stddev)
  [Immediate] Computes the mean value and standard deviation.

3.29.2 Function Documentation

VX_API vx_node vxMeanStdDevNode (vx_graph graph, vx_image input, vx_scalar mean, vx_scalar stddev)

[Graph] Creates a mean value and standard deviation 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>mean</td>
<td>The VX_TYPE_FLOAT32 average pixel value.</td>
</tr>
<tr>
<td>out</td>
<td>stddev</td>
<td>The VX_TYPE_FLOAT32 standard deviation of the pixel values.</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_API vx_status vxuMeanStdDev (vx_context context, vx_image input, vx_float32 *mean, vx_float32 *stddev)

[Immediate] Computes the mean value and standard deviation.

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>mean</td>
<td>The average pixel value.</td>
</tr>
<tr>
<td>out</td>
<td>stddev</td>
<td>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 vx_status_e.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.30 Function: Median Filter

3.30.1 Detailed Description

Compute a median pixel value over a window of the input image. The median is the middle value over an odd numbered sorted range of values.

Functions

- VX_API vx_node vxMedian3x3Node (vx_graph graph, vx_image input, vx_image output)
  [Graph] Creates a Median Image Node.

- VX_API vx_status vxuMedian3x3 (vx_context context, vx_image input, vx_image output)
  [Immediate] Computes a median filter on the image by a 3x3 window.

3.30.2 Function Documentation

VX_API vx_node vxMedian3x3Node ( vx_graph graph, vx_image input, vx_image output )

[Graph] Creates a Median 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 FOURCC_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in FOURCC_U8 format.</td>
</tr>
</tbody>
</table>

Returns

vx_node

Return values

<table>
<thead>
<tr>
<th>vx_node</th>
<th>Node could not be created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Node Handle</td>
</tr>
</tbody>
</table>

VX_API vx_status 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>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 FOURCC_U8 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output image in FOURCC_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>
3.31 Function: Min, Max Location

3.31.1 Detailed Description

Finds the minimum and maximum values in an image and a location for each. If the input image has several
minimums/maximums, the kernel will return all of them.

\[
\begin{align*}
\text{minVal} &= \min_{0 \leq x' \leq \text{width}} \min_{0 \leq y' \leq \text{height}} \text{src}(x', y') \\
\text{maxVal} &= \max_{0 \leq x' \leq \text{width}} \max_{0 \leq y' \leq \text{height}} \text{src}(x', y')
\end{align*}
\]

Functions

- **VX_API vx_node 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_API vx_status 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.

3.31.2 Function Documentation

**VX_API vx_node 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>in</th>
<th>graph</th>
<th>The reference to create the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>input</td>
<td>The input image in FOURCC_U8 or FOURCC_S16 format.</td>
</tr>
<tr>
<td>out</td>
<td>minVal</td>
<td>The minimum value in the image, which corresponds to the type of the input.</td>
</tr>
<tr>
<td>out</td>
<td>maxVal</td>
<td>The maximum value in the image, which corresponds to the type of the input.</td>
</tr>
<tr>
<td>out</td>
<td>minLoc</td>
<td>The minimum VX_TYPE_COORDINATES2D locations (optional, if the input image has several minimums, the kernel will return up to the capacity of the array).</td>
</tr>
<tr>
<td>out</td>
<td>maxLoc</td>
<td>The maximum VX_TYPE_COORDINATES2D locations (optional, if the input image has several maximums, the kernel will return up to the capacity of the array).</td>
</tr>
<tr>
<td>out</td>
<td>minCount</td>
<td>The total number of detected minimums in image (optional). Use a VX_TYPE_E_UINT32 scalar.</td>
</tr>
<tr>
<td>out</td>
<td>maxCount</td>
<td>The total number of detected maximums in image (optional). Use a VX_TYPE_E_UINT32 scalar.</td>
</tr>
</tbody>
</table>

**Returns**

vx_node

**Return values**

| 0 | Node could not be created. |
| * | Node Handle |

**VX_API vx_status 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>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 FOURCC_U8 or FOURCC_S16 format.</td>
</tr>
<tr>
<td>out</td>
<td>minVal</td>
<td>The minimum value in the image.</td>
</tr>
<tr>
<td>out</td>
<td>maxVal</td>
<td>The maximum value in the image.</td>
</tr>
<tr>
<td>out</td>
<td>minLoc</td>
<td>The minimum locations (optional, if the input image has several minimums, the kernel will return all of them).</td>
</tr>
<tr>
<td>out</td>
<td>maxLoc</td>
<td>The maximum locations (optional, if the input image has several maximums, the kernel will return all of them).</td>
</tr>
<tr>
<td>out</td>
<td>minCount</td>
<td>The total number of detected minimums in image (optional).</td>
</tr>
<tr>
<td>out</td>
<td>maxCount</td>
<td>The total number of detected maximums in image (optional).</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>
3.32  Function: Optical Flow Pyramid (LK)

3.32.1  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 [1]. 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. 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 older 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:

\[
\begin{bmatrix}
V_x \\
V_y
\end{bmatrix} = \begin{bmatrix}
\sum_i I_x^2 & \sum_i I_x I_y \\
\sum_i I_x I_y & \sum_i I_y^2
\end{bmatrix}^{-1} \begin{bmatrix}
-\sum_i I_x I_t \\
-\sum_i I_y I_t
\end{bmatrix}
\]

Where \( I_x \) and \( I_y \) are obtained using the Scharr gradients on the input image:

\[
\begin{align*}
G_x &= \begin{bmatrix}
+3 & 0 & -3 \\
+10 & 0 & -10 \\
+3 & 0 & -3
\end{bmatrix} \\
G_y &= \begin{bmatrix}
+3 & +10 & +3 \\
0 & 0 & 0 \\
-3 & -10 & -3
\end{bmatrix}
\]

\( I_t \) 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 check if the pixel to track was lost. The criteria for lost tracking is that the matrix above is invertible. (The determinant of the matrix is less then a threshold : \( 10^{-7} \)). Or the minimum eigenvalue of the matrix is smaller then a threshold (\( 10^{-4} \)). Also lost tracking happen 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 from the old tracking vx_keypoint_t to new image vx_keypoint_t.

Functions:

- VX_API vx_node 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.

- VX_API vx_status 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.
3.32.2 Function Documentation

VX_API vx_node 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>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> old_images</td>
<td>Input of first (old) image pyramid in <strong>FOURCC_U8</strong></td>
</tr>
<tr>
<td><strong>in</strong> new_images</td>
<td>Input of destination (new) image pyramid <strong>FOURCC_U8</strong></td>
</tr>
<tr>
<td><strong>in</strong> old_points</td>
<td>an array of key points in a <strong>vx_array</strong> of VX_TYPE_KEYPOINT those key points are defined at the old_images high resolution pyramid</td>
</tr>
<tr>
<td><strong>in</strong> new_points_estimates</td>
<td>an array of estimation on what is the output key points in a <strong>vx_array</strong> of VX_TYPE_KEYPOINT those keypoints are defined at the new_images high resolution pyramid</td>
</tr>
<tr>
<td><strong>out</strong> new_points</td>
<td>a output array of key points in a <strong>vx_array</strong> of VX_TYPE_KEYPOINT those key points are defined at the new_images high resolution pyramid</td>
</tr>
<tr>
<td><strong>in</strong> termination</td>
<td>termination can be VX_TERM_CRITERIA_ITERATIONS or VX_TERM_CRITERIA_EPSILON or VX_TERM_CRITERIA_BOTH</td>
</tr>
<tr>
<td><strong>in</strong> epsilon</td>
<td>is the <strong>vx_float32</strong> error for terminating the algorithm</td>
</tr>
<tr>
<td><strong>in</strong> num_iterations</td>
<td>This is the number of iterations. Use a VX_TYPE_UINT32 scalar.</td>
</tr>
<tr>
<td><strong>in</strong> use_initial_estimate</td>
<td>Use a VX_TYPE_BOOL scalar.</td>
</tr>
<tr>
<td><strong>in</strong> window_dimension</td>
<td>is the window on which to perform the algorithm.</td>
</tr>
</tbody>
</table>

### Returns

<table>
<thead>
<tr>
<th>Return</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_API vx_statusvxuOpticalFlowPyrLK ( 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>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> old_images</td>
<td>Input of first (old) image pyramid</td>
</tr>
<tr>
<td><strong>in</strong> new_images</td>
<td>Input of destination (new) image pyramid</td>
</tr>
<tr>
<td><strong>in</strong> old_points</td>
<td>an array of key points in a <strong>vx_array</strong> of VX_TYPE_KEYPOINT those key points are defined at the old_images high resolution pyramid</td>
</tr>
<tr>
<td><strong>in</strong> new_points_estimates</td>
<td>an array of estimation on what is the output key points in a <strong>vx_array</strong> of VX_TYPE_KEYPOINT those keypoints are defined at the new_images high resolution pyramid</td>
</tr>
<tr>
<td><strong>out</strong> new_points</td>
<td>an output array of key points in a <strong>vx_array</strong> of VX_TYPE_KEYPOINT those key points are defined at the new_images high resolution pyramid</td>
</tr>
<tr>
<td><strong>in</strong> termination</td>
<td>termination can be VX_TERM_CRITERIA_ITERATIONS or VX_TERM_CRITERIA_EPSILON or VX_TERM_CRITERIA_BOTH</td>
</tr>
<tr>
<td><strong>in</strong> epsilon</td>
<td>is the <strong>vx_float32</strong> error for terminating the algorithm</td>
</tr>
<tr>
<td><strong>in</strong> num_iterations</td>
<td>is the number of iterations</td>
</tr>
<tr>
<td><strong>in</strong> use_initial_estimate</td>
<td>Can be set to either vx_false_e or vx_true_e.</td>
</tr>
</tbody>
</table>
### in window dimension

is the window on which to perform the algorithm.

**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>
3.33 Function: Phase

3.33.1 Detailed Description

The Gradient Phase Computation Kernel. This kernel takes two gradients in FOURCC_S16 format and computes the angles for each pixel and store this in a FOURCC_U8 image.

$$\phi = \tan^{-1} \frac{\text{grad}_y(x, y)}{\text{grad}_x(x, y)}$$

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_API vx_node vxPhaseNode (vx_graph graph, vx_image grad_x, vx_image grad_y, vx_image orientation)**
  [Graph] Create a Magnitude node.

- **VX_API vx_status vxuPhase (vx_context context, vx_image grad_x, vx_image grad_y, vx_image output)**
  [Immediate] Invokes an immediate Phase.

3.33.2 Function Documentation

**VX_API vx_node vxPhaseNode (vx_graph graph, vx_image grad_x, vx_image grad_y, vx_image orientation)**

[Graph] Create a Magnitude 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>grad_x</td>
<td>The input x image. This should be in FOURCC_S16 format.</td>
</tr>
<tr>
<td>in</td>
<td>grad_y</td>
<td>The input y image. This should be in FOURCC_S16 format.</td>
</tr>
<tr>
<td>out</td>
<td>orientation</td>
<td>The phase image. This will be in FOURCC_U8 format.</td>
</tr>
</tbody>
</table>

See Also

**VX_KERNEL_PHASE**

Returns

vx_node

Return values

| 0   | Node could not be created. |
| *   | Node Handle |

**VX_API vx_status vxuPhase (vx_context context, vx_image grad_x, vx_image grad_y, vx_image output)**

[Immediate] Invokes an immediate Phase.

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 should be in FOURCC_S16 format.</td>
</tr>
<tr>
<td>in</td>
<td>grad_y</td>
<td>The input y image. This should be in FOURCC_S16 format.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The phase image. This will be in FOURCC_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 <code>vx_status_e</code>.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

3.34 Function: Pixel-wise Multiplication

3.34.1 Detailed Description

Performs element-wise multiplication between two images and a scalar value. Pixel-wise multiplication is performed between the pixel values in two FOURCC_U8 or FOURCC_S16 images and a scalar floating-point number “scale”. The output image can be FOURCC_U8 only if both source images are FOURCC_U8 and the output image is explicitly set to FOURCC_U8. It is otherwise FOURCC_S16. If one of the input images is of type FOURCC_S16, all values are converted to FOURCC_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:

$$reference(x,y,\text{scale}) = \text{truncate}(((\text{int}32_t)\text{in}1(x,y)) * ((\text{int}32_t)\text{in}2(x,y)) * \text{(double)}\text{scale})$$

The rounding policy VX_ROUND_POLICY_TO_NEAREST_EVEN for this function is defined as:

$$reference(x,y,\text{scale}) = \text{round}_0\text{to}_\text{nearest}_\text{even}(((\text{int}32_t)\text{in}1(x,y)) * ((\text{int}32_t)\text{in}2(x,y)) * \text{(double)}\text{scale})$$

The overflow handling is controlled by an overflow-policy parameter. For each pixel value in the two input images:

$$\text{out}(x,y) = \text{in}_1(x,y)\text{in}_2(x,y)\text{scale}$$

Functions

- VX_API vx_node vxMultiplyNode (vx_graph graph, vx_image in1, vx_image in2, vx_scalar scale, vx_enum overflow_policy, vx_enum rounding_policy, vx_image out)
  
  [Graph] Creates an pixelwise-multiplication node.

- VX_API vx_status 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.

3.34.2 Function Documentation

VX_API vx_node vxMultiplyNode ( vx_graph graph, vx_image in1, vx_image in2, vx_scalar scale, vx_enum overflow_policy, vx_enum rounding_policy, vx_image out )

[Graph] Creates an pixelwise-multiplication 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, FOURCC_U8 or FOURCC_S16.</td>
</tr>
<tr>
<td>in</td>
<td>in2</td>
<td>An input image, FOURCC_U8 or FOURCC_S16.</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_TYPE_ENUM of the vx_convert_policy_e enumeration.</td>
</tr>
<tr>
<td>in</td>
<td>rounding_policy</td>
<td>A VX_TYPE_ENUM of the vx_round_policy_e enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>The output image, a FOURCC_U8 or FOURCC_S16 image.</td>
</tr>
</tbody>
</table>

Returns

vx_node
Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>

\texttt{VX\_API\ vx\_status\ vxuMultiply (\ vx\_context\ \textit{context},\ vx\_image\ \textit{in1},\ vx\_image\ \textit{in2},\ vx\_float32\ \textit{scale},\ vx\_enum\ \textit{overflow\_policy},\ vx\_enum\ \textit{rounding\_policy},\ vx\_image\ \textit{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>\textit{in1}</td>
<td>A \texttt{FOURCC_U8} or \texttt{FOURCC_S16} input image.</td>
</tr>
<tr>
<td>in</td>
<td>\textit{in2}</td>
<td>A \texttt{FOURCC_U8} or \texttt{FOURCC_S16} input image.</td>
</tr>
<tr>
<td>in</td>
<td>\textit{scale}</td>
<td>The scale value.</td>
</tr>
<tr>
<td>in</td>
<td>\textit{overflow_policy}</td>
<td>A \texttt{vx_convert_policy_e} enumeration.</td>
</tr>
<tr>
<td>in</td>
<td>\textit{rounding_policy}</td>
<td>A \texttt{vx_round_policy_e} enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>\textit{out}</td>
<td>The output image in \texttt{FOURCC_U8} or \texttt{FOURCC_S16} 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>
3.35  Function: Remap

3.35.1  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:

\[
output(x_1, y_1) = input(map_x(x_0, y_0), map_y(x_0, y_0))
\]

However, the mapping functions are contained in the vx_remap object.

Functions

- **VX_API vx_node vxRemapNode** (vx_graph graph, vx_image input, vx_remap table, vx_enum policy, vx_image output)
  
  [Graph] Creates a Remap Node.

- **VX_API vx_status 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.

3.35.2  Function Documentation

**VX_API vx_node 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>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>graph</td>
<td>The reference to the graph which will contain the node.</td>
</tr>
<tr>
<td>in</td>
<td>input</td>
<td>The input FOURCC_U8 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 vx_interpolation_type_e. VX_INTERPOLATION_TYPE_AREA is not supported.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output FOURCC_U8 image.</td>
</tr>
</tbody>
</table>

Note

Only VX_NODE_ATTRIBUTE_BORDER value VX_BORDER_MODE_UNDEFINED or VX_BORDER_MODE_CONSTANT is supported.

Returns

Returns a 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_API vx_status 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
<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 FOURCC_U8 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>The interpolation policy from vx_interpolation_type_e. VX_INTERPOLATION_AREA is not supported.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output FOURCC_U8 image.</td>
</tr>
</tbody>
</table>

Returns

Returns A vx_status_e enumeration.
3.36  Function: Scale Image

3.36.1 Detailed Description

The Image Resizing Kernel. This kernel resizes an image from the source to the destination dimensions. The only format supported is FOURCC_U8. The supported interpolation types are currently:

- VX_INTERPOLATION_TYPE_NEAREST_NEIGHBOR
- VX_INTERPOLATION_TYPE_AREA
- VX_INTERPOLATION_TYPE_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 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.

Functions

- VX_API vx_node vxScaleImageNode (vx_graph graph, vx_image src, vx_image dst, vx_enum type)  
  [Graph] Create a Scale Image Node.

- VX_API vx_status vxuHalfScaleGaussian3x3 (vx_context context, vx_image input, vx_image output)  
  [Immediate] Performs a Gaussian Blur (3x3) on an image then half-scales it.

- VX_API vx_status vxuScaleImage (vx_context context, vx_image src, vx_image dst, vx_enum type)  
  [Immediate] Scales an input image to an output image.

3.36.2 Function Documentation

VX_API vx_node vxScaleImageNode ( vx_graph graph, vx_image src, vx_image dst, vx_enum type )

[Graph] Create a Scale 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>src</td>
<td>The source image.</td>
</tr>
<tr>
<td>out</td>
<td>dst</td>
<td>The destination image.</td>
</tr>
<tr>
<td>in</td>
<td>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.

Returns

vx_node

Return values

| 0   | Node could not be created. |
| *   | Node Handle                |

VX_API vx_status vxuHalfScaleGaussian3x3 ( vx_context context, vx_image input, vx_image output )

[Immediate] Performs a Gaussian Blur (3x3) on an image then half-scales it.

The output image size is determined by:

\[
W_{\text{output}} = \frac{W_{\text{input}} + 1}{2} \\
H_{\text{output}} = \frac{H_{\text{input}} + 1}{2}
\]
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 FOURCC_U8 image.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output FOURCC_U8 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>

VX_API vx_status vxuScaleImage ( vx_context context, vx_image src, vx_image dst, vx_enum type )

[Immediate] Scales an input image to an 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>src</td>
<td>The source image.</td>
</tr>
<tr>
<td>out</td>
<td>dst</td>
<td>The destination image.</td>
</tr>
<tr>
<td>in</td>
<td>type</td>
<td>The interpolation type.</td>
</tr>
</tbody>
</table>

See Also

vx_interpolation_type_e.

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 Function: Sobel 3x3

#### 3.37.1 Detailed Description

The Sobel Image Filter Kernel. 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:

\[
G_x = \begin{bmatrix}
-1 & 0 & +1 \\
-2 & 0 & +2 \\
-1 & 0 & +1 \\
\end{bmatrix}, \quad G_y = \begin{bmatrix}
-1 & -2 & -1 \\
0 & 0 & 0 \\
+1 & +2 & +1 \\
\end{bmatrix}
\]

**Functions**

- **VX_API vx_node vxSobel3x3Node (vx_graph graph, vx_image input, vx_image output_x, vx_image output_y)**
  
  [Graph] Create a Sobel3x3 node.

- **VX_API vx_status vxuSobel3x3 (vx_context context, vx_image input, vx_image output_x, vx_image output_y)**
  
  [Immediate] Invokes an immediate Sobel 3x3.

#### 3.37.2 Function Documentation

**VX_API vx_node vxSobel3x3Node ( vx_graph graph, vx_image input, vx_image output_x, vx_image output_y )**

[Graph] Create a Sobel3x3 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>out</td>
<td>output_x</td>
</tr>
<tr>
<td>out</td>
<td>output_y</td>
</tr>
</tbody>
</table>

**See Also**

**VX\_KERNEL\_SOBEL\_3x3**

**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_API vx_status 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>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>out</td>
<td>output_x</td>
</tr>
<tr>
<td>out</td>
<td>output_y</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>
3.38 Function: TableLookup

3.38.1 Detailed Description

The Table Lookup Image Kernel. This kernel uses each pixel in an image to index into a LUT and put the indexed LUT value into the output image. The format supported is FOURCC_U8.

Functions

- **VX_API vx_node vxTableLookupNode (vx_graph graph, vx_image input, vx_lut lut, vx_image output)**
  [Graph] Creates a Table Lookup node.

- **VX_API vx_status vxuTableLookup (vx_context context, vx_image input, vx_lut lut, vx_image output)**
  [Immediate] Processes the image through the LUT.

3.38.2 Function Documentation

**VX_API vx_node vxTableLookupNode ( vx_graph graph, vx_image input, vx_lut lut, vx_image output )**

[Graph] Creates a Table Lookup node.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in graph</td>
<td>vx_graph</td>
<td>The reference to the graph.</td>
</tr>
<tr>
<td>in input</td>
<td>vx_image</td>
<td>The input image in FOURCC_U8.</td>
</tr>
<tr>
<td>in lut</td>
<td>vx_lut</td>
<td>The LUT which is of type VX_TYPE_UINT8.</td>
</tr>
<tr>
<td>out output</td>
<td>vx_image</td>
<td>The output image of type FOURCC_U8.</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_API vx_status 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>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in context</td>
<td>vx_context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>in input</td>
<td>vx_image</td>
<td>The input image in FOURCC_U8.</td>
</tr>
<tr>
<td>in lut</td>
<td>vx_lut</td>
<td>The LUT which is of type VX_TYPE_UINT8, or VX_TYPE_UINT16.</td>
</tr>
<tr>
<td>out output</td>
<td>vx_image</td>
<td>The output image of type FOURCC_U8.</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>
3.39 Function: Thresholding

3.39.1 Detailed Description

Thresholds an input image and produces an output boolean image. In VX_THRESHOLD_TYPE_BINARY, the output is determined by

$$\text{dst}(x,y) = \begin{cases} 255 & \text{if } \text{src}(x,y) > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

In VX_THRESHOLD_TYPE_RANGE, the output is determined by:

$$\text{dst}(x,y) = \begin{cases} 0 & \text{if } \text{src}(x,y) > \text{upper} \\ 0 & \text{if } \text{src}(x,y) < \text{lower} \\ 255 & \text{otherwise} \end{cases}$$

Functions

- **VX_API vx_node vxThresholdNode (vx_graph graph, vx_image input, vx_threshold thresh, vx_image output)**
  - [Graph] Creates a Threshold node.

- **VX_API vx_status vxuThreshold (vx_context context, vx_image input, vx_threshold thresh, vx_image output)**
  - [Immediate] Threshold’s an input image and produces a FOURCC_U8 * boolean image.

3.39.2 Function Documentation

**VX_API vx_node vxThresholdNode ( vx_graph graph, vx_image input, vx_threshold thresh, vx_image output )**

[Graph] Creates a Threshold 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. FOURCC_U8 is supported.</td>
</tr>
<tr>
<td>in</td>
<td>thresh</td>
<td>The thresholding object which defines the parameters of the operation.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output boolean image. Values are either 0 or 255.</td>
</tr>
</tbody>
</table>

Returns

vx_node

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>Node could not be created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Node Handle</td>
</tr>
</tbody>
</table>

**VX_API vx_status vxuThreshold ( vx_context context, vx_image input, vx_threshold thresh, vx_image output )**

[Immediate] Threshold’s an input image and produces a FOURCC_U8 * boolean 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. FOURCC_U8 is supported.</td>
</tr>
<tr>
<td>in</td>
<td>thresh</td>
<td>The thresholding object which defines the parameters of the operation.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output boolean image. Values are 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>
### 3.40 Function: Warp Affine

#### 3.40.1 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);
vxAccessMatrix(matrix, NULL);
vxCommitMatrix(matrix, mat);
```

#### Functions

- **VX_API vx_status 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_API vx_node vxWarpAffineNode** (vx_graph graph, vx_image input, vx_matrix matrix, vx_enum type, vx_image output)
  
  [Graph] Creates a Affine Warp Node.

#### 3.40.2 Function Documentation

**VX_API vx_status 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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td>context</td>
</tr>
<tr>
<td><code>in</code></td>
<td>input</td>
</tr>
<tr>
<td><code>in</code></td>
<td>matrix</td>
</tr>
<tr>
<td><code>in</code></td>
<td>type</td>
</tr>
<tr>
<td><code>out</code></td>
<td>output</td>
</tr>
</tbody>
</table>

**Returns**

A `vx_status_e` enumeration.

**Return values**

- **VX_SUCCESS** | Success |
  - An error occurred. See `vx_status_e`. |

**VX_API vx_node vxWarpAffineNode** (vx_graph graph, vx_image input, vx_matrix matrix, vx_enum type, vx_image output)

[Graph] Creates a Affine 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 FOURCC_U8 image.</td>
</tr>
<tr>
<td>in</td>
<td>matrix</td>
<td>The affine matrix. Must be 2x3 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_TYPE_AREA is not supported.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output FOURCC_U8 image.</td>
</tr>
</tbody>
</table>

Returns

vx_node

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>
3.41 Function: Warp Perspective

3.41.1 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:

$$
x_0 = M_{1,1} x + M_{1,2} y + M_{1,3}
$$

$$
y_0 = M_{2,1} x + M_{2,2} y + M_{2,3}
$$

$$
z_0 = M_{3,1} x + M_{3,2} y + M_{3,3}
$$

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);

vxCommitMatrix(matrix, mat);
```

Functions

- **VX_API vx_status 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_API vx_node vxWarpPerspectiveNode (vx_graph graph, vx_image input, vx_matrix matrix, vx_enum type, vx_image output)**
  [Graph] Creates a Perspective Warp Node.

3.41.2 Function Documentation

**VX_API vx_status 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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</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 FOURCC_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_TYPE_AREA is not supported.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output FOURCC_U8 image.</td>
</tr>
</tbody>
</table>

**Returns**

- A **vx_status_e** enumeration.

**Return values**

| VX_SUCCESS | Success |
An error occurred. See `vx_status_e`.

\[
\text{VX\_API `vx\_node` `vxWarpPerspectiveNode` ( `vx\_graph` `graph`, `vx\_image` `input`, `vx\_matrix` `matrix`, `vx\_enum` `type`, `vx\_image` `output`)}
\]

[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 FOURCC_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 <code>vx\_interpolation\_type\_e</code>. VX_INTERPOLATION_TYPE_AREA is not supported.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>The output FOURCC_U8 image.</td>
</tr>
</tbody>
</table>

**Returns**

`vx\_node`

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>Node could not be created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Node Handle</td>
</tr>
</tbody>
</table>
3.42 Basic Features

3.42.1 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

- **Basic Framework**
  The framework concepts and interfaces of OpenVX.
- **Objects**
  The basic objects within OpenVX.

Data Structures

- **struct vx_coordinates2d_t**
  The 2D Coordinates structure. More...
- **struct vx_coordinates3d_t**
  The 3D Coordinates structure. More...
- **struct vx_keypoint_t**
  The keypoint data structure. More...
- **struct vx_rectangle_t**
  The rectangle data structure which is shared with the users. More...

Macros

- **#define VX_API**
  This is a tag used to identify exported, public API functions as distinct from internal functions, helpers, and other non-public interfaces.
- **#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 (0x00000FF)**
  An object’s attribute ID is within the range of [0, 2^8 − 1] (inclusive).
- **#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"**
- **#define VX_FMT_SIZE "%zu"**
- **#define VX_FOURCC(a, b, c, d) (((a) | (b << 8) | (c << 16) | (d << 24))**
  Converts a set of four chars into a uint32_t container of a FOURCC code.
- **#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)
  The maximum length of a message buffer to copy from the log.
- \#define VX_SCALE_UNITY (1024u)
- \#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 in 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 ID’s are 2 nibbles in size and are located in the upper byte of the 4 bytes of an enumeration.
- \#define VX_VERSION VX_VERSION (10)
- \#define VX_VERSION_1_0 (VX_VERSION_MAJOR(1) | VX_VERSION_MINOR(0))
  The predefined version number for 1.0.
- \#define VX_VERSION_MAJOR(x) ((x & 0xFF) << 8)
- \#define VX_VERSION_MINOR(x) ((x & 0xFF) << 0)
  Typedefs

- typedef char vx_char
  An 8 bit ASCII character.
- 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 uint32_t vx_fourcc
  Used to hold a FOURCC code to describe the pixel format and color space.
- typedef int16_t vx_int16
  A 16-bit signed value.
- typedef int32_t vx_int32
  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 {**
  
  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_CHANNEL_0,
  VX_CHANNEL_G = VX_CHANNEL_1,
  VX_CHANNEL_B = VX_CHANNEL_2,
  VX_CHANNEL_A = VX_CHANNEL_3,
  VX_CHANNEL_Y = VX_CHANNEL_0,
  VX_CHANNEL_U = VX_CHANNEL_1,
  VX_CHANNEL_V = VX_CHANNEL_2 **}**
  
  The channel enumerations for channel extractions.

- **enum vx_convert_policy_e {**
  
  VX_CONVERT_POLICY_TRUNCATE = ((( 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.

- **enum vx_enum_e {**
  
  VX_ENUM_DIRECTION = 0x00,
  VX_ENUM_ACTION = 0x01,
  VX_ENUM_HINT = 0x02,
  VX_ENUM_DIRECTIVE = 0x03,
  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_MODE = 0x0C,
  VX_ENUM_COMPARISON = 0x0D,
  VX_ENUM_IMPORT_MEM = 0x0E,
  VX_ENUM_TERM_CRITERIA = 0x0F,
  VX_ENUM_NORM_TYPE = 0x10,
  VX_ENUM_ACCESSOR = 0x11,
  VX_ENUM_ROUND_POLICY = 0x12 **}**
  
  The set of supported enumerations in OpenVX.

- **enum vx_fourcc_e {**
FOURCC_VIRT = ((( 'V' ) | ( 'I' << 8 ) | ( 'R' << 16 ) | ( 'T' << 24 ))
FOURCC_RGB = ((( 'R' ) | ( 'G' << 8 ) | ( 'B' << 16 ) | ( '2' << 24 ))
FOURCC_NV12 = ((( 'N' ) | ( 'V' << 8 ) | ( '1' << 16 ) | ( '2' << 24 ))
FOURCC_NV21 = ((( 'N' ) | ( 'V' << 8 ) | ( '2' << 16 ) | ( '1' << 24 ))
FOURCC_UYVY = ((( 'U' ) | ( 'Y' << 8 ) | ( 'V' << 16 ) | ( 'Y' << 24 ))
FOURCC_YUYV = ((( 'Y' ) | ( 'U' << 8 ) | ( 'Y' << 16 ) | ( 'V' << 24 ))
FOURCC_IYUV = ((( 'I' ) | ( 'Y' << 8 ) | ( 'U' << 16 ) | ( 'V' << 24 ))
FOURCC_YUV4 = ((( 'Y' ) | ( 'U' << 8 ) | ( 'V' << 16 ) | ( '4' << 24 ))
FOURCC_U8 = ((( 'U' ) | ( '0' << 8 ) | ( '0' << 16 ) | ( '8' << 24 ))
FOURCC_U16 = ((( 'U' ) | ( '0' << 8 ) | ( '1' << 16 ) | ( '6' << 24 ))
FOURCC_S16 = ((( 'S' ) | ( '0' << 8 ) | ( '1' << 16 ) | ( '6' << 24 ))
FOURCC_U32 = ((( 'U' ) | ( '0' << 8 ) | ( '3' << 16 ) | ( '2' << 24 ))
FOURCC_S32 = ((( 'S' ) | ( '0' << 8 ) | ( '3' << 16 ) | ( '2' << 24 ))

Based on the FOURCC definition referenced on http://fourcc.org.

• enum vx_interpolation_type_e {
  VX_INTERPOLATION_TYPE_NEAREST_NEIGHBOR = ((( VX_ID_KHRONOS ) << 20 ) | ( VX_ENUM_INTERPOLATION << 12 )) + 0x0,
  VX_INTERPOLATION_TYPE_BILINEAR = ((( VX_ID_KHRONOS ) << 20 ) | ( VX_ENUM_INTERPOLATION << 12 )) + 0x1,
  VX_INTERPOLATION_TYPE_AREA = ((( VX_ID_KHRONOS ) << 20 ) | ( VX_ENUM_INTERPOLATION << 12 )) + 0x2
}

The image reconstruction filters supported by image resampling operations.

• 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_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_FOURCC = 0x00E,
  VX_TYPE_BOOL = 0x010,
  VX_TYPE_SCALAR_MAX,
  VX_TYPE_RECTANGLE = 0x020,
  VX_TYPE_KEYPOINT = 0x021,
  VX_TYPE_COORDINATES2D = 0x022,
  VX_TYPE_COORDINATES3D = 0x023,
  VX_TYPE_STRUCT_MAX,
  VX_TYPE_USER_STRUCT_START = 0x100,
  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_MAX
  }


CHAPTER 3. MODULE DOCUMENTATION

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_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 vxGetStatus (vx_reference reference)
  A generic API to return status values from Object constructors if they fail.

3.42.2 Data Structure Documentation

struct vx_coordinates2d_t
The 2D Coordinates structure.
Definition at line 1363 of file vx_types.h.
Data Fields

| vx_uint32 x | The X coordinate. |
| vx_uint32 y | The Y coordinate. |

struct vx_coordinates3d_t
The 3D Coordinates structure.
Definition at line 1371 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 1340 of file vx_types.h.
### Data Fields

| vx_float32 | error                  | An tracking method specific error. |
| vx_float32 | orientation           | Unused field reserved for future use. |
| vx_float32 | scale                 | Unused field reserved for future use. |
| vx_float32 | strength              | The strength of the keypoint.       |
| vx_int32   | tracking_status       | A zero indicates a lost point.      |
| vx_int32   | x                     | The x coordinate.                   |
| vx_int32   | y                     | The y coordinate.                   |

#### struct vx_rectangle_t

The rectangle data structure which is shared with the users.

Definition at line 1353 of file vx_types.h.

| vx_uint32 | end_x | The End X coordinate. |
| vx_uint32 | end_y | The End Y coordinate. |
| vx_uint32 | start_x | The Start X coordinate. |
| vx_uint32 | start_y | The Start Y coordinate. |

### 3.42.3 Macro Definition Documentation

#### #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 which are exceptions are vx_vendor_id_e, vx_type_e, vx_enum_e, vx_fourcc_e, and vx_bool.

#### #define VX_FMT_REF "%p"

Used to aid in debugging values in OpenVX.

Definition at line 1232 of file vx_types.h.

#### #define VX_FMT_SIZE "%zu"

Used to aid in debugging values in OpenVX.

Definition at line 1236 of file vx_types.h.

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

Converts a set of four chars into a uint32_t container of a FOURCC code.

Note

Use a vx_fourcc variable to hold the value.

#### #define VX_SCALE_UNITY (1024u)

Used to indicate the 1:1 ratio in Q22.10 format.

Definition at line 1241 of file vx_types.h.

#### #define VX_TYPE_MASK (0x000FFF00)

A type mask removes the scalar/object type from the attribute. It is 3 nibbles in size and in contained between the third and second byte.

See Also

vx_type_e

Definition at line 385 of file vx_types.h.
```c
#define VX_VERSION VX_VERSION_1_0
The OpenVX Version Number
   Definition at line 72 of file vx.h.

#define VX_VERSION_MAJOR( x ) ((x & 0xFF) << 8)
The major version number macro
   Definition at line 57 of file vx.h.

#define VX_VERSION_MINOR( x ) ((x & 0xFF) << 0)
The minor version number macro
   Definition at line 62 of file vx.h.

3.42.4 Typedef Documentation
typedef int32_t vx_enum
Sets the standard enumeration type size to be a fixed quantity.
   All enumerable fields should use this type as the container to enforce enumeration ranges and sizeof().
   Definition at line 119 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 357 of file vx_types.h.

3.42.5 Enumeration Type Documentation
enum vx_bool
A boolean value. This allows 0 to be false, as it is in C, and any non-zero to be true.

   vx_bool ret = vx_true_e;
   if (ret) printf("true\n");
   ret = vx_false_e;
   if (!ret) printf("false\n");

This would print both strings.

Enumerator
   vx_false_e The "false" value.
   vx_true_e The "true" value.
   Definition at line 250 of file vx_types.h.

enum vx_channel_e
The channel enumerations for channel extractions.
```
See Also
vxChannelExtractNode
vxuChannelExtract
VX_KERNEL_CHANNEL_EXTRACT

Enumerator

**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**  Used to extract the RED channel, no matter the byte or packing order.
**VX_CHANNEL_G**  Used to extract the GREEN channel, no matter the byte or packing order.
**VX_CHANNEL_B**  Used to extract the BLUE channel, no matter the byte or packing order.
**VX_CHANNEL_A**  Used to extract the ALPHA channel, no matter the byte or packing order.
**VX_CHANNEL_Y**  Used to extract the LUMA channel, no matter the byte or packing order.
**VX_CHANNEL_U**  Used to extract the Cb/U channel, no matter the byte or packing order.
**VX_CHANNEL_V**  Used to extract the Cr/V/Value channel, no matter the byte or packing order.

Definition at line 935 of file vx_types.h.

**enum vx_convert_policy_e**

The Conversion Policy Enumeration.

Enumerator

**VX_CONVERT_POLICY_TRUNCATE** 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 556 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_MODE**  Border Mode List.
**VX_ENUM_COMPARISON**  Comparison Values.
The memory import 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.

Definition at line 478 of file vx_types.h.

enum vx_fourcc_e

Based on the FOURCC definition referenced on http://fourcc.org.

Note

Use vx_fourcc to contain these values.

Enumerator

FOURCC_VIRT A virtual image of no defined type.

FOURCC_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.

FOURCC_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.

FOURCC_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.

FOURCC_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.

FOURCC_UVUV A single plane of 32 bit macro pixel of U0, Y0, V0, Y1 bytes. This uses the BT709 full range by default.

FOURCC_YUYV A single plane of 32 bit macro pixel of Y0, U0, Y1, V0 bytes. This uses the BT709 full range by default.

FOURCC_IYUV A 3 plane of 8 bit 4:2:0 sampled Y, U, V planes. This uses the BT709 full range by default.

FOURCC_YUV4 A 3 plane of 8 bit 4:4:4 sampled Y, U, V planes. This uses the BT709 full range by default.

FOURCC_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.

FOURCC_S16 A single place of signed 16 bit data. The range of data is not specified, as it may be extracted from a YUV or generated.

FOURCC_U16 A single place of unsigned 16 bit data. The range of data is not specified, as it may be extracted from a YUV or generated.

FOURCC_S32 A single place 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 569 of file vx_types.h.

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 which cross the boundary of the source image have values determined by the border mode - see vx_border_mode_e and VX_NODE_ATTRIBUTE_BORDER_MODE.
See Also

- vxuScaleImage
- vxScaleImageNode
- VX_KERNEL_SCALE_IMAGE
- vxuWarpAffine
- vxWarpAffineNode
- VX_KERNEL_WARP_AFFINE
- vxuWarpPerspective
- vxWarpPerspectiveNode
- VX_KERNEL_WARP_PERSPECTIVE

**Enumerator**

**VX_INTERPOLATION_TYPE_NEAREST_NEIGHBOR**  
Output values are defined to match the source pixel whose center is nearest to the sample position.

**VX_INTERPOLATION_TYPE_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_TYPE_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 995 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 could not complete due to a reference count being non-zero.

**VX_ERROR_MULTIPLE_WRITERS**  
Indicates that the graph had more than one node outputting to the same data object. This is an invalid graph structure.

**VX_ERROR_GRAPH_ABANDONED**  
Indicates that the graph was stopped due to an error or a callback which 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 can not be used in the current scope.

**VX_ERROR_INVALID_NODE**  
Indicates that the supplied node could not be created.

**VX_ERROR_INVALID_GRAPH**  
Indicates that the supplied graph had invalid connections (cycles)

**VX_ERROR_INVALID_TYPE**  
Indicates that the supplied type parameter was incorrect.

**VX_ERROR_INVALID_VALUE**  
Indicates that the supplied parameter had an incorrect value.

**VX_ERROR_INVALID_DIMENSION**  
Indicates that the supplied parameter was too big or too small in dimension.

**VX_ERROR_INVALID_FORMAT**  
Indicates that the supplied parameter was in an invalid format.

**VX_ERROR_INVALID_LINK**  
Indicates that the link was not possible as specified. The parameters were incompatible.

**VX_ERROR_INVALID_REFERENCE**  
Indicates that the reference provided was not valid.

**VX_ERROR_INVALID_MODULE**  
This is returned from vxLoadKernels when the module did not contain the entry point.
**VX_ERROR_INVALID_PARAMETERS**  The supplied parameter information did not match the kernel contract.

**VX_ERROR_OPTIMIZED_AWAY**  This code indicates that the object refered to has been optimized out of existence.

**VX_ERROR_NO_MEMORY**  An internal or implicit allocation failed. Typically catastrophic. After detection, deconstruct the context.
   See Also
   vxVerifyGraph

**VX_ERROR_NO_RESOURCES**  An internal or implicit resource could not be acquired (not memory). This is typically catastrophic. After detection, deconstruct the context.
   See Also
   vxVerifyGraph

**VX_ERROR_NOT_COMPATIBLE**  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**  The given graph has failed verification due to an insufficient number of required parameters which can not be automatically created. Typically this indicate required atomic parameters.
   See Also
   vxVerifyGraph

**VX_ERROR_NOT_SUPPORTED**  The requested set of parameter produce a configuration which can not be supported. Refer to the supplied documentation on the configured kernels.
   See Also
   vx_kernel_e

**VX_ERROR_NOT_IMPLEMENTED**  The requested kernel is missing.
   See Also
   vx_kernel_e vxGetKernelByName

**VX_FAILURE**  The generic error code, used when no other will describe the error.

**VX_SUCCESS**  No error.

Definition at line 323 of file vx_types.h.

eenum vx_type_e

The type enumeration lists all the known types in OpenVX.

**Enumenator**

**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_FOURCC A vx_fourcc
VX_TYPE_BOOLEAN A vx_bool
VX_TYPE_SCALAR_MAX A floating value for comparison between scalars and structs.
VX_TYPE_RECTANGLE A vx_rectangle_t
VX_TYPE_KEYPOINT A vx_keypoint_t
VX_TYPE_COORDINATES2D A vx_coordinates2d_t
VX_TYPE_COORDINATES3D A vx_coordinates3d_t
VX_TYPE_STRUCT_MAX A floating value for comparison between structs and 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_TYPEOBJECT_MAX A value used for bound checking the object types.

Definition at line 260 of file vx_types.h.

e num vx_vendor_id_e

The Vendor ID of the Implementation. As new vendors submit their implementations, this enumeration will grow.

Enum

VX_ID_KHRONOS The Khronos Group.
VX_ID_TI Texas Instruments, Inc.
VX_ID_QUALCOMM Qualcomm, Inc.
VX_ID_NVIDIA NVIDIA Corporation.
VX_ID_ARM ARM Ltd.
VX_ID_BDTI Berkley Design Technology, Inc.
VX_ID_RENESAS Renasas Electronics.
### 3.42.6 Function Documentation

#### `vx_status vxGetStatus (vx_reference reference)`

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 Runtime.

```c
vx_image img = vxCreateImage(context, 639, 480, FOURCC_UYVY);
vx_status status = vxGetStatus((vx_reference)img);
// status == VX_ERROR_INVALID_DIMENSIONS
vxReleaseImage(&img);
```

**Precondition**

Appropriate Object Creator function.

**Postcondition**

Appropriate Object Release function.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>reference</th>
<th>The reference to check for construction errors.</th>
</tr>
</thead>
</table>

**Returns**

Return a `vx_status_e` enumeration.

**Return values**

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No error.</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Some error occurred, please check enumeration list and constructor.</td>
</tr>
</tbody>
</table>
3.43 Objects

3.43.1 Detailed Description

The basic objects within OpenVX. All objects in OpenVX derive from a `vx_reference` and contain a reference to the `vx_context` from which they were made, except the `vx_context` itself.

Modules

- **Object: Array**
  The Array Object Interface.
- **Object: Context**
  The Context Object Interface.
- **Object: Convolution**
  The Image Convolution Object interface.
- **Object: Distribution**
  The Distribution Object Interface.
- **Object: Graph**
  The Graph Object interface.
- **Object: Image**
  The Image Object interface.
- **Object: LUT**
  The Look-Up Table Interface.
- **Object: Matrix**
  The Matrix Object Interface.
- **Object: Node**
  The Node Object interface.
- **Object: Pyramid**
  The Image Pyramid Object Interface.
- **Object: Reference**
  The Reference Object interface.
- **Object: Remap**
  The Remap Object Interface.
- **Object: Scalar**
  The Scalar Object interface.
- **Object: Threshold**
  The Threshold Object Interface.
3.44 Object: Reference

3.44.1 Detailed Description

The Reference Object interface. All objects in OpenVX are derived (in the Object Oriented sense) from \texttt{vxreference}. All objects shall be able to be cast back to this type safely.

**Typedefs**

- typedef struct \texttt{vxreference} * \texttt{vxreference}

  A generic opaque reference to any object within OpenVX.

**Enumerations**

- enum \texttt{vxreference(attribute} e
  
  \begin{verbatim}
  VX_REF_ATTRIBUTE_COUNT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_REFERENCE << 8)) + 0x0,
  VX_REF_ATTRIBUTE_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_REFERENCE << 8)) + 0x1
  \end{verbatim}

  The reference attributes list.

**Functions**

- \texttt{vxstatus vxQueryReference (vxreference ref, vxenum attribute, void *ptr, vxsize size)}

  Used to query any reference type for some basic information (count, type)

3.44.2 Typedef Documentation

typedef struct \texttt{vxreference} \* \texttt{vxreference}

A generic opaque reference to any object within OpenVX.

A user of OpenVX should not assume that this can be casted 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 which take a \texttt{vxreference} 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 112 of file \texttt{vxtypes.h}.

3.44.3 Enumeration Type Documentation

enum \texttt{vxreference(attribute} e

The reference attributes list.

**Enumerator**

\begin{verbatim}
VX_REF_ATTRIBUTE_COUNT \quad Returns the reference count of the object. Use a \texttt{vxuint32} parameter.
VX_REF_ATTRIBUTE_TYPE \quad Returns the \texttt{vxtype} of the reference. Use a \texttt{vxenum} parameter.
\end{verbatim}

Definition at line 635 of file \texttt{vxtypes.h}.

3.44.4 Function Documentation

\texttt{vxstatus vxQueryReference ( \texttt{vxreference ref, vxenum attribute, void *ptr, vxsize size) }

Used to query any reference type for some basic information (count, type)

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>The reference to query.</td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>attribute</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</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.
3.45 Object: Context

3.45.1 Detailed Description

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 is private in that the references which refer to data objects are given only to the creating party.

Macros

- \#define VX_MAX_IMPLEMENTATION_NAME (64)

  Defines the maximum number of characters in a implementation string.

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_ATTRIBUTE_VENDOR_ID = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x0,
  VX_CONTEXT_ATTRIBUTE_VERSION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x1,
  VX_CONTEXT_ATTRIBUTE_NUMKERNELS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x2,
  VX_CONTEXT_ATTRIBUTE_NUMMODULES = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x3,
  VX_CONTEXT_ATTRIBUTE_NUMREFS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x4,
  VX_CONTEXT_ATTRIBUTE_IMPLEMENTATION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x5,
  VX_CONTEXT_ATTRIBUTE_EXTENSIONS_SIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x6,
  VX_CONTEXT_ATTRIBUTE_EXTENSIONS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x7,
  VX_CONTEXT_ATTRIBUTE_CONVOLUTION_MAXIMUM_DIMENSION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x8,
  VX_CONTEXT_ATTRIBUTE_OPTICAL_FLOW_WINDOW_MAXIMUM_DIMENSION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x9,
  VX_CONTEXT_ATTRIBUTE_IMMEDIATE_BORDER_MODE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0xA,
  VX_CONTEXT_ATTRIBUTE_KERNELTABLE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0xB
}

  A list of context attributes.

- enum vx_import_type_e {
  VX_IMPORT_TYPE_NONE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_IMPORT_MEM << 12)) + 0x0,
  VX_IMPORT_TYPE_HOST = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_IMPORT_MEM << 12)) + 0x1
}
An enumeration of memory import types.

- `enum vx_round_policy_t { 
  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_t { 
  VX_TERM_CRITERIA_ITERATIONS = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_TERM_CRITERIA << 12)) + 0x0, 
  VXTERM_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 vxCreateContext ()`  
  Creates a `vx_context`

- `vx_context vxGetContext (vx_reference reference)`  
  Retrieves the context from any reference from within a context.

- `vx_status vxQueryContext (vx_context context, vx_enum attribute, void *ptr, vx_size size)`  
  Queries the context for some specific information.

- `void vxReleaseContext (vx_context *context)`  
  Releases the OpenVX object context.

- `vx_status vxSetContextAttribute (vx_context context, vx_enum attribute, void *ptr, vx_size size)`  
  Sets an attribute on the context.

### 3.45.2 Typedef Documentation

typedef struct _vx_context* vx_context

An opaque reference to the implementation context.

See Also

vxCreateContext

Definition at line 180 of file `vx_types.h`.

### 3.45.3 Enumeration Type Documentation

class `enum vx_accessor_t`

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 1105 of file `vx_types.h`.  

enum vx_context_attribute_e
A list of context attributes.

Enumerator

- **VX_CONTEXT_ATTRIBUTE_VENDOR_ID** Used to query the unique vendor ID. Use a `vx_uint16`.
- **VX_CONTEXT_ATTRIBUTE_VERSION** Used to query the OpenVX Version Number. Use a `vx_uint16`.
- **VX_CONTEXT_ATTRIBUTE_NUMKERNELS** Used to query the context for the number of active kernels. Use a `vx_uint32` parameter.
- **VX_CONTEXT_ATTRIBUTE_NUMMODULES** Used to query the context for the number of active modules. Use a `vx_uint32` parameter.
- **VX_CONTEXT_ATTRIBUTE_NUMREFS** Used to query the context for the number of active references. Use a `vx_uint32` parameter.
- **VX_CONTEXT_ATTRIBUTE_IMPLEMENTATION** Used to query the context for its implementation name. Use a `vx_char[VX_MAX_IMPLEMENTATION_NAME]` array.
- **VX_CONTEXT_ATTRIBUTE_EXTENSIONS_SIZE** Used to query the number of bytes in the extensions string. Use a `vx_size` parameter.
- **VX_CONTEXT_ATTRIBUTE_EXTENSIONS** Used to retrieve the extensions string. This is a space separated string of extension names. Use a `vx_char` pointer allocated to the size returned from `VX_CONTEXT_ATTRIBUTE_EXTENSIONS_SIZE`.
- **VX_CONTEXT_ATTRIBUTE_CONVOLUTION_MAXIMUM_DIMENSION** The maximum width or height of a convolution matrix. Use a `vx_size` parameter. Each vendor will have to support centered kernels of size w \times h, where both w and h are odd numbers, 3 \leq w \leq n and 3 \leq h \leq n, where n is the value of the `VX_CONTEXT_ATTRIBUTE_CONVOLUTION_MAXIMUM_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 should be supported. The behavior of `vxCreateConvolution` is undefined for values larger than the value returned by this attribute.
- **VX_CONTEXT_ATTRIBUTE_OPTICAL_FLOW_WINDOW_MAXIMUM_DIMENSION** The maximum window dimension of the OpticalFlowPyrLK kernel. See Also
  - **VX_KERNEL_OPTICAL_FLOW_PYR_LK**. Use a `vx_size` parameter.
- **VX_CONTEXT_ATTRIBUTE_IMMEDIATE_BORDER_MODE** The border mode for immediate mode functions. Graph mode functions are unaffected by this attribute. Use a pointer to a `vx_border_mode_t` structure as parameter.
  - **Note** The assumed default value for immediate mode functions is `VX_BORDER_MODE_UNDEFINED`.
- **VX_CONTEXT_ATTRIBUTE_KERNELTABLE** Returns the table of all the kernels that exist in the context. Use a `vx_kernel_info_t` array.
  - **Precondition** You must call `vxQueryContext` with `VX_CONTEXT_ATTRIBUTE_NUMKERNELS` to compute the necessary size of the array.

Definition at line 645 of file `vx_types.h`.

---

enum vx_import_type_e
An enumeration of memory import types.

Enumerator

- **VX_IMPORT_TYPE_NONE** For memory allocated through OpenVX, this is the import type.
- **VX_IMPORT_TYPE_HOST** The default memory type to import from the Host.

Definition at line 964 of file `vx_types.h`. 
enum vx_round_policy_e
The Round Policy Enumeration.

Enumerator

\texttt{VX\_ROUND\_POLICY\_TO\_ZERO} When scaling, this will truncate the least significant values which are lost in operations.

\texttt{VX\_ROUND\_POLICY\_TO\_NEAREST\_EVEN} When scaling, this will round to nearest even output value.

Definition at line 1122 of file \texttt{vx\_types.h}.

enum vx_termination_criteria_e
The termination criteria list.

See Also

Function: Optical Flow Pyramid (LK)

Enumerator

\texttt{VX\_TERM\_CRITERIA\_ITERATIONS} Indicates a termination after a set number of iterations.

\texttt{VX\_TERM\_CRITERIA\_EPSILON} Indicates a termination after matching against the value of epsilon provided to the function.

\texttt{VX\_TERM\_CRITERIA\_BOTH} Indicates that both an iterations and epsilon method are employed. Whichever one matches first, will cause the termination.

Definition at line 1067 of file \texttt{vx\_types.h}.

3.45.4 Function Documentation

\texttt{vx\_context vxCreateContext ( )}

Creates a \texttt{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.

Return values

\begin{tabular}{|c|c|}
\hline
\texttt{0} & No context was created. \\
\hline
\texttt{*} & A context reference. \\
\hline
\end{tabular}

Postcondition

\texttt{vxReleaseContext}

\texttt{vx\_context vxGetContext ( vx\_reference reference )}

Retrieves the context from any reference from within a context.

Parameters
CHAPTER 3. MODULE DOCUMENTATION

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td>The reference to the extract the context from.</td>
</tr>
</tbody>
</table>

Returns

Returns the overall context which created the particular reference.

```c
vx_status vxQueryContext ( vx_context context, vx_enum attribute, void * ptr, vx_size size )
```

Queries 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 <code>vx_context_attribute_e</code>.</td>
</tr>
<tr>
<td>ptr</td>
<td>The location at which the resulting value will be stored.</td>
</tr>
<tr>
<td>size</td>
<td>The size 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</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>If the context is not a <code>vx_context</code>.</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>

```c
void 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>The pointer to the reference to the context.</td>
</tr>
</tbody>
</table>

**Note**

After returning from this function the reference will be zeroed.

**Precondition**

`vxCreateContext`  

```c
vx_status vxSetContextAttribute ( vx_context context, vx_enum attribute, void * ptr, vx_size size )
```

Sets 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>
</tbody>
</table>
### CHAPTER 3. MODULE DOCUMENTATION

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>attribute</strong></td>
<td>The attribute to set from <code>vx_context_attribute_e</code>.</td>
</tr>
<tr>
<td>in</td>
<td><strong>ptr</strong></td>
<td>The pointer to the data to set the attribute to.</td>
</tr>
<tr>
<td>in</td>
<td><strong>size</strong></td>
<td>The size in bytes of the data that ptr points to.</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 context is not a <code>vx_context</code>.</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>
3.46 Object: Graph

3.46.1 Detailed Description

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 which are unconnected to other sets of Nodes within the same Graph. See Graph Formalisms.

Typedefs

- typedef struct _vx.graph * vx.graph
  
  An opaque reference to a graph.

Enumerations

- enum vx.graph.attribute.e {
  
  VX_GRAPH_ATTRIBUTE_NUMNODES = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_GRAPH << 8)) + 0x0,
  VX_GRAPH_ATTRIBUTE_STATUS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_GRAPH << 8)) + 0x1,
  VX_GRAPH_ATTRIBUTE_PERFORMANCE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_GRAPH << 8)) + 0x2,
  VX_GRAPH_ATTRIBUTE_NUMPARAMETERS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_GRAPH << 8)) + 0x3
  }

  The graph attributes list.

Functions

- vx.graph vxCreateGraph (vx.context context)
  
  Creates an empty graph.

- vx.bool vxIsGraphVerified (vx.graph graph)
  
  Returns a boolean to indicate the state of graph verification.

- vx.status vxProcessGraph (vx.graph graph)
  
  This function causes the synchronous processing of a graph. If the graph has not been verified, then the implementation will verify the graph immediately. If verification fails this function will return a status identical to what vxVerifyGraph would return. After the graph verifies successfully then processing will occur. If the graph was previously verified via vxVerifyGraph or vxProcessGraph then the graph is processed. This function will block until the graph is completed.

- vx.status vxQueryGraph (vx.graph graph, vx.enum attribute, void *ptr, vx.size size)
  
  Allows the user to query attributes of the Graph.

- void 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 will be automatically released as well. Data referenced by those nodes may not be released as the user may have external references to the data.

- vx.status vxScheduleGraph (vx.graph graph)
  
  Schedules a graph for future execution.

- vx.status vxSetGraphAttribute (vx.graph graph, vx.enum attribute, void *ptr, vx.size size)
  
  Allows the set to attributes on the Graph.

- vx.status vxVerifyGraph (vx.graph graph)
  
  This call 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 will verify before being processed.

- vx.status vxWaitGraph (vx.graph graph)
  
  Waits for a specific graph to complete.
3.46.2 Typedef Documentation

typedef struct vx_graph * vx_graph

An opaque reference to a graph.

See Also

vxCreateGraph

Definition at line 173 of file vx_types.h.

3.46.3 Enumeration Type Documentation

enum vx_graph_attribute_e

The graph attributes list.

Enumerator

VX_GRAPH_ATTRIBUTE_NUMNODES Returns the number of nodes in a graph. Use a vx_uint32 parameter.

VX_GRAPH_ATTRIBUTE_STATUS Returns the overall status of the graph. Use a vx_status parameter.

VX_GRAPH_ATTRIBUTE_PERFORMANCE Returns the overall performance of the graph. Use a vx_perf_t parameter.

VX_GRAPH_ATTRIBUTE_NUMPARAMETERS Returns the number of explicitly declared parameters on the graph. Use a vx_uint32 parameter.

Definition at line 788 of file vx_types.h.

3.46.4 Function Documentation

vx_graph vxCreateGraph ( vx_context context )

Creates an empty graph.

Parameters

in context The reference to the implementation context.

Returns

Return a graph reference

Return values

0 if an error occurred.

vx_bool 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 not verified. It must be verified before execution either through vxVerifyGraph or automatically through vxProcessGraph, vxScheduleGraph. |

**vx_status vxProcessGraph ( vx_graph graph )**

This function causes the synchronous processing of a graph. If the graph has not been verified, the implementation will verify the graph immediately. If verification fails the function will return a status identical to what vxVerifyGraph would return. After the graph verifies successfully then processing will occur. If the graph was previously verified via vxVerifyGraph or vxProcessGraph then the graph is processed. This function will block until the graph is completed.

**Parameters**

| in | graph | The graph to execute. |

**Return values**

| VX.SUCCESS | Graph has been processed. |
| VX.FAILURE | A catastrophic error occurred during processing. |

* See vxVerifyGraph

**Precondition**

vxVerifyGraph must return VX.SUCCESS before this function will pass.

**See Also**

vxVerifyGraph

**vx_status 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 the resulting value will be stored. |
| in | size | The size of the container to which ptr points. |

**Returns**

A vx_status_e enumeration.

**void 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 will be automatically released as well. Data referenced by those nodes may not be released as the user may have external references to the data.
Parameters

| in  | graph               | The pointer to the graph to release. |

Note

After returning from this function the reference will be zeroed.

**vx_status vxScheduleGraph ( vx_graph graph )**

Schedules a graph for future execution.

Parameters

| in  | graph               | The graph to schedule. |

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_ERROR_NO_RESOURCES</td>
<td>The graph can not be scheduled now.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUFFICIENT</td>
<td>The graph was not verified and has failed forced verification.</td>
</tr>
<tr>
<td>VX_SUCCESS</td>
<td>The graph has been scheduled.</td>
</tr>
</tbody>
</table>

Precondition

**vxVerifyGraph** must return **VX_SUCCESS** before this function will pass.

**vx_status vxSetGraphAttribute ( vx_graph graph, vx_enum attribute, void * ptr, vx_size size )**

Allows the set to attributes on the Graph.

Parameters

| in  | graph               | The reference to the graph.                                              |
| in  | attribute           | The **vx_graph_attribute_e** type needed.                                |
| in  | ptr                 | The location at which the value will be read from.                       |
| in  | size                | The size of the container to which ptr points.                           |

Returns

A **vx_status_e** enumeration.

**vx_status vxVerifyGraph ( vx_graph graph )**

This call 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 will verify before being processed.

Note

Memory for data objects is not guaranteed to exist before this call. After this call data objects will 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.

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 graph is not a vx.graph.</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 was 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 was not compatible.</td>
</tr>
</tbody>
</table>

See Also

vxConvertReference
vxProcessGraph

**vx_status vxWaitGraph ( vx_graph graph )**

Waits for a specific graph to complete.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>graph</th>
</tr>
</thead>
</table>
The graph to wait on.

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>The graph has completed.</td>
</tr>
<tr>
<td>VX_FAILURE</td>
<td>The graph has not completed yet</td>
</tr>
</tbody>
</table>

Precondition

vxScheduleGraph
3.47 Object: Node

3.47.1 Detailed Description

The Node Object interface. A node is an instance of a kernel which 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_ATTRIBUTE_STATUS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x0,
  VX_NODE_ATTRIBUTE_PERFORMANCE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x1,
  VX_NODE_ATTRIBUTE_BORDER_MODE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x2,
  VX_NODE_ATTRIBUTE_LOCAL_DATA_SIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x3,
  VX_NODE_ATTRIBUTE_LOCAL_DATA_PTR = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_NODE << 8)) + 0x4
  }

  The node attributes list.

**Functions**

- vx_status vxQueryNode (vx_node node, vx_enum attribute, void ∗ptr, vx_size size)
  
  Allows a user to query information out of a node.

- void 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.

- void vxRemoveNode (vx_node ∗node)
  
  Removes a Node from it's parent Graph and releases it.

- vx_status vxSetNodeAttribute (vx_node node, vx_enum attribute, void ∗ptr, vx_size size)
  
  Allows a user to set attribute of a node before Graph Validation.

3.47.2 Typedef Documentation

typedef struct vx_node ∗ vx_node

An opaque reference to a kernel node.

See Also

vxCreateNode

Definition at line 166 of file vx_types.h.
3.47.3 Enumeration Type Documentation

```c
enum vx_node_attribute_e
```

The node attributes list.

Enumerator

- **VX_NODE_ATTRIBUTE_STATUS**: Used to query the status of node execution. Use a `vx_status` parameter.
- **VX_NODE_ATTRIBUTE_PERFORMANCE**: Used to query the performance of the node execution. Use a `vx_perf_t` parameter.
- **VX_NODE_ATTRIBUTE_BORDER_MODE**: Used to get or set the border mode of the node. Use a `vx_border_mode_t` structure.
- **VX_NODE_ATTRIBUTE_LOCAL_DATA_SIZE**: Used to indicate the size of the kernel local memory area. Use a `vx_size` parameter.
- **VX_NODE_ATTRIBUTE_LOCAL_DATA_PTR**: Used to indicate the pointer kernel local memory area. Use a `void *` parameter.

Definition at line 722 of file `vx_types.h`.

3.47.4 Function Documentation

```c
vx_status vxQueryNode ( vx_node node, vx_enum attribute, void * ptr, vx_size size )
```

Allows a user to query information out of a node.

**Parameters**

- `node`: The reference to the node to query.
- `attribute`: Use `vx_node_attribute_e` value to query for information.
- `ptr`: The location at which the resulting value will be stored.
- `size`: The size 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>Successful</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>The type or size was incorrect.</td>
</tr>
</tbody>
</table>

```c
void 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**

- `node`: The pointer to the reference of the node to release.

**Note**

After returning from this function the reference will be zeroed.

```c
void vxRemoveNode ( vx_node * node )
```

Removes a Node from it's parent Graph and releases it.
Parameters

| in  | node | The pointer to the node to remove and release. |

Note

After returning from this function the reference will be zeroed.

\[
\text{vx\_status vxSetNodeAttribute ( vx\_node node, vx\_enum attribute, void * ptr, vx\_size size )}
\]

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 query for information. |
| out   | ptr  | The output pointer where the value will be sent. |
| in    | size | The size of the objects to which ptr points. |

Note

Some attributes are inherited from the \text{vx\_kernel} which was used to create the node. Some of these can be overridden using this API, notable \text{VX\_NODE\_ATTRIBUTE\_LOCAL\_DATA\_SIZE, VX\_NODE\_ATTRIBUTE\_LOCAL\_DATA\_PTR}.

Returns

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

Return values

| \text{VX\_SUCCESS} | The attribute was set. |
| \text{VX\_ERROR\_INVALID\_REFERENCE} | node was not a vx\_node. |
| \text{VX\_ERROR\_INVALID\_PARAMETER} | size was not correct for the type needed. |
3.48 Object: Array

3.48.1 Detailed Description

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 own elements and holds copies of data. This is an example “for” loop over an Array:

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

Macros

- `#define vxArrayItem(type, ptr, index, stride) (*(type *)((vx_uint8 *)ptr)[index * stride])`
  
  Allows access to an array item as a typecast pointer deference.

- `#define vxFormatArrayPointer(ptr, index, stride) (&(((vx_uint8 *)ptr)[index * stride]))`
  
  Used to access 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_ATTRIBUTE_ITEMTYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_ARRAY << 8)) + 0x0,
  VX_ARRAY_ATTRIBUTE_NUMITEMS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_ARRAY << 8)) + 0x1,
  VX_ARRAY_ATTRIBUTE_CAPACITY = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_ARRAY << 8)) + 0x2,
  VX_ARRAY_ATTRIBUTE.ITEMSIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_ARRAY << 8)) + 0x3
}

  The array object attributes.

Functions

- `vx_status vxAccessArrayRange (vx_array arr, vx_size start, vx_size end, vx_size *stride, void **ptr, vx_enum usage)`

  Grant access to a sub-range of an Array.

- `vx_status vxAddArrayItems (vx_array arr, vx_size count, void *ptr, vx_size stride)`

  Adds items to the Array.

- `vx_status vxCommitArrayRange (vx_array arr, vx_size start, vx_size end, void *ptr)`

  Commits data back to the Array object.

- `vx_array vxCreateArray (vx_context context, vx_enum item_type, vx_size capacity)`

  Creates a reference to an Array object.

- `vx_array 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 vxQueryArray (vx_array arr, vx_enum attribute, void *ptr, vx_size size)`

  Queries the Array for some specific information.

- `void 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 will be zeroed.

- `vx_status vxTruncateArray (vx_array arr, vx_size new_num_items)`

  Truncate Array (remove items from the end).
3.48.2 Macro Definition Documentation

#define vxArrayItem(type, ptr, index, stride) ((type *)((type *)vxFormatArrayPointer(ptr, index, stride)))

Allows access to an array item as a typecast pointer deference.
### Parameters

<table>
<thead>
<tr>
<th></th>
<th>type</th>
<th>The type of the item to access.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ptr</td>
<td>The base pointer for the array range.</td>
</tr>
<tr>
<td></td>
<td>index</td>
<td>The index of the element, not byte, to access.</td>
</tr>
<tr>
<td></td>
<td>stride</td>
<td>The stride of the array range given by <code>vxAccessArrayRange</code></td>
</tr>
</tbody>
</table>

Definition at line 1720 of file `vx_api.h`.

```c
#define vxFormatArrayPointer( ptr, index, stride ) (&(((vx_uint8*)ptr)[index * stride]))
```

Used to access a specific indexed element in an array.

### Parameters

<table>
<thead>
<tr>
<th></th>
<th>ptr</th>
<th>The base pointer for the array range.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>index</td>
<td>The index of the element, not byte, to access.</td>
</tr>
<tr>
<td></td>
<td>stride</td>
<td>The stride of the array range given by <code>vxAccessArrayRange</code></td>
</tr>
</tbody>
</table>

Definition at line 1709 of file `vx_api.h`.

### 3.48.3 Enumeration Type Documentation

#### `enum vx_array_attribute_e`

The array object attributes.

#### Enumerator

- `VX_ARRAY_ATTRIBUTE_ITEMTYPE` The type of the Array items. Use a `vx_enum` parameter.
- `VX_ARRAY_ATTRIBUTE_NUMITEMS` The number of items in the Array. Use a `vx_size` parameter.
- `VX_ARRAY_ATTRIBUTE_CAPACITY` The maximal number of items that the Array can hold. Use a `vx_size` parameter.
- `VX_ARRAY_ATTRIBUTE_ITEMSIZE` Used to query an array item size. Use a `vx_size` parameter.

Definition at line 918 of file `vx_types.h`.

### 3.48.4 Function Documentation

#### `vx_status vxAccessArrayRange ( vx_array arr, vx_size start, vx_size end, vx_size * stride, void ** ptr, vx_enum usage )`

Grant access to a sub-range of an Array.

#### Parameters

<table>
<thead>
<tr>
<th></th>
<th>arr</th>
<th>The reference to the Array.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>start</td>
<td>The start index.</td>
</tr>
<tr>
<td></td>
<td>end</td>
<td>The end index.</td>
</tr>
<tr>
<td></td>
<td>stride</td>
<td>The stride in bytes between elements.</td>
</tr>
<tr>
<td></td>
<td>ptr</td>
<td>The user-supplied pointer to a pointer, via which the requested contents will be returned. If (*ptr) is non-NULL, data is copied to it, else (*ptr) is set to the address of existing internal memory, allocated, or mapped memory. (*ptr) must be given to <code>vxCommitArrayRange</code>. Use a <code>vx_rectangle_t</code> for <code>VX_TYPE_RECTANGLE</code> and a <code>vx_keypoint_t</code> for <code>VX_TYPE_KEYPOINT</code>.</td>
</tr>
</tbody>
</table>
This declares the intended usage of the pointer using the \texttt{vx accessor_e} enumeration.

Returns

A \texttt{vx status e} enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>If the \texttt{arr} is not a \texttt{vx array}.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>If any of the other parameters are incorrect.</td>
</tr>
</tbody>
</table>

Postcondition

\texttt{vxCommitArrayRange}

\texttt{vx status vxAddArrayItems ( vx array arr, vx size count, void * ptr, vx size stride )}

Adds items to the Array.

This function increases the container size.

By default, the function will 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 will return \texttt{VX_FAILURE} error code.

Parameters

| in | arr | The reference to the Array. |
| in | count | The total number of elements to insert. |
| in | ptr | The location at which the input values is stored. |
| in | stride | The stride in bytes between elements. User can pass 0, which means that stride is equal to item size. |

Returns

A \texttt{vx status e} enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>If the \texttt{arr} is not a \texttt{vx array}.</td>
</tr>
<tr>
<td>VX_FAILURE</td>
<td>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>

\texttt{vx status vxCommitArrayRange ( vx array arr, vx size start, vx size end, void * ptr )}

Commits data back to the Array object.

This allows a user to commit data to a sub-range of an Array.

Parameters

| in | arr | The reference to the Array. |
| in | start | The start index. |
| in | end | The end index. |
in | The user supplied pointer.

Returns

A `vx_status_e` enumeration.

Return values

<table>
<thead>
<tr>
<th>VX_SUCCESS</th>
<th>No errors.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>If the <code>arr</code> is not a <code>vx_array</code>.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>If any of the other parameters are incorrect.</td>
</tr>
</tbody>
</table>

`vx_array vxCreateArray ( vx_context context, vx_enum item_type, vx_size capacity )`

Creates a reference to an Array object.

User must specify the Array capacity (maximal number of items that the array can hold).

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 objects to hold. Use:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• VX_TYPE_RECTANGLE for <code>vx_rectangle_t</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• VX_TYPE_KEYPOINT for <code>vx_keypoint_t</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• VX_TYPE_COORDINATES2D for <code>vx_coordinates2d_t</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• VX_TYPE_COORDINATES3D for <code>vx_coordinates3d_t</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>vx_enum</code> Returned from <code>vxRegisterUserStruct</code></td>
</tr>
<tr>
<td>in</td>
<td>capacity</td>
<td>The maximal number of items that the array can hold.</td>
</tr>
</tbody>
</table>

Returns

`vx_array`

Return values

| 0 | No Array was created. |
| * | An Array was created. |

`vx_array 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
vx_context context = vxCreateContext();
vx_graph graph = vxCreateGraph(context);
vx_array virt[] = {
    vxCreateVirtualArray(graph, 0, 0), // totally unspecified
    vxCreateVirtualArray(graph, VX_TYPE_KEYPOINT, 0), // unspecified
capacity
    vxCreateVirtualArray(graph, VX_TYPE_KEYPOINT, 1000), // no 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>item_type</td>
<td>The type of objects to hold. This may to 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 to set to zero to indicate an unspecified capacity.</td>
</tr>
</tbody>
</table>

See Also

vxCreateArray for a type list.

Returns

vx_array

Return values

| 0 | No Array was created. |
| * | An Array was created or an error occurred. Use vxGetStatus to determine. |

vx_status vxQueryArray ( vx_array arr, vx_enum attribute, void * ptr, vx_size size )

Queries the Array for some specific information.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>arr</th>
<th>The reference to the Array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>attribute</td>
<td>The attribute to query. Use a vx_array_attribute_e.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The location at which the resulting value will be stored.</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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>If the arr is not a vx_array.</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>

void 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 will be zeroed.

Parameters

| in  | arr | The pointer to the Array to release. |

vx_status vxTruncateArray ( vx_array arr, vx_size new_num_items )

Truncate Array (remove items from the end).
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out arr</td>
<td>The reference to the Array.</td>
</tr>
<tr>
<td>in new num_items</td>
<td>The new number of items for the Array.</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.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>If the <code>arr</code> is not a <code>vx_array</code>.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>The <code>new_size</code> is greater than the current size.</td>
</tr>
</tbody>
</table>
3.49 Object: Convolution

3.49.1 Detailed Description

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_ATTRIBUTE_ROWS** = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONVOLUTION << 8)) + 0x0,
  
  **VX_CONVOLUTION_ATTRIBUTE_COLUMNS** = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONVOLUTION << 8)) + 0x1,
  
  **VX_CONVOLUTION_ATTRIBUTE_SCALE** = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONVOLUTION << 8)) + 0x2,
  
  **VX_CONVOLUTION_ATTRIBUTE_SIZE** = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONVOLUTION << 8)) + 0x3
  
  *The convolution attributes.*

**Functions**

- vx_status vxAccessConvolutionCoefficients (vx_convolution conv, vx_int16 *array)

  *Gets the convolution data (copy)*

- vx_status vxCommitConvolutionCoefficients (vx_convolution conv, vx_int16 *array)

  *Sets the convolution data (copy)*

- vx_convolution vxCreateConvolution (vx_context context, vx_size columns, vx_size rows)

  *Creates a reference to a convolution matrix object.*

- vx_status vxQueryConvolution (vx_convolution conv, vx_enum attribute, void *ptr, vx_size size)

  *Queries an attribute on the convolution matrix object.*

- void 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 vxSetConvolutionAttribute (vx_convolution conv, vx_enum attribute, void *ptr, vx_size size)

  *Sets attributes on the convolution object.*

3.49.2 Enumeration Type Documentation

**enum vx_convolution_attribute_e**

*The convolution attributes.*

**Enumerator**

- VX_CONVOLUTION_ATTRIBUTE_ROWS  The number of rows of the convolution matrix. Use a **vx_size** parameter.

- VX_CONVOLUTION_ATTRIBUTE_COLUMNS  The number of columns of the convolution matrix. Use a **vx_int32** parameter.

- VX_CONVOLUTION_ATTRIBUTE_SCALE  The scale of the convolution matrix. Use a **vx_uint32** parameter.
Note
For 1.0, only powers of 2 are supported up to $2^{31}$.

**VX_CONVOLUTION_ATTRIBUTE_SIZE** The total size of the convolution matrix in bytes. Use a **vx_size** parameter.

Definition at line 870 of file **vx_types.h**.

### 3.49.3 Function Documentation

**vx_status vxAccessConvolutionCoefficients ( vx_convolution conv, vx_int16 * array )**

Gets the convolution data (copy)

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>conv</th>
<th>The reference to the convolution.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>array</td>
<td>The array to place the convolution.</td>
</tr>
</tbody>
</table>

**See Also**

**vxQueryConvolution** and **VX_CONVOLUTION_ATTRIBUTE_SIZE** to get the needed number of bytes of the array.

**Returns**

A **vx_status_e** enumeration.

**Postcondition**

**vxCommitConvolutionCoefficients**

**vx_status vxCommitConvolutionCoefficients ( vx_convolution conv, vx_int16 * array )**

Sets the convolution data (copy)

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>conv</th>
<th>The reference to the convolution.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>array</td>
<td>The array to read the convolution.</td>
</tr>
</tbody>
</table>

**See Also**

**vxQueryConvolution** and **VX_CONVOLUTION_ATTRIBUTE_SIZE** to get the needed number of bytes of the array.

**Returns**

A **vx_status_e** enumeration.

**Precondition**

**vxAccessConvolutionCoefficients**

**vx_convolution vxCreateConvolution ( vx_context context, vx_size columns, 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 <code>VX_CONTEXT_ATTRIBUTE_CONVOLUTION_MAXIMUM_DIMENSION</code>.</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 <code>VX_CONTEXT_ATTRIBUTE_CONVOLUTION_MAXIMUM_DIMENSION</code>.</td>
</tr>
</tbody>
</table>

### Returns

```
vx_convolution
```

#### vx_status 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>in</th>
<th>conv</th>
<th>The convolution matrix object to set.</th>
</tr>
</thead>
<tbody>
<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 the resulting value will be stored.</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.

#### void 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.

### Parameters

| in   | conv | The pointer to the convolution matrix to release. |

### Note

After returning from this function the reference will be zeroed.

#### vx_status vxSetConvolutionAttribute( vx_convolution conv, vx_enum attribute, void * ptr, vx_size size )

Sets attributes on the convolution object.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>conv</th>
<th>The coordinates object to set.</th>
</tr>
</thead>
<tbody>
<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 set the attribute to.</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>The size of the data pointed to by ptr.</td>
</tr>
</tbody>
</table>

### Returns

A `vx_status_e` enumeration.
3.50 Object: Distribution

3.50.1 Detailed Description

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_ATTRIBUTE_DIMENSIONS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DISTRIBUTION << 8)) + 0x0,
  VX_DISTRIBUTION_ATTRIBUTE_OFFSET = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DISTRIBUTION << 8)) + 0x1,
  VX_DISTRIBUTION_ATTRIBUTE_RANGE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DISTRIBUTION << 8)) + 0x2,
  VX_DISTRIBUTION_ATTRIBUTE_BINS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DISTRIBUTION << 8)) + 0x3,
  VX_DISTRIBUTION_ATTRIBUTE_WINDOW = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DISTRIBUTION << 8)) + 0x4,
  VX_DISTRIBUTION_ATTRIBUTE_SIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DISTRIBUTION << 8)) + 0x5
}

The distribution attribute list.

Functions

- vx_status vxAccessDistribution (vx_distribution distribution, void **ptr, vx_enum usage)
  Gets direct access to a Distribution in memory.
- vx_status vxCommitDistribution (vx_distribution distribution, void *ptr)
  Sets the Distribution back to the memory. The memory is the array must be a vx_uint32 array of a value at least as big as the value returned via VX_DISTRIBUTION_ATTRIBUTE_RANGE.
- vx_distribution vxCreateDistribution (vx_context context, vx_size numBins, vx_size offset, vx_size range)
  Creates a reference to a 1D Distribution with a start offset, valid range, and number of equally weighted bins.
- vx_status vxQueryDistribution (vx_distribution distribution, vx_enum attribute, void *ptr, vx_size size)
  Queries a Distribution object.
- void 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.

3.50.2 Enumeration Type Documentation

enum vx_distribution_attribute_e

The distribution attribute list.

Enumerator

- VX_DISTRIBUTION_ATTRIBUTE_DIMENSIONS Indicates the number of dimensions in the distribution. Use a vx_size parameter.
- VX_DISTRIBUTION_ATTRIBUTE_OFFSET Indicates the start of the values to use (inclusive). Use a vx_uint32 parameter.
- VX_DISTRIBUTION_ATTRIBUTE_RANGE Indicates end value to use as the range (exclusive). Use a vx_uint32 parameter.
**VX.DISTRIBUTION_ATTRIBUTE_BINS** Indicates the number of bins. Use a `vx.uint32` parameter.

**VX.DISTRIBUTION_ATTRIBUTE_WINDOW** Indicates the range of a bin. Use a `vx.uint32` parameter.

**VX.DISTRIBUTION_ATTRIBUTE_SIZE** The total size of the distribution in bytes. Use a `vx.size` parameter.

Definition at line 814 of file `vx_types.h`.

### 3.50.3 Function Documentation

**vx_status vxAccessDistribution ( vx.distribution distribution, void ** ptr, vx_enum usage )**

Gets direct access to a Distribution in memory.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>distribution</th>
<th>The reference to the distribution to access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>ptr</td>
<td>The address of the location to store the pointer to the Distribution memory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If (*ptr) is not NULL, the Distribution will be copied to that address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If (*ptr) is NULL, the pointer will be allocated, mapped, or use internal memory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In any case, <code>vxCommitDistribution</code> must be called with (*ptr).</td>
</tr>
<tr>
<td>in</td>
<td>usage</td>
<td>The <code>vx_accessor_e</code> value to describe the access of the object.</td>
</tr>
</tbody>
</table>

**Returns**

A `vx_status_e` enumeration.

**Postcondition**

`vxCommitDistribution`

**vx_status vxCommitDistribution ( vx.distribution distribution, void * ptr )**

Sets the Distribution back to the memory. The memory is the array must be a `vx.uint32` array of a value at least as big as the value returned via `VX.DISTRIBUTION_ATTRIBUTE_RANGE`.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>distribution</th>
<th>The Distribution to modify.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ptr</td>
<td>The pointer returned from (or not modified by) <code>vxAccessDistribution</code>.</td>
</tr>
</tbody>
</table>

**Returns**

A `vx_status_e` enumeration.

**Precondition**

`vxAccessDistribution`.

**vx_distribution vxCreateDistribution ( vx.context context, vx.size numBins, vx.size offset, vx.size range )**

Creates a reference to a 1D Distribution with a start offset, valid range, and number of equally weighted bins.

**Parameters**

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

---

**Note:** The text above is a natural representation of the document content. It has been formatted to improve readability and coherence, but the original structure and context have been maintained. The code examples and function descriptions are presented as they appear in the document, ensuring they are accurately transcribed.
<table>
<thead>
<tr>
<th>in</th>
<th>numBins</th>
<th>The number of bins in the distribution.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>offset</td>
<td>The offset into the range value.</td>
</tr>
<tr>
<td>in</td>
<td>range</td>
<td>The total range of the values.</td>
</tr>
</tbody>
</table>

Returns

\[ \text{vx.distribution} \]

\textbf{vx\_status vxQueryDistribution ( vx\_distribution \textit{distribution}, vx\_enum \textit{attribute}, void * \textit{ptr}, vx\_size \textit{size} )}

Queries a Distribution object.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>distribution</th>
<th>The reference to the distribution to query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>attribute</td>
<td>The attribute to query. Use a \text{vx_distribution_attribute_e} enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The location at which the resulting value will be stored.</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>The size of the container to which \textit{ptr} points.</td>
</tr>
</tbody>
</table>

Returns

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

\textbf{void vxReleaseDistribution ( vx\_distribution * \textit{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. |

Note

After returning from this function the reference will be zeroed.
3.51 Object: Image

3.51.1 Detailed Description

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...

Macros

- **#define VX_IMAGEPATCH_ADDRINIT {0u, 0u, 0, 0u, 0u, 0u, 0u, 0u}**
  
  Used to initialize a `vx_imagepatch_addressing_t` structure on the stack.

Typedefs

- **typedef struct vx_image ∗vx_image**
  
  An opaque reference to an image.

Enumerations

- **enum vx_channelrange_e**
  
  The image channel range list used by the `VX_IMAGE_ATTRIBUTE_RANGE` attribute of a `vx_image`.

- **enum vx_colorspace_e**
  
  The image color space list used by the `VX_IMAGE_ATTRIBUTE_SPACE` attribute of a `vx_image`.

- **enum vx_imageattribute_e**
  
  The image attributes list.

Functions

- **vx_status vxAccessImagePatch (vx_image image, vx_rectangle_t ∗rect, vx_uint32 plane_index, vx_imagepatch_addressing_t ∗addr, void ∗∗ptr, vx_enum usage)**
  
  This allows the User to extract a rectangular patch (subset) of an image from a single plane.

- **vx_status vxCommitImagePatch (vx_image image, vx_rectangle_t ∗rect, vx_uint32 plane_index, vx_imagepatch_addressing_t ∗addr, void ∗ptr)**
  
  This allows the User to commit a rectangular patch (subset) of an image from a single plane.
• **vx_size vxComputeImagePatchSize (vx_image image, vx_rectangle_t *rect, vx_uint32 plane_index)**
  
  This computes the size needed to retrieve an image patch from an image.

• **vx_image vxCreateImage (vx_context context, vx_uint32 width, vx_uint32 height, vx_fourcc color)**
  
  Creates an opaque reference to an image buffer.

• **vx_image vxCreateImageFromHandle (vx_context context, vx_fourcc color, vx_imagepatch_addressing_t addr[], void *ptr[], vx_enum import_type)**
  
  Creates a reference to an image object which was externally allocated.

• **vx_image vxCreateImageFromROI (vx_image img, 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 will update the parent image. The rectangle must be defined within the pixel space of the parent image.

• **vx_image vxCreateUniformImage (vx_context context, vx_uint32 width, vx_uint32 height, vx_fourcc color, void *value)**
  
  Creates an reference to an image object which has a singular, uniform value in all pixels.

• **vx_image vxCreateVirtualImage (vx_graph graph, vx_uint32 width, vx_uint32 height, vx_fourcc 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 *vxFormatImagePatchAddress1d (void *ptr, vx_uint32 index, vx_imagepatch_addressing_t *addr)**
  
  Used to access a specific indexed pixel in an image patch.

• **void *vxFormatImagePatchAddress2d (void *ptr, vx_uint32 x, vx_uint32 y, vx_imagepatch_addressing_t *addr)**
  
  Used to access a specific pixel at a 2d coordinate in an image patch.

• **vx_status vxGetValidRegionImage (vx_image image, vx_rectangle_t *rect)**
  
  Retrieves the valid region of the image as a rectangle.

• **vx_status vxQueryImage (vx_image image, vx_enum attribute, void *ptr, vx_size size)**
  
  Retrieves various attributes of an image.

• **void 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 vxSetImageAttribute (vx_image image, vx_enum attribute, void *out, vx_size size)**
  
  Allows setting attributes on the image.

### 3.51.2 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 \( \text{scale} = \frac{\text{unity}}{f} \) and an integer up-scaling factor of \( f \) shall be set to a value of \( \text{scale} = \text{unity} \times f \). unity is defined as \( \text{VX SCALE UNITY} \).

- **step** - The step is the number of logical pixel units to skip in order to arrive at the next physically unique pixel. For example, on a plane which 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

vxAccessImagePatch

/*
 * Copyright (c) 2012-2013 The Khronos Group Inc.
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and/or associated documentation files (the
 * "Materials"), to deal in the Materials without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Materials, and to
 * permit persons to whom the Materials are furnished to do so, subject to
 * the following conditions:
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Materials.
 * THE MATERIALS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS.
 */
#include <VX/vx.h>
#define PATCH_DIM 16
vx_status example_imagepatch(vx_context context) {
    vx_status status = VX_SUCCESS;
    void *base_ptr = NULL;
    vx_uint32 width = 640, height = 480, plane = 0;
    vx_image image = vxCreateImage(context, width, height, FOURCC_U8);
    vx_rectangle_t rect;
    vx_imagepatch_addressing_t addr;
    rect.start_x = rect.start_y = 0;
    rect.end_x = rect.end_y = PATCH_DIM;
    status = vxAccessImagePatch(image, &rect, plane, &addr, &base_ptr, VX_READ_AND_WRITE);
    if (status == VX_SUCCESS) {
        vx_uint32 x,y,i,j;
        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, &addr);
            *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, &addr);
                *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 = ((addr.stride_y*y*addr.scale_y) / VX_SCALE_UNITY) +
                    ((addr.stride_x*x*addr.scale_x) / VX_SCALE_UNITY);
                tmp[i] = pixel;
            }
        }
        */
    }
}

*/
\[ j = \frac{(addr.\text{stride}_y \times \text{y} \times addr.\text{scale}_y)}{\text{VX\_SCALE\_UNITY}}; \]
for (x = 0; x < addr.\text{dim}_x; x+=addr.\text{step}_x) {
\text{vx\_uint8 *tmp = (vx\_uint8 *)base\_ptr;}
\text{i = j + (addr.\text{stride}_x \times \text{x} \times addr.\text{scale}_x) / VX\_SCALE\_UNITY;}
\text{tmp[i] = pixel;}
}
/* this commits the data back to the image. If rect were 0 or empty, it */
/* would just decrement the reference (used when reading an image only) */
\text{status = vxCommitImagePatch(image, &rect, plane, &addr, base\_ptr);} 
\text{vxReleaseImage(ximage); return status;}

Definition at line 1263 of file \textit{vx\_types.h}.

Data Fields

| \text{vx\_uint32} | \text{dim}_x | Width of patch in X dimension in pixels. |
| \text{vx\_uint32} | \text{dim}_y | Height of patch in Y dimension in pixels. |
| \text{vx\_uint32} | \text{scale}_x | Scale of X dimension. For sub-sampled planes this will be the scaling factor of the dimension of the plane in relation to the zero plane. Used \text{VX\_SCALE\_UNITY} in the numerator. |
| \text{vx\_uint32} | \text{scale}_y | Scale of Y dimension. For sub-sampled planes this will be the scaling factor of the dimension of the plane in relation to the zero plane. Used \text{VX\_SCALE\_UNITY} in the numerator. |
| \text{vx\_uint32} | \text{step}_x | Step of X dimension in pixels. |
| \text{vx\_uint32} | \text{step}_y | Step of Y dimension in pixels. |
| \text{vx\_int32} | \text{stride}_x | Stride in X dimension in bytes. |
| \text{vx\_int32} | \text{stride}_y | Stride in Y dimension in bytes. |

3.51.3 Typedef Documentation

typedef struct \_vx\_image* \textit{vx\_image}

An opaque reference to an image.

See Also

\texttt{vxCreateImage}

Definition at line 144 of file \textit{vx\_types.h}.

3.51.4 Enumeration Type Documentation

\texttt{enum \_vx\_channel\_range\_e}

The image channel range list used by the \texttt{VX\_IMAGE\_ATTRIBUTE\_RANGE} attribute of a \texttt{vx\_image}.

Enumerator

\texttt{VX\_CHANNEL\_RANGE\_FULL} Full range of the unit of the channel.

\texttt{VX\_CHANNEL\_RANGE\_RESTRICTED} Restricted range of the unit of the channel based on the space given.

Definition at line 1026 of file \textit{vx\_types.h}.

\texttt{enum \_vx\_color\_space\_e}

The image color space list used by the \texttt{VX\_IMAGE\_ATTRIBUTE\_SPACE} attribute of a \texttt{vx\_image}.

Enumerator

\texttt{VX\_COLOR\_SPACE\_NONE} Used to indicate that no color space is used.
CHAPTER 3. MODULE DOCUMENTATION

**VX.COLOR_SPACE_BT601.525** Used to indicate that the BT.601 coefficients and SMPTE C primaries are used for conversions.

**VX.COLOR_SPACE_BT601.625** Used to indicate that the BT.601 coefficients and BTU primaries are used for conversions.

**VX.COLOR_SPACE_BT709** Used 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 1009 of file vx_types.h.

**enum vx.image_attribute_e**

The image attributes list.

**Enumerator**

**VX.IMAGE_ATTRIBUTE_WIDTH** Used to query an image for its height. Use a **vx.uint32** parameter.

**VX.IMAGE_ATTRIBUTE_HEIGHT** Used to query an image for its width. Use a **vx.uint32** parameter.

**VX.IMAGE_ATTRIBUTE_FORMAT** Used to query an image for its format. Use a **vx.fourcc** parameter.

**VX.IMAGE_ATTRIBUTE_PLANES** Used to query an image for its number of planes. Use a **vx.size** parameter.

**VX.IMAGE_ATTRIBUTE_SPACE** Used to query an image for its color space (see **vx.color_space_e**). Use a **vx.enum** parameter.

**VX.IMAGE_ATTRIBUTE_RANGE** Used to query an image for its channel range (see **vx.channel_range_e**). Use a **vx.enum** parameter.

**VX.IMAGE_ATTRIBUTE_SIZE** Used to query an image for its total number of bytes. Use a **vx.size** parameter.

Definition at line 760 of file vx_types.h.

### 3.51.5 Function Documentation

**vx_status vxAccessImagePatch ( vx.image image, vx.rectangle_t * rect, vx.uint32 plane_index, vx.imagepatch_addressing_t * addr, void ** ptr, vx.enum usage )**

This allows the User to extract a rectangular patch (subset) of an image from a single plane.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>image</th>
<th>The reference to the image to extract the patch from.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rect</td>
<td>The coordinates to get the patch from. Must be 0 &lt;= start &lt; end.</td>
</tr>
<tr>
<td>in</td>
<td>plane_index</td>
<td>The plane index to get the data from.</td>
</tr>
<tr>
<td>out</td>
<td>addr</td>
<td>The addressing information for the image patch will be written into the data structure.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The pointer to a pointer of a location to store the data.</td>
</tr>
</tbody>
</table>

- If the user passes in a NULL, an error occurs.
- If the user passes in a pointer to a NULL, the function will return internal memory, map, or allocate a buffer and return it.
- If the user passes in a pointer to a non-NULL pointer, the function will attempt to copy to the location provided by the user.

**(ptr)** must be given to **vxCommitImagePatch**.
Returns

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

Return values

| \texttt{VX\_ERROR\_OPTIMIZED\_AWAY} | The image is not present in memory. |
| \texttt{VX\_ERROR\_INVALID\_PARAMETERS} | The start, end, plane index, stride\_x or stride\_y pointer was incorrect. |
| \texttt{VX\_ERROR\_INVALID\_REFERENCE} | The image reference was not actually an image reference. |

Note

The user may ask for data outside the bounds of the valid region, but such data has an undefined value. Users must be cautious to prevent passing in \texttt{uninitialized} pointers or addresses of uninitialized pointers to this function.

Precondition

\texttt{vx\_Compute\_Image\_Patch\_Size} if users wish to allocate their own memory.

Postcondition

\texttt{vx\_Commit\_Image\_Patch} with same (*ptr) value.

/*
 * Copyright (c) 2012-2013 The Khronos Group Inc.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and/or associated documentation files (the
 * "Materials"), to deal in the Materials without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Materials, and to
 * permit persons to whom the Materials are furnished to do so, subject to
 * the following conditions:
 * 
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Materials.
 * 
 * THE MATERIALS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS.
 */
#include <VX/vx.h>
#define PATCH\_DIM 16

\texttt{vx\_status example\_imagepatch(vx\_context context)}
{
    \texttt{vx\_status status = VX\_SUCCESS;}
    void *base\_ptr = NULL;
    \texttt{vx\_uint32 width = 640, height = 480, plane = 0;}
    \texttt{vx\_image image = vx\_Create\_Image(context, width, height, FOURCC\_U8);} 
    \texttt{vx\_rectangle\_t rect;}
    \texttt{vx\_imagepatch\_addressing\_t addr;}
    \texttt{rect\_start\_x = rect\_start\_y = 0;}
    \texttt{rect\_end\_x = rect\_end\_y = PATCH\_DIM;}
    \texttt{status = vx\_Access\_Image\_Patch(image, \&rect, plane,}
    \texttt{\&addr, \&base\_ptr, VX\_READ\_AND\_WRITE);} 
    \texttt{if (status == VX\_SUCCESS)}
    \{
        \texttt{vx\_uint32 x,y,i,j;}
        \texttt{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, &addr);
    *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, &addr);
        *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) / VXSCALEUNITY) +
            ((addr.stride_x*x*addr.scale_x) / VXSCALEUNITY);
        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_x*x*addr.scale_x) / VXSCALEUNITY;
        tmp[i] = pixel;
    }
}

/* this commits the data back to the image. If rect were 0 or empty, it */
/* would just decrement the reference (used when reading an image only) */
vx_status vxCommitImagePatch (vx_image image, vx_rectangle_t *rect, vx_uint32 plane_index,
    vx_imagepatch_addressing_t *addr, void *ptr)

This allows the User to commit a rectangular patch (subset) of an image from a single plane.

Parameters

| in | image          | The reference to the image to extract the patch from. |
| in | rect           | The coordinates to set the patch to. Must be 0 <= start <= end. This may be |
|    |                | 0 or a rectangle of zero area in order to indicate that the commit should only |
|    |                | decrement the reference count. |
| in | plane_index    | The plane index to set the data to. |
| in | addr           | The addressing information for the image patch. |
| in | ptr            | The pointer of a location to read the data from. If the user allocated the pointer |
|    |                | they must free it. If the pointer was set by vxAccessImagePatch, the user |
|    |                | may not access the pointer after this call is complete. |

Returns

A vx_status_e enumeration.
Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_OPTIMIZED_AWAY</td>
<td>The image is not present in memory.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>The start, end, plane index, stride_x or stride_y pointer was incorrect.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>The image reference was not actually an image reference.</td>
</tr>
</tbody>
</table>

/*
 * Copyright (c) 2012-2013 The Khronos Group Inc.
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and/or associated documentation files (the
 * "Materials"), to deal in the Materials without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Materials, and to
 * permit persons to whom the Materials are furnished to do so, subject to
 * the following conditions:
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Materials.
 * THE MATERIALS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS.
 */
#include <VX/vx.h>
#define PATCH_DIM 16

xv_status example_imagepatch(vx_context context)
{
    xv_status status = VX_SUCCESS;
    void *base_ptr = NULL;
    xv_int32 width = 640, height = 480, plane = 0;
    xv_image image = vxCreateImage(context, width, height,
        FOURCC_U8);
    xv_rectangle_t rect;
    xv_imagepatch_addressing_t addr;
    rect.start_x = rect.start_y = 0;
    rect.end_x = rect.end_y = PATCH_DIM;
    status = vxAccessImagePatch(image, &rect, plane,
        &addr, &base_ptr, VX_READANDWRITE);
    if (status == VX_SUCCESS)
    {
        xv_uint32 x,y,i,j;
        xv_uint8 pixel = 0;
        /* a couple addressing options */
        /* use linear addressing function/macro */
        for (i = 0; i < addr.dim_x*addr.dim_y; i++)
        {
            xv_uint8 *ptr2 = vxFormatImagePatchAddress1d(base_ptr,
                i, &addr);
            *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) {
                xv_uint8 *ptr2 = vxFormatImagePatchAddress2d(base_ptr,
                    x, y, &addr);
                *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) {
                xv_uint8 *tmp = (vuint8 *)base_ptr;
                i = ((addr.stride_y*y*addr.scale_y) /
                    VX_SCALEUNITY) + ((addr.stride_x*x*addr.scale_x) /
                    VX_SCALEUNITY);
```c
    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) {
  j = (addr.stride_y*y*addr.scale_y)/vx_SCALE_UNITY;
  for (x = 0; x < addr.dim_x; x+=addr.step_x) {
    i = j + (addr.stride_x*x*addr.scale_x) / vx_SCALE_UNITY;
    tmp[i] = pixel;
  }
}

/* this commits the data back to the image. If rect were 0 or empty, it
 * would just decrement the reference (used when reading an image only)
 */
status = vxCommitImagePatch(image, &rect, plane, &addr, base_ptr);
vxReleaseImage(image);
return status;
```

**Note**

If the implementation gave the client a pointer from `vxAccessImagePatch` then implementation specific behavior may occur. If not, then a copy occurs from the users pointer to the internal data of the object.

If the rectangle is intersects bounds of the current valid region, the valid region grows to the union of the two rectangles as long as they occur within the bounds of the original image dimensions.

### vx_size vxComputeImagePatchSize ( vx_image, vx_rectangle_t * rect, vx_uint32 plane_index )

This computes the size needed to retrieve an image patch from an image.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>image</th>
<th>The reference to the image to extract the patch from.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rect</td>
<td>The coordinates. Must be 0 &lt;= start &lt;= end &lt;= dimension where dimension is width for x and height for y.</td>
</tr>
<tr>
<td>in</td>
<td>plane_index</td>
<td>The plane index to get the data from.</td>
</tr>
</tbody>
</table>

**Returns**

vx_size

### vx_image vxCreateImage ( vx_context context, vx_uint32 width, vx_uint32 height, vx_fourcc 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>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.</td>
</tr>
<tr>
<td>in</td>
<td>height</td>
<td>The image height in pixels.</td>
</tr>
<tr>
<td>in</td>
<td>color</td>
<td>The FOURCC (vx_fourcc_e) code which represents the format of the image and the color space.</td>
</tr>
</tbody>
</table>

**Returns**

Returns an image reference or zero when an error is encountered.

**See Also**

`vxAccessImagePatch` to obtain direct memory access to the image data.
vx_image vxCreateImageFromHandle ( vx_context context, vx_fourcc color, vx_imagepatch_addressing_t addrs[], void *ptrs[], vx_enum import_type )

Creates a reference to an image object which was externally allocated.
Parameters

| in | context | The reference to the implementation context. |
| in | color | See the vx_fourcc_t codes. This mandates the number planes needed to be valid in the addr and ptrs arrays based on the format given. |
| in | addr[] | The array of image patch addressing structures which defines the dimension and stride of the array of pointers. |
| in | ptrs[] | The array of platform defined references to each plane. |
| in | import_type | vx_import_type_e. When giving VX_IMPORT_TYPE_HOST the ptrs[] is assumed to be HOST accessible pointers to memory. |

Returns

Returns vx_image.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Image could not be created.</td>
</tr>
<tr>
<td>*</td>
<td>Valid Image reference.</td>
</tr>
</tbody>
</table>

vx_image vxCreateImageFromROI ( vx_image img, 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 will update the parent image. The rectangle must be defined within the pixel space of the parent image.

Parameters

| in | img | The reference to the parent image. |
| in | rect | The region of interest rectangle. Must contain points within the parent image pixel space. |

Returns

Returns the reference to the sub-image or zero if the rectangle was invalid.

vx_image vxCreateUniformImage ( vx_context context, vx_uint32 width, vx_uint32 height, vx_fourcc color, void * value )

Creates an reference to an image object which has a singular, uniform value in all pixels.

The value pointer must reflect the specific format of the desired image. For example:

<table>
<thead>
<tr>
<th>Color</th>
<th>Value Ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOURCC_U8</td>
<td>vx_uint8 *</td>
</tr>
<tr>
<td>FOURCC_S16</td>
<td>vx_int16 *</td>
</tr>
<tr>
<td>FOURCC_U16</td>
<td>vx_uint16 *</td>
</tr>
<tr>
<td>FOURCC_S32</td>
<td>vx_int32 *</td>
</tr>
<tr>
<td>FOURCC_U32</td>
<td>vx_uint32 *</td>
</tr>
<tr>
<td>FOURCC_RGBX</td>
<td>vx_uint8 pixels[4]</td>
</tr>
<tr>
<td>Any YUV</td>
<td>vx_uint8 pixel[3] in Y, U, V order</td>
</tr>
</tbody>
</table>

Parameters

| in | context | The reference to the implementation context. |
| in | width | The image width in pixels. |
| in | height | The image height in pixels. |
| in | color | The FOURCC (vx_fourcc_t) code which represents the format of the image and the color space. |
in | value | The pointer to the pixel value to set all pixels to.

Returns

Returns an image reference or zero when an error is encountered.

See Also

vxAccessImagePatch to obtain direct memory access to the image data.

Note

vxAccessImagePatch and vxCommitImagePatch may be called with a uniform image reference.

vx_image vxCreateVirtualImage ( vx_graph graph, vx_uint32 width, vx_uint32 height, vx_fourcc 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 is 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.

vx_image virt[] = {
    vxCreateVirtualImage(graph, 0, 0, FOURCC_U8), // no specified dimension
    vxCreateVirtualImage(graph, 320, 240, FOURCC_VIRT), // no specified format
    vxCreateVirtualImage(graph, 640, 480, FOURCC_U8), // no user access
};

Parameters

| in | graph | The reference to the parent graph. |
|    | width | The width of the image in pixels. A value of zero informs the interface that the value is unspecified. |
|    | height | The height of the image in pixels. A value of zero informs the interface that the value is unspecified. |
|    | color | The FOURCC (vx_fourcc_t) code which represents the format of the image and the color space. A value of FOURCC_VIRT informs the interface that the format is unspecified. |

Returns

Returns an image reference or zero when an error is encountered.

Note

Passing this reference to vxAccessImagePatch will return an error.

void* vxFormatImagePatchAddress1d ( void *ptr, vx_uint32 index, vx_imagepatch_addressing_t *addr )

Used to access 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 <code>vxAccessImagePatch</code>.</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 <code>vxAccessImagePatch</code>.</td>
</tr>
</tbody>
</table>

### Returns

```c
void * Returns the pointer to the specified pixel.
```

### Precondition

```c
vxAccessImagePatch
```

/*
 * Copyright (c) 2012-2013 The Khronos Group Inc.
 */

/* Permission is hereby granted, free of charge, to any person obtaining a */
/* copy of this software and/or associated documentation files (the */
/* "Materials"), to deal in the Materials without restriction, including */
/* without limitation the rights to use, copy, modify, merge, publish, */
/* distribute, sublicense, and/or sell copies of the Materials, and to */
/* permit persons to whom the Materials are furnished to do so, subject to */
/* the following conditions: */
/* */
/* * The above copyright notice and this permission notice shall be included */
/* * in all copies or substantial portions of the Materials. */
/* */
/* * THE MATERIALS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, */
/* * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF */
/* * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. */
/* * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY */
/* * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, */
/* * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE */
/* * MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS. */
/* */

/* */

#include <VX/vx.h>

#define PATCH_DIM 16

vx_status example_imagepatch(vx_context context) {
    vx_status status = VX_SUCCESS;
    void *base_ptr = NULL;
    vx_uint32 width = 640, height = 480, plane = 0;
    vx_image image = vxCreateImage(context, width, height, FOURCC_U8);
    vx_rectangle_t rect;
    vx_imagepatch_addressing_t addr;
    rect.start.x = rect.start.y = 0;
    rect.end.x = rect.end.y = PATCH_DIM;
    status = vxAccessImagePatch(image, &rect, plane, &addr, &base_ptr, VX_READ_AND_WRITE);
    if (status == VX_SUCCESS) {
        vx_uint32 x, y, i, j;
        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, &addr);
            *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, &addr);
                *ptr2 = pixel;
            }
        }

        /* direct addressing by client
        */
Chapter 3. Module Documentation

* for subsampled planes, scale will change

```c
* 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. */
```c
* for (y = 0; y < addr.dim_y; y += addr.step_y)
  j = (addr.stride_y*y*addr.scale_y)/VX_SCALE_UNITY;
  for (x = 0; x < addr.dim_x; x += addr.step_x)
    vx_uint8 *tmp = (vx_uint8 *)base_ptr;
    i = j + (addr.stride_x*x*addr.scale_x) / VX_SCALE_UNITY;
    tmp[i] = pixel;
```

/* this commits the data back to the image. If rect were 0 or empty, it * would just decrement the reference (used when reading an image only) */
```c
status = vxCommitImagePatch(image, &rect, plane, &addr, base_ptr);
vxReleaseImage(image);
return status;
```

```c
void vxFormatImagePatchAddress2d ( void *ptr, vx_uint32 x, vx_uint32 y, vx_imagepatch_addressing_t *addr )
```

Used to access 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 <code>vxAccessImagePatch</code>.</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 <code>vxAccessImagePatch</code>.</td>
</tr>
</tbody>
</table>

**Returns**

```c
void * Returns the pointer to the specified pixel.
```

**Precondition**

```c
vxAccessImagePatch
```

/* Copyright (c) 2012-2013 The Khronos Group Inc.*

* Permission is hereby granted, free of charge, to any person obtaining a *
  copy of this software and/or associated documentation files (the *
  "Materials"), to deal in the Materials without restriction, including *
  without limitation the rights to use, copy, modify, merge, publish, *
  distribute, sublicense, and/or sell copies of the Materials, and to *
  permit persons to whom the Materials are furnished to do so, subject to *
  the following conditions:
*
* The above copyright notice and this permission notice shall be included *
  in all copies or substantial portions of the Materials.
*
* THE MATERIALS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, *
  EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF *
  MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. *
  IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY *
  CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, *
  TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE *
  MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS.
/*
#include <VX/vx.h>
#define PATCH_DIM 16

vx_status example_imagepatch(vx_context context)
{
  vx_status status = VX_SUCCESS;
  void *base_ptr = NULL;
  vx_uint32 width = 640, height = 480, plane = 0;
  vx_image image = vxCreateImage(context, width, height, FOURCC_U8);
  vx_rectangle_t rect;
  vx_imagepatch_addressing_t addr;

  rect.start.x = rect.start.y = 0;
  rect.end.x = rect.end.y = PATCH_DIM;

  status = vxAccessImagePatch(image, &rect, plane,
                               &addr, &base_ptr, VX_READ_AND_WRITE);
  if (status == VX_SUCCESS)
  {
    vx_uint32 x, y, i, j;
    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, &addr);
      *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, &addr);
        *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.stride.x*x) /
            VX_SCALEUNITY;
        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)
    {
      j = (addr.stride.y*y + addr.stride.x*x) /
          VX_SCALEUNITY;
      for (x = 0; x < addr.dim.x; x+=addr.step.x)
      {
        vx_uint8 *tmp = (vx_uint8 *)base_ptr;
        i = j + (addr.stride.x*x + addr.stride.y*y) /
            VX_SCALEUNITY;
        tmp[i] = pixel;
      }
    }

    /* this commits the data back to the image. If rect were 0 or empty, it
     * would just decrement the reference (used when reading an image only)*/
    status = vxCommitImagePatch(image, &rect, plane, &addr, base_ptr);
  }
  vxReleaseImage(&image);
  return status;
}

vx_status vxGetValidRegionImage ( vx_image image, vx_rectangle_t * rect )

Retrieves the valid region of the image as a rectangle.
After the image is allocated but has not been written to this will return the full rectangle of the image so that functions do not have to manage a case for uninitialized data. The image will still retain an uninitialized value but once the image is written to via any means such as `vxCommitImagePatch`, the valid region will be altered to contain the maximum bounds of the written area.

**Parameters**

<table>
<thead>
<tr>
<th>Role</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>image</td>
<td>The image to retrieve the valid region from.</td>
</tr>
<tr>
<td>out</td>
<td>rect</td>
<td>The destination rectangle.</td>
</tr>
</tbody>
</table>

**Returns**

`vx_status`

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>VX_ERROR_INVALID_REFERENCE</code></td>
<td>Invalid image.</td>
</tr>
<tr>
<td><code>VX_ERROR_INVALID_PARAMETERS</code></td>
<td>Invalid rect.</td>
</tr>
<tr>
<td><code>VX_SUCCESS</code></td>
<td>Valid image.</td>
</tr>
</tbody>
</table>

**Note**

This rectangle can be passed directly to `vxAccessImagePatch` to get the full valid region of the image. Modifications from `vxCommitImagePatch` will grow the valid region.

```c
void 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.

**Parameters**

<table>
<thead>
<tr>
<th>Role</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>image</td>
<td>The reference to the image to query.</td>
</tr>
<tr>
<td>in</td>
<td>attribute</td>
<td>The attribute to query. Use a <code>vx_image_attribute_t</code>.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The location at which the resulting value will be stored.</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_t` enumeration.

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>VX_SUCCESS</code></td>
<td>No errors</td>
</tr>
<tr>
<td><code>VX_ERROR_INVALID_REFERENCE</code></td>
<td>if the image is not a <code>vx_image</code>.</td>
</tr>
<tr>
<td><code>VX_ERROR_INVALID_PARAMETERS</code></td>
<td>if any of the other parameters are incorrect.</td>
</tr>
<tr>
<td><code>VX_ERROR_NOT_SUPPORTED</code></td>
<td>if the attribute is not supported on this implementation.</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

The pointer to the image to release

Note

After returning from this function the reference will be zeroed.

vx_status vxSetImageAttribute ( vx_image image, vx_enum attribute, void * out, vx_size size )

Allows setting attributes on the image.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>image</td>
<td>The pointer to the image to release.</td>
</tr>
<tr>
<td>in</td>
<td>attribute</td>
<td>The attribute to set. Use a vx_image_attribute_e enumeration.</td>
</tr>
<tr>
<td>in</td>
<td>out</td>
<td>The pointer to the where the value will be read from.</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>The size of the object pointed to by out.</td>
</tr>
</tbody>
</table>

Returns

A vx_status_e enumeration.

Return values

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No errors</td>
<td></td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>if the image is not a vx_image.</td>
<td></td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>if any of the other parameters are incorrect.</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 3. Module Documentation

3.52 Object: LUT

3.52.1 Detailed Description

The Look-Up Table Interface. A lookup table is an array that simplifies runtime 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`
  The LUT attribute list.
  
  - `VX_LUT_ATTRIBUTE_TYPE` Indicates the value type of the LUT. Use a `vx_enum`.
  - `VX_LUT_ATTRIBUTE_COUNT` Indicates the number of elements in the LUT. Use a `vx_size`.
  - `VX_LUT_ATTRIBUTE_SIZE` The total size of the LUT in bytes. Uses a `vx_size`.

Definition at line 802 of file `vx_types.h`.

Functions

- `vx_status vxAccessLUT (vx_lut lut, void **ptr, vx_enum usage)`  
  Gets direct access to the LUT table data.
  
  There are several variations of call methodology:
  
  - If `ptr` is NULL (which means the current data of the LUT is not desired), the LUT reference count is incremented.

3.52.2 Enumeration Type Documentation

`enum vx_lut_attribute_e`

The LUT attribute list.

Enumerator

- `VX_LUT_ATTRIBUTE_TYPE` Indicates the value type of the LUT. Use a `vx_enum`.
- `VX_LUT_ATTRIBUTE_COUNT` Indicates the number of elements in the LUT. Use a `vx_size`.
- `VX_LUT_ATTRIBUTE_SIZE` The total size of the LUT in bytes. Uses a `vx_size`.

Definition at line 802 of file `vx_types.h`.

3.52.3 Function Documentation

`vx_status vxAccessLUT ( vx_lut lut, void **ptr, vx_enum usage )`

Gets direct access to the LUT table data.

There are several variations of call methodology:

- If `ptr` is NULL (which means the current data of the LUT is not desired), the LUT reference count is incremented.
CHAPTER 3. MODULE DOCUMENTATION

• If ptr is not NULL but (∗ptr) is NULL, (∗ptr) will contain the address of the LUT data when the function returns and the reference count will be incremented. Whether the (∗ptr) address is mapped or allocated is undefined. (∗ptr) must be returned to vxCommitLUT.

• If ptr is not NULL and (∗ptr) is not NULL, the user is signalling the implementation to copy the LUT data into the location specified by (∗ptr). Users must use vxQueryLUT with VX_LUT_ATTRIBUTE_SIZE to determine how much memory to allocate for the LUT data.

In any case, vxCommitLUT must be called after LUT access is complete.

Parameters

| in | lut | The LUT to get the data from. |
| in,out | ptr | The address of the location to store the pointer to the LUT memory. |
| in | usage | This declares the intended usage of the pointer using the ∗vx_accessor_e enumeration. |

Returns

A vx_status_e enumeration.

Postcondition

vxCommitLUT

vx_status vxCommitLUT ( vx_lut lut, void * ptr )

Commits the Lookup Table.

Commits the data back to the LUT object and decrements the reference count. There are several variations of call methodology:

• If a user should allocated their own memory for the LUT data copy, the user is obligated to free this memory.

• If ptr is not NULL and the (∗ptr) for vxAccessLUT was NULL, it is undefined whether the implementation will unmap or copy and free the memory.

Parameters

| in | lut | The LUT to modify. |
| in | ptr | The pointer used with vxAccessLUT. This may not be NULL. |

Returns

A vx_status_e enumeration.

Precondition

vxAccessLUT.

vx_lut vxCreateLUT ( vx_context context, vx_enum data_type, vx_size count )

Creates LUT object of a given type.

Parameters

| in | context | The reference to the context. |
| in | data_type | The type of data stored in the LUT. |
| in | count | The number of entries desired. |

Note

For OpenVX 1.0, count must be equal to 256 and data_type can only be VX_TYPE_UINT8.

Returns

vx_lut
vx_status vxQueryLUT ( vx_lut lut, vx_enum attribute, void * ptr, vx_size size )

Queries attributes from a LUT.
CHAPTER 3. MODULE DOCUMENTATION

Parameters

| in | lut | The LUT to query. |
| in | attribute | The attribute to query. Use a vx_lut_attribute_e enumeration. |
| out | ptr | The location at which the resulting value will be stored. |
| in | size | The size of the container to which ptr points. |

Returns

A vx_status_e enumeration.

void vxReleaseLUT ( vx_lut * lut )

Release 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. |

Note

After returning from this function the reference will be zeroed.
3.53 Object: Matrix

3.53.1 Detailed Description

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_ATTRIBUTE_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_MATRIX << 8)) + 0x0,
  VX_MATRIX_ATTRIBUTE_ROWS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_MATRIX << 8)) + 0x1,
  VX_MATRIX_ATTRIBUTE_COLUMNS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_MATRIX << 8)) + 0x2,
  VX_MATRIX_ATTRIBUTE_SIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_MATRIX << 8)) + 0x3
  
  The matrix attributes.

Functions

- vx_status vxAccessMatrix (vx_matrix mat, void *array)
  
  Gets the matrix data (copy)

- vx_status vxCommitMatrix (vx_matrix mat, void *array)
  
  Sets the matrix data (copy)

- vx_matrix vxCreateMatrix (vx_context c, vx_enum data_type, vx_size columns, vx_size rows)
  
  Creates a reference to a matrix object.

- vx_status vxQueryMatrix (vx_matrix mat, vx_enum attribute, void *ptr, vx_size size)
  
  Queries an attribute on the matrix object.

- void 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.

3.53.2 Enumeration Type Documentation

enum vx_matrix_attribute_e

The matrix attributes.

Enumerator

- VX_MATRIX_ATTRIBUTE_TYPE The value type of the matrix. Use a vx_enum parameter.
- VX_MATRIX_ATTRIBUTE_ROWS The M dimension of the matrix. Use a vx_size parameter.
- VX_MATRIX_ATTRIBUTE_COLUMNS The N dimension of the matrix. Use a vx_size parameter.
- VX_MATRIX_ATTRIBUTE_SIZE The total size of the matrix in bytes. Use a vx_size parameter.

Definition at line 856 of file vx_types.h.

3.53.3 Function Documentation

vx_status vxAccessMatrix ( vx_matrix mat, void *array )

Gets the matrix data (copy)
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mat</th>
<th>The reference to the matrix.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>array</td>
<td>The array to place the matrix.</td>
</tr>
</tbody>
</table>

See Also

vxQueryMatrix and VX_MATRIX_ATTRIBUTE_COLUMNS and VX_MATRIX_ATTRIBUTE_ROWS to get the needed number of elements of the array.

Returns

A vx_status_e enumeration.

Postcondition

vxCommitMatrix

vx_status vxCommitMatrix ( vx_matrix mat, void * array )

Sets the matrix data (copy)

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mat</th>
<th>The reference to the matrix.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>array</td>
<td>The array to read the matrix.</td>
</tr>
</tbody>
</table>

See Also

vxQueryMatrix and VX_MATRIX_ATTRIBUTE_COLUMNS and VX_MATRIX_ATTRIBUTE_ROWS to get the needed number of elements of the array.

Returns

A vx_status_e enumeration.

Precondition

vxAccessMatrix

vx_matrix 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. VX_TYPE_INT32 or VX_TYPE_FLOAT32.</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

vx_matrix

vx_status vxQueryMatrix ( vx_matrix mat, vx_enum attribute, void * ptr, vx_size size )

Queries an attribute on the matrix object.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mat</th>
<th>The matrix object to set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>attribute</td>
<td>The attribute to query. Use a vx_matrix_attribute_e enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The location at which the resulting value will be stored.</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.

void 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

| in | mat     | The matrix reference to release. |

Note

After returning from this function the reference will be zeroed.
3.54 Object: Pyramid

3.54.1 Detailed Description

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 \texttt{VX\_PYRAMID\_ATTRIBUTE\_SCALE} attribute. For instance, in a pyramid with 3 levels and \texttt{VX\_SCALE\_PYRAMID\_HALF}, the level one image will be one-half the width and one-half the height of the level zero image and the level two image will be 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 are always 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 \texttt{VX\_SCALE\_PYRAMID\_HALF} will result in the level one image with a width of $5 = \lceil 9 \times 0.5 \rceil$ and a level two image with a width of $3 = \lceil 5 \times 0.5 \rceil$).

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 \texttt{VX\_SCALE\_PYRAMID\_HALF} (0.5f)
  Used to indicate a half-scale pyramid.
- \#define \texttt{VX\_SCALE\_PYRAMID\_ORB} ((vx\_float32)0.8408964f)
  Used 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\_ATTRIBUTE\_LEVELS = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_PYRAMID << 8)) + 0x0,
  VX\_PYRAMID\_ATTRIBUTE\_SCALE = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_PYRAMID << 8)) + 0x1,
  VX\_PYRAMID\_ATTRIBUTE\_WIDTH = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_PYRAMID << 8)) + 0x2,
  VX\_PYRAMID\_ATTRIBUTE\_HEIGHT = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_PYRAMID << 8)) + 0x3,
  VX\_PYRAMID\_ATTRIBUTE\_FORMAT = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_PYRAMID << 8)) + 0x4
}
  The pyramid object attributes.

Functions

- \_vx\_pyramid \_vx\_create\_pyramid (\_vx\_context context, \_vx\_size levels, \_vx\_float32 scale, \_vx\_uint32 width, \_vx\_uint32 height, \_vx\_fourcc format)
  Creates a reference to a pyramid object of the supplied number of levels.
- \_vx\_pyramid \_vx\_create\_virtual\_pyramid (\_vx\_graph graph, \_vx\_size levels, \_vx\_float32 scale, \_vx\_uint32 width, \_vx\_-uint32 height, \_vx\_fourcc format)
  Creates a reference to a virtual pyramid object of the supplied number of levels.
- \_vx\_image \_vx\_get\_pyramid\_level (\_vx\_pyramid pyr, \_vx\_uint32 index)
  Retrieves a level of the pyramid as a \_vx\_image, which can be used elsewhere in OpenVX.
- \_vx\_status \_vx\_query\_pyramid (\_vx\_pyramid pyr, \_vx\_enum attribute, void *ptr, \_vx\_size size)
  Queries an attribute from an image pyramid.
- void \_vx\_release\_pyramid (\_vx\_pyramid *pyr)
  Releases a reference to a pyramid object. The object may not be garbage collected until its total reference count is zero.
3.54.2 Enumeration Type Documentation

```c
enum vx_pyramid_attribute_e {
    VX_PYRAMID_ATTRIBUTE_LEVELS, // The number of levels of the pyramid. Use a vx_size parameter.
    VX_PYRAMID_ATTRIBUTE_SCALE, // The scale factor between each level of the pyramid. Use a vx_float32 parameter.
    VX_PYRAMID_ATTRIBUTE_WIDTH, // The width of the 0th image in pixels. Use a vx_uint32 parameter.
    VX_PYRAMID_ATTRIBUTE_HEIGHT, // The height of the 0th image in pixels. Use a vx_uint32 parameter.
    VX_PYRAMID_ATTRIBUTE_FORMAT // The vx_fourcc_e format of the image. Use a vx_fourcc parameter.
};

Definition at line 888 of file vx_types.h.
```

3.54.3 Function Documentation

```c
vx_pyramid vxCreatePyramid ( vx_context context, vx_size levels, vx_float32 scale, vx_uint32 width, vx_uint32 height, vx_fourcc format )
```

Creates a reference to a pyramid object of the supplied number of levels.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>The reference to the overall context.</td>
</tr>
<tr>
<td>levels</td>
<td>The number of levels desired. This is required to be a non-zero value.</td>
</tr>
<tr>
<td>scale</td>
<td>Used to indicate the scale between pyramid levels. This is required to be a non-zero positive value. In OpenVX 1.0, the only permissible values are VX_SCALE_PYRAMID_HALF or VX_SCALE_PYRAMID_ORB.</td>
</tr>
<tr>
<td>width</td>
<td>The width of the 0th level image in pixels.</td>
</tr>
<tr>
<td>height</td>
<td>The height of the 0th level image in pixels.</td>
</tr>
<tr>
<td>format</td>
<td>The format of all images in the pyramid.</td>
</tr>
</tbody>
</table>

Returns

- `vx_pyramid`:

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No pyramid was created.</td>
</tr>
<tr>
<td>*</td>
<td>A pyramid reference.</td>
</tr>
</tbody>
</table>

```c
vx_pyramid vxCreateVirtualPyramid ( vx_graph graph, vx_size levels, vx_float32 scale, vx_uint32 width, vx_uint32 height, vx_fourcc 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
vx_context context = vxCreateContext();
vx_graph graph = vxCreateGraph(context);
.vx_pyramid virt[] = {
    vxCreateVirtualPyramid(graph, 4, VX_SCALE_PYRAMID_HALF, 0, 0, FOURCC_VIRT), // no dimension and format specified for level 0
    vxCreateVirtualPyramid(graph, 4, VX_SCALE_PYRAMID_HALF, 640, 480, FOURCC_VIRT), // no format specified.
    vxCreateVirtualPyramid(graph, 4, VX_SCALE_PYRAMID_HALF, 640, 480, FOURCC_U8), // no access
};
```
CHAPTER 3. MODULE DOCUMENTATION

Parameters

| in | graph | The reference to the parent graph. |
| in | levels | The number of levels desired. This is required to be a non-zero value. |
| in | scale | Used to indicate the scale between pyramid levels. This is required to be a non-zero positive value. In OpenVX 1.0, the only permissible values are VX_SCALE_PYRAMID_HALF, or VX_SCALE_PYRAMID_ORB |
| in | width | 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. |
| in | height | 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. |
| in | format | The format of all images in the pyramid. This may be set to FOURCC_VIRT to indicate that the format is unspecified. |

Returns

A vx_pyramid reference.

Note

Images extracted with vxGetPyramidLevel behave as Virtual Images and will cause vxAccessImagePatch to return errors.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No pyramid was created.</td>
</tr>
<tr>
<td>*</td>
<td>A pyramid reference.</td>
</tr>
</tbody>
</table>

vx_image vxGetPyramidLevel ( vx_pyramid pyr, vx_uint32 index )

Retrieves a level of the pyramid as a vx_image, which can be used elsewhere in OpenVX.

Parameters

| in | pyr | The pyramid object. |
| in | index | The index of the level, such that index is less than levels. |

Returns

A vx_image reference.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Indicates that the index or the object was invalid.</td>
</tr>
</tbody>
</table>

vx_status vxQueryPyramid ( vx_pyramid pyr, vx_enum attribute, void * ptr, vx_size size )

Queries an attribute from an image pyramid.

Parameters

| in | pyr | The pyramid to query. |
| in | attribute | The attribute to query for. Use a vx_pyramid_attribute_e enumeration. |
| out | ptr | The location at which the resulting value will be stored. |
| in | size | The size of the container to which ptr points. |

Returns

A vx_status_e enumeration.

void 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></th>
<th></th>
<th><strong>pyr</strong></th>
<th>The pointer to the pyramid to release.</th>
</tr>
</thead>
</table>

Note

After returning from this function the reference will be zeroed.
3.55 Object: Remap

3.55.1 Detailed Description

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_ATTRIBUTE_SOURCE_WIDTH = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_REMAP << 8)) + 0x0,
  VX_REMAP_ATTRIBUTE_SOURCE_HEIGHT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_REMAP << 8)) + 0x1,
  VX_REMAP_ATTRIBUTE_DESTINATION_WIDTH = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_REMAP << 8)) + 0x2,
  VX_REMAP_ATTRIBUTE_DESTINATION_HEIGHT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_REMAP << 8)) + 0x3
} 

The remap object attributes.

Functions

- vx_remap 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_status vxGetRemapPoint (vx_remap table, vx_uint32 dst_x, vx_uint32 dst_y, vx_float32 *src_x, vx_float32 *src_y)

  Retrieves the source pixel point from a destination pixel.

- vx_status vxQueryRemap (vx_remap remap, vx_enum attribute, void *ptr, vx_size size)

  Queries attributes from a Remap table.

- void vxReleaseRemap (vx_remap *table)

  Release a reference to a remap table object. The object may not be garbage collected until its total reference count is zero.

- vx_status vxSetRemapPoint (vx_remap table, vx_uint32 dst_x, vx_uint32 dst_y, vx_float32 src_x, vx_float32 src_y)

  Assigns a destination pixel mapping to the source pixel.

3.55.2 Enumeration Type Documentation

enum vx_remap_attribute_e

The remap object attributes.

Enumerator

- VX_REMAP_ATTRIBUTE_SOURCE_WIDTH The source width. Use a vx_uint32 parameter.
- VX_REMAP_ATTRIBUTE_SOURCE_HEIGHT The source height. Use a vx_uint32 parameter.
- VX_REMAP_ATTRIBUTE_DESTINATION_WIDTH The destination width. Use a vx_uint32 parameter.
- VX_REMAP_ATTRIBUTE_DESTINATION_HEIGHT The destination height. Use a vx_uint32 parameter.

Definition at line 904 of file vx_types.h.
3.55.3 Function Documentation

vx_remap 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

<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>src_width</td>
<td>Width of the source image in pixels.</td>
</tr>
<tr>
<td>in</td>
<td>src_height</td>
<td>Height of the source image in pixels.</td>
</tr>
<tr>
<td>in</td>
<td>dst_width</td>
<td>Width of the destination image in pixels.</td>
</tr>
<tr>
<td>in</td>
<td>dst_height</td>
<td>Height of the destination image in pixels.</td>
</tr>
</tbody>
</table>

### Returns

**Returns** vx_remap

**Return values**

| 0 | Object could not be created. |
| ∗ | Object was created. |

#### vx_status vxGetRemapPoint ( vx_remap table, vx_uint32 dst_x, vx_uint32 dst_y, vx_float32 ∗ src_x, vx_float32 ∗ src_y )

Retrieves the source pixel point from a destination pixel.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>table</th>
<th>The remap table reference.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dst_x</td>
<td>The destination x coordinate.</td>
</tr>
<tr>
<td>in</td>
<td>dst_y</td>
<td>The destination y coordinate.</td>
</tr>
<tr>
<td>out</td>
<td>src_x</td>
<td>The pointer to the location to store the source x coordinate in float representation to allow interpolation.</td>
</tr>
<tr>
<td>out</td>
<td>src_y</td>
<td>The pointer to the location to store the source y coordinate in float representation to allow interpolation.</td>
</tr>
</tbody>
</table>

**Returns**

A vx_status_e enumeration.

#### vx_status vxQueryRemap ( vx_remap r, vx_enum attribute, void ∗ ptr, vx_size size )

Queries attributes from a Remap table.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>r</th>
<th>The remap to query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>attribute</td>
<td>The attribute to query. Use a vx_remap_attribute_e enumeration.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The location at which the resulting value will be stored.</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.

#### void vxReleaseRemap ( vx_remap ∗ table )

Release a reference to a remap table object. The object may not be garbage collected until its total reference count is zero.

**Parameters**
CHAPTER 3. MODULE DOCUMENTATION

177

The pointer to the remap table to release.

Note

After returning from this function the reference will be zeroed.

vx_status vxSetRemapPoint ( vx_remap table, vx_uint32 dst_x, vx_uint32 dst_y, vx_float32 src_x, vx_float32 src_y )

Assigns a destination pixel mapping to the source pixel.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>The remap table reference.</td>
</tr>
<tr>
<td>dst_x</td>
<td>The destination x coordinate.</td>
</tr>
<tr>
<td>dst_y</td>
<td>The destination y coordinate.</td>
</tr>
<tr>
<td>src_x</td>
<td>The source x coordinate in float representation to allow interpolation.</td>
</tr>
<tr>
<td>src_y</td>
<td>The source y coordinate in float representation to allow interpolation.</td>
</tr>
</tbody>
</table>

Returns

Returns a vx_status_e enumeration.
3.56 Object: Scalar

3.56.1 Detailed Description

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_ATTRIBUTE_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_SCALAR << 8)) + 0x0 }

  The scalar attributes list.

Functions

- vx_status vxAccessScalarValue (vx_scalar ref, void *ptr)
  
  Gets the scalar value out of a reference.

- vx_status vxCommitScalarValue (vx_scalar ref, void *ptr)
  
  Sets the scalar value in a reference.

- vx_scalar vxCreateScalar (vx_context context, vx_enum data_type, void *ptr)
  
  Creates a reference to a scalar object. Also see Node Parameters.

- vx_status vxQueryScalar (vx_scalar scalar, vx_enum attribute, void *ptr, vx_size size)
  
  Queries attributes from a scalar.

- void 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.

3.56.2 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 137 of file vx_types.h.

3.56.3 Enumeration Type Documentation

enum vx_scalar_attribute_e

The scalar attributes list.

Enumerator

- VX_SCALAR_ATTRIBUTE_TYPE Used to query the type of atomic is contained in the scalar. Use a vx_enum parameter.

  Definition at line 780 of file vx_types.h.
3.56.4 Function Documentation

vx_status vxAccessScalarValue ( vx_scalar ref, void * ptr )

Gets the scalar value out of a reference.

Note

Use this in conjunction with Query APIs which return references which should be converted into values.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ref</th>
<th>The reference to get the scalar value from.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>ptr</td>
<td>An appropriate typed pointer which points to a location to copy the scalar value to.</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_ERROR_INVALID_REFERENCE</td>
<td>Will be returned if the ref is not a valid reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>will be returned if ptr is NULL.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_TYPE</td>
<td>will be returned if the type does not match the type in the reference or is a bad value.</td>
</tr>
</tbody>
</table>

vx_status vxCommitScalarValue ( vx_scalar ref, void * ptr )

Sets the scalar value in a reference.

Note

Use this in conjunction with Parameter APIs which return references to parameters which need to be altered.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ref</th>
<th>The reference to get the scalar value from.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ptr</td>
<td>An appropriately typed pointer which points to a location to copy the scalar value to.</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_ERROR_INVALID_REFERENCE</td>
<td>Will be returned if the ref is not a valid reference.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>will be returned if ptr is NULL.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_TYPE</td>
<td>will be returned if the type does not match the type in the reference or is a bad value.</td>
</tr>
</tbody>
</table>

vx_scalar vxCreateScalar ( vx_context context, vx_enum data_type, void * ptr )

Creates a reference to a scalar object. Also see Node Parameters.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>The reference to the system context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data_type</td>
<td>The vx.type.e of the scalar. Must be greater than VX_TYPE_INVALID and less than VX_TYPE_SCALAR_MAX.</td>
</tr>
<tr>
<td>in</td>
<td>ptr</td>
<td>The pointer to the initial value of the scalar.</td>
</tr>
</tbody>
</table>

Returns

A vx.scalar reference.

Return values

| 0   | The scalar could not be created. |
| *   | The scalar was created. Check for further errors with vxGetStatus. |

vx_status 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 the resulting value will be stored.</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.

void 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. |

Note

After returning from this function the reference will be zeroed.
3.57 Object: Threshold

3.57.1 Detailed Description

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_ATTRIBUTE_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_THRESHOLD << 8)) + 0x0,
    VX_THRESHOLD_ATTRIBUTE_VALUE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_THRESHOLD << 8)) + 0x1,
    VX_THRESHOLD_ATTRIBUTE_LOWER = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_THRESHOLD << 8)) + 0x2,
    VX_THRESHOLD_ATTRIBUTE_UPPER = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_THRESHOLD << 8)) + 0x3
  }

  The threshold attributes.

- enum vx_threshold_type_e {
  
    VX_THRESHOLD_TYPE_BINARY = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_THRESHOLD_TYPE << 12)) + 0x0,
    VX_THRESHOLD_TYPE_RANGE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_THRESHOLD_TYPE << 12)) + 0x1
  }

  The Threshold types.

Functions

- vx_threshold vxCreateThreshold (vx_context c, vx_enum thresh_type, vx_enum data_type)
  
  Creates a reference to a threshold object of a given type.

- vx_status vxQueryThreshold (vx_threshold thresh, vx_enum attribute, void *ptr, vx_size size)
  
  Queries an attribute on the threshold object.

- void 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 vxSetThresholdAttribute (vx_threshold thresh, vx_enum attribute, void *ptr, vx_size size)
  
  Sets attributes on the threshold object.

3.57.2 Enumeration Type Documentation

enum vx_threshold_attribute_e

The threshold attributes.

Enumerator

- VX_THRESHOLD_ATTRIBUTE_TYPE The value type of the threshold. Use a vx_enum parameter. Will contain a vx_threshold_type_e.

- VX_THRESHOLD_ATTRIBUTE_VALUE The value of the single threshold. Use a vx_int32 parameter.

- VX_THRESHOLD_ATTRIBUTE_LOWER The value of the lower threshold. Use a vx_int32 parameter.

- VX_THRESHOLD_ATTRIBUTE_UPPER The value of the higher threshold. Use a vx_int32 parameter.

Definition at line 842 of file vx_types.h.
enum vx_threshold_type_e
The Threshold types.

Enumerator

\texttt{VX\_THRESHOLD\_TYPE\_BINARY} A threshold with only 1 value.
\texttt{VX\_THRESHOLD\_TYPE\_RANGE} A threshold with 2 values (upper/lower). Used with canny edge detection.

Definition at line 832 of file \texttt{vx\_types.h}.

3.57.3 Function Documentation

\texttt{vx\_threshold vxCreateThreshold ( vx\_context c, vx\_enum thresh\_type, vx\_enum data\_type )}
Creates a reference to a threshold object of a given type.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{in} \texttt{c}</td>
</tr>
<tr>
<td>\textit{in} \texttt{thresh_type}</td>
</tr>
<tr>
<td>\textit{in} \texttt{data_type}</td>
</tr>
</tbody>
</table>

Note For OpenVX 1.0, \texttt{data\_type} can only be \texttt{VX\_TYPE\_UINT8}.

Returns \texttt{vx\_threshold}

\texttt{vx\_status vxQueryThreshold ( vx\_threshold thresh, vx\_enum attribute, void \* ptr, vx\_size size )}
Queries an attribute on the threshold object.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{in} \texttt{thresh}</td>
</tr>
<tr>
<td>\textit{in} \texttt{attribute}</td>
</tr>
<tr>
<td>\textit{out} \texttt{ptr}</td>
</tr>
<tr>
<td>\textit{in} \texttt{size}</td>
</tr>
</tbody>
</table>

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

\texttt{void 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.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{in} \texttt{thresh}</td>
</tr>
</tbody>
</table>

Note After returning from this function the reference will be zeroed.

\texttt{vx\_status vxSetThresholdAttribute ( vx\_threshold thresh, vx\_enum attribute, void \* ptr, vx\_size size )}
Sets attributes 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 modify. Use a <code>vx_threshold_attribute_e</code> enumeration.</td>
</tr>
<tr>
<td>in</td>
<td>ptr</td>
<td>The pointer to the value to set the attribute to.</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>The size of the data pointed to by ptr.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.
3.58 Basic Framework

3.58.1 Detailed Description

The framework concepts and interfaces of OpenVX. These interfaces are used to perform basic tasks such as performance measurement on \texttt{vx\_graph} or \texttt{vx\_node} objects.

Modules

- **Framework: Node Callbacks**
  
  This allows Clients to receive a callback after a specific node has completed execution.

- **Framework: Performance Measurement**
  
  The Performance measurement and reporting interfaces.
3.59 Framework: Node Callbacks

3.59.1 Detailed Description

This 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 `vx_action` return values. An example of setting up a callback can be seen below:

```c
vx_graph graph = vxCreateGraph(context);
if (graph) {
  vx_int32 lmin = 0, lmax = 0;
  vx_int32 minCount = 0, maxCount = 0;
  vx_scalar scalars[] = {
    vxCreateScalar(context, VX_TYPE_UINT8, &lmin),
   vxCreateScalar(context, VX_TYPE_UINT8, &lmax),
    vxCreateScalar(context, VX_TYPE_UINT32, &minCount),
    vxCreateScalar(context, VX_TYPE_UINT32, &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 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_ATTRIBUTE_REF, &smax, sizeof(smax));
    if (smax) {
      vx_int8 value = 0;
      vxAccessScalarValue(smax, &value);
      if (value >= MY_DESIRED_THRESHOLD) {
        action = VX_ACTION_CONTINUE;
      }
    }
    vxReleaseScalar(&smax);
  }
  vxReleaseParameter(&pmax);
  return action;
}
```

Note

This should be used with extreme caution as it can ruin optimizations in the power/performance efficiency of a graph.

The callback must return a `vx_action` code which indicates how the graph processing should proceed.

- If `VX_ACTION_CONTINUE` is returned, the graph will continue execution with no changes.
- If `VX_ACTION_ABANDON` is returned, all further nodes following this node in the graph will not execute. Executions of nodes in independent branches of the graph are unspecified.
- If `VX_ACTION_RESTART` is returned, all further nodes following this node of the graph will not execute. Executions of nodes in independent branches of the graph are unspecified. Once the graph halts it will restart execution.
CHAPTER 3. MODULE DOCUMENTATION

Graph Setup

vxAssignNodeCallback(node, &myfunc);

vx_status

vxVerifyGraph(graph);

vx_status

vxProcessGraph(graph);

node executes

myfunc(node, ...);

vx_action

action taken

vx_status

vxRetrieveNodeCallback

(vx_nodecomplete_f)myfunc

Figure 3.1: 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_action(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_RESTART = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_ACTION << 12)) + 0x1,
  VX_ACTION_ABANDON = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_ACTION << 12)) + 0x2
  }

  A return code enumeration from a vx_nodecomplete_f during execution.

**Functions**

- vx_status 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 vxRetrieveNodeCallback (vx_node node)
  
  Retrieves the current node callback function pointer set on the node.
3.59.2 Typedef Documentation

typedef vx enum vx action
The formal typedef of the response from the callback.
See Also
  vx_action_e
  Definition at line 363 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 which the callback was attached.

Returns
  Returns an action code from vx_action_e.
  Definition at line 372 of file vx_types.h.

3.59.3 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_RESTART Stop executing the graph at the current point and restart from the beginning.
  VX_ACTION_ABANDON Stop executing the graph.
  Definition at line 504 of file vx_types.h.

3.59.4 Function Documentation

vx status 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.

Note
  This should 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_SUCCESS</th>
<th>Callback assigned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>The value passed as node was not a vx_node.</td>
</tr>
</tbody>
</table>

`vx_nodecomplete_f 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>NULL</th>
<th>No callback has been set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>The node callback function.</td>
</tr>
</tbody>
</table>
3.60  Framework: Performance Measurement

3.60.1  Detailed Description

The 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>_\_ATTRIBUTE\_PERFORMANCE along with a vx\_perf\_t structure to obtain the performance information.

\[
\begin{align*}
\text{vx\_perf\_t \ perf;} \\
vxQueryNode(node, VX\_\text{NODE}_\text{ATTRIBUTE\_PERFORMANCE}, &perf, sizeof(perf))
\end{align*}
\]

Data Structures

- struct vx\_perf\_t
  
  \textit{The performance measurement structure. More...}

3.60.2  Data Structure Documentation

struct vx\_perf\_t

The performance measurement structure.

Definition at line 1282 of file vx\_types.h.

<table>
<thead>
<tr>
<th>Data Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_uint64 avg</td>
<td>Used to hold the average of the durations.</td>
</tr>
<tr>
<td>vx_uint64 beg</td>
<td>Used to hold the first measurement in a set.</td>
</tr>
<tr>
<td>vx_uint64 end</td>
<td>Used to hold the last measurement in a set.</td>
</tr>
<tr>
<td>vx_uint64 min</td>
<td>Used to hold the minimum of the durations.</td>
</tr>
<tr>
<td>vx_uint64 num</td>
<td>Used to hold the number of measurements.</td>
</tr>
<tr>
<td>vx_uint64 sum</td>
<td>Used to hold the summation of durations.</td>
</tr>
<tr>
<td>vx_uint64 tmp</td>
<td>Used to hold the last measurement.</td>
</tr>
</tbody>
</table>
3.61 Advanced Features

3.61.1 Detailed Description

The more advanced features of OpenVX. These features require more understanding and are more complex to use.

Modules

- Advanced Framework API
  
  Components in this set are considered to be advanced.

- Advanced Objects
3.62 Advanced Objects

3.62.1 Detailed Description

Modules

• **Object: Array (Advanced)**
  
  *These are the advanced features of the Array Interface.*

• **Object: Delay**
  
  *The Delay Object interface.*

• **Object: Kernel**
  
  *The Kernel Object and Interface.*

• **Object: Node (Advanced)**
  
  *These are advanced features of the Node Interface.*

• **Object: Parameter**
  
  *The Parameter Object interface.*
CHAPTER 3. MODULE DOCUMENTATION

3.63 Object: Array (Advanced)

3.63.1 Detailed Description

These are the advanced features of the Array Interface.

Functions

- \texttt{vx\_enum vxRegisterUserStruct (vx\_context context, vx\_size size)}

  \textit{Register user defined structures to the context.}

3.63.2 Function Documentation

\texttt{vx\_enum vxRegisterUserStruct (vx\_context context, vx\_size size)}

Register user defined structures to the context.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{context}</td>
</tr>
<tr>
<td>\textit{size}</td>
</tr>
</tbody>
</table>

Returns

Returns a \texttt{vx\_enum} value which is a type given to the User to refer to their custom structure when declaring a \texttt{vx\_array} of that structure.

Return values

| \texttt{VX\_TYPE\_INVALID} | Returned when 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.

```c
typedef struct _mystruct {
  vx\_uint32 some\_uint;
  vx\_float64 some\_double;
} mystruct;
#define MY\_NUM\_ITEMS (10)
vx\_enum mytype = vxRegisterUserStruct(context, sizeof\{mystruct\});
vx\_array array = vxCreateArray(context, mytype, MY\_NUM\_ITEMS);
```
3.64 Object: Node (Advanced)

3.64.1 Detailed Description

These are advanced features of the Node Interface.

Modules

- Node: Border Modes
  The border mode behaviors.

Functions

- vx_node vxCreateNode (vx_graph graph, vx_kernel kernel)
  Creates a reference to a node object for a given kernel.

3.64.2 Function Documentation

vx_node vxCreateNode ( 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 Client Defined Functions), this is the API that should be used along with the parameter setting API.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph</td>
<td>The reference to the graph in which this node will exist.</td>
</tr>
<tr>
<td>kernel</td>
<td>The kernel reference which will be associated with this new node.</td>
</tr>
</tbody>
</table>

Returns

vx_node

Return values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The node failed to create.</td>
</tr>
<tr>
<td>*</td>
<td>A node was created.</td>
</tr>
</tbody>
</table>

Postcondition

Call vxSetParameterByIndex for as many parameters as need to be set.
3.65  Node: Border Modes

3.65.1  Detailed Description

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 which supports border modes. Most Nodes do not support any explicit border modes beyond VX_BORDER_MODE_UNDEFINED.

Data Structures

- struct vx_border_mode_t

  Used with the enumeration VX_NODE_ATTRIBUTE_BORDER_MODE to set the border mode behavior of a node which supports borders. More...

Enumerations

- enum vx_border_mode_e {

  VX_BORDER_MODE_UNDEFINED = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_BORDER_MODE << 12)) + 0x0,
  VX_BORDER_MODE_CONSTANT = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_BORDER_MODE << 12)) + 0x1,
  VX_BORDER_MODE_REPLICATE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_BORDER_MODE << 12)) + 0x2

  The border mode list.
}

3.65.2  Data Structure Documentation

struct vx_border_mode_t

Used with the enumeration VX_NODE_ATTRIBUTE_BORDER_MODE to set the border mode behavior of a node which supports borders.

Definition at line 1327 of file vx_types.h.

Data Fields

- vx_uint32 constant_value

  For the mode VX_BORDER_MODE_CONSTANT, this value will be filled into each pixel. If there are sub-channels in the pixel then this value will be divided up accordingly.

- vx_enum mode

  See vx_border_mode_e.

3.65.3  Enumeration Type Documentation

enum vx_border_mode_e

The border mode list.

Enumerator

- VX_BORDER_MODE_UNDEFINED No defined border mode behavior is given.
- VX_BORDER_MODE_CONSTANT For nodes which support this behavior, a constant value is "filled-in" when accessing out-of-bounds pixels.
- VX_BORDER_MODE_REPLICATE For nodes which support this behavior, a replication of the nearest edge pixels value is given for out-of-bounds pixels.

Definition at line 1050 of file vx_types.h.
3.66 Object: Delay

3.66.1 Detailed Description

The Delay Object interface. A Delay is an opaque object which contains a manually control temporally-delayed list of objects.

Typedefs

- typedef struct _vx_delay * vx_delay
  
  The delay object. This is like a ring buffer of objects which is maintained by the OpenVX implementation.

Enumerations

- enum vx_delay_attribute_e {
  
  VX_DELAY_ATTRIBUTE_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DELAY << 8)) + 0x0,
  VX_DELAY_ATTRIBUTE_COUNT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_DELAY << 8)) + 0x1
  }

  The delay attribute list.

Functions

- vx_status vxAgeDelay (vx_delay delay)
  
  Ages the internal delay ring by one. This means that once this API is called the reference from index 0 will go to index -1 and so forth until -count + 1 is reached. This last object will become 0. Once the delay has been aged, it will update the reference in any associated nodes.

- vx_status vxAssociateDelayWithNode (vx_delay delay, vx_int32 delay_index, vx_node node, vx_uint32 param_index)
  
  Associates a delay index with a particular node’s parameter.

- vx_delay vxCreateDelay (vx_context context, vx_reference exemplar, vx_size count)
  
  Creates a Delay object.

- vx_status vxDissociateDelayFromNode (vx_delay delay, vx_int32 delay_index, vx_node node, vx_uint32 param_index)
  
  Dissociates a delay index from a particular node parameter.

- vx_reference vxGetReferenceFromDelay (vx_delay delay, vx_int32 index)
  
  Retrieves a reference from an delay object.

- vx_status vxQueryDelay (vx_delay delay, vx_enum attribute, void **ptr, vx_size size)
  
  Queries a vx_delay object attribute.

- void 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..

3.66.2 Typedef Documentation

typedef struct _vx_delay * vx_delay

The delay object. This is like a ring buffer of objects which is maintained by the OpenVX implementation.

See Also

vxCreateDelay

Definition at line 188 of file vx_types.h.
3.66.3 Enumeration Type Documentation

```c
enum vx_delay_attribute_e
```

The delay attribute list.

**Enumerator**

- `VX_DELAY_ATTRIBUTE_TYPE` The type of reference contained in the delay. Use a `vx_enum` parameter.
- `VX_DELAY_ATTRIBUTE_COUNT` The number of items in the delay. Use a `vx_uint32` parameter.

Definition at line 1092 of file `vx_types.h`.

3.66.4 Function Documentation

```c
vx_status vxAgeDelay ( vx_delay delay )
```

Ages the internal delay ring by one. This means that once this API is called the reference from index 0 will go to index -1 and so forth until $-\text{count} + 1$ is reached. This last object will become 0. Once the delay has been aged, it will update the reference in any associated nodes.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>delay</th>
</tr>
</thead>
</table>

**Returns**

A `vx_status_e` enumeration.

**Return values**

| `VX_SUCCESS` | Delay was aged. |
| `VX_ERROR_INVALID_REFERENCE` | The value passed as delay was not a `vx_delay`. |

**Precondition**

`vxAssociateDelayWithNode`

```c
vx_status vxAssociateDelayWithNode ( vx_delay delay, vx_int32 delay_index, vx_node node, vx_uint32 param_index )
```

Associates a delay index with a particular node's parameter.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>delay</th>
<th>The reference to the delay object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>delay_index</td>
<td>The index of the object in the delay object. This is in the range of $[-\text{count} + 1, 0]$.</td>
</tr>
<tr>
<td>in</td>
<td>node</td>
<td>The reference to the node.</td>
</tr>
<tr>
<td>in</td>
<td>param_index</td>
<td>The index of the parameter on the node. This is in the range of $[0, \text{numParams} - 1]$.</td>
</tr>
</tbody>
</table>

**Returns**

A `vx_status_e` enumeration.
### Postcondition

```c
vxAgeDelay
/*
   * Copyright (c) 2012-2013 The Khronos Group Inc.
   * Permission is hereby granted, free of charge, to any person obtaining a
   * copy of this software and/or associated documentation files (the
   * "Materials"), to deal in the Materials without restriction, including
   * without limitation the rights to use, copy, modify, merge, publish,
   * distribute, sublicense, and/or sell copies of the Materials, and to
   * permit persons to whom the Materials are furnished to do so, subject to
   * the following conditions:
   * The above copyright notice and this permission notice shall be included
   * in all copies or substantial portions of the Materials.
   * THE MATERIALS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
   * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
   * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
   * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
   * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
   * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
   * MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS.
*/
#include <VX/vx.h>
void example_delaygraph(vx_context context) {
  vx_status status = VX_SUCCESS;
  vx_image yuv = vxCreateImage(context, 320, 240,
                                 FOURCC_UYVY);
  vx_delay delay = vxCreateDelay(context, (vx_reference)yuv, 4);
  vx_image rgb = vxCreateImage(context, 320, 240,
                                FOURCC_JPEG);
  vx_graph graph = vxCreateGraph(context);
  vx_node convert = vxColorConvertNode(graph, (vx_image)vxGetReferenceFromDelay(delay, 0), rgb);
  if (vxAssociateDelayWithNode(delay, 0, convert, 0) == VX_SUCCESS) {
    status = vxVerifyGraph(graph);
    if (status == VX_SUCCESS) {
      do {
        /* capture or read image into vxGetImageFromDelay(delay, 0); */
        status = vxProcessGraph(graph);
        /* 0 becomes -1, -1 becomes -2, etc. convert is updated with new 0 */
        vxAgeDelay(delay);
      } while (1);
    }
  }
}

vx_delay vxCreateDelay ( vx_context context, vx_reference exemplar, vx_size count )

Creates a Delay object.

This function uses only the metadata from the exemplar, ignoring the object data. It does not alter the exemplar
or keep or release the reference to the exemplar.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in context</td>
<td>The reference to the system context.</td>
</tr>
<tr>
<td>in exemplar</td>
<td>The exemplar object.</td>
</tr>
<tr>
<td>in count</td>
<td>The number of reference in the delay.</td>
</tr>
</tbody>
</table>

**Returns**

vx_delay

vx_status vxDissociateDelayFromNode ( vx_delay delay, vx_int32 delay_index, vx_node node, vx_uint32 param_index )

Dissociates a delay index from a particular node parameter.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>delay</td>
<td>The reference to the delay object.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td>delay_index</td>
<td>The relative index of the object in the delay.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td>node</td>
<td>The reference to the node.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td>param_index</td>
<td>The index to the parameter on the node.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

```c
vx_reference vxGetReferenceFromDelay ( vx_delay delay, vx_int32 index )
```

Retrieves a reference from a delay object.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>delay</td>
<td>The reference to the delay object.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td>index</td>
<td>An index into the delay from which to extract the reference.</td>
</tr>
</tbody>
</table>

Returns

A `vx_reference`

Note

The delay index is in the range \([-\text{count} + 1, 0]\). 0 is always the "current" object.
A reference from a delay object should not be given to its associated release API (e.g. `vxReleaseImage`).
Use the `vxReleaseDelay` only.

```c
vx_status vxQueryDelay ( vx_delay delay, vx_enum attribute, void * ptr, vx_size size )
```

Queries a `vx_delay` object attribute.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>delay</td>
<td>The coordinates object to set.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td>attribute</td>
<td>The attribute to query. Use a <code>vx_delay_attribute_e</code> enumeration.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td>ptr</td>
<td>The location at which the resulting value will be stored.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td>size</td>
<td>The size of the container to which ptr points.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

```c
void 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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>delay</td>
<td>The pointer to the delay to release.</td>
</tr>
</tbody>
</table>

Note

After returning from this function the reference will be zeroed.
3.67 Object: Kernel

3.67.1 Detailed Description

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 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 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 = vxCreateNode(graph, kernel);
```

The second method depends on using strings to get the kernel reference.

```c
vx_kernel kernel = vxGetKernelByName(context, "org.khronos.openvx.sobel3x3");
vx_node node = vxCreateNode(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)
  
  The maximum string length of a kernel name to be added to OpenVX.

Typedefs

- typedef struct vx_kernel *vx_kernel
  
  An opaque reference to the descriptor of a kernel.

Enumerations

- enum vx_kernel_attribute_e {

  VX_KERNEL_ATTRIBUTE_NUMPARAMS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_KERNEL << 8)) + 0x0,
  VX_KERNEL_ATTRIBUTE_NAME = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_KERNEL << 8)) + 0x1,
  VX_KERNEL_ATTRIBUTE_ENUM = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_KERNEL << 8)) + 0x2,
  VX_KERNEL_ATTRIBUTE_LOCAL_DATA_SIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_KERNEL << 8)) + 0x3,
  VX_KERNEL_ATTRIBUTE_LOCAL_DATA_PTR = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_KERNEL << 8)) + 0x4
  }

  The kernel attributes list.

- enum vx_kernel_e {

  VX_KERNEL_INVALID = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x0,
  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,
VX_KERNEL_HISTOGRAM = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x9,
VX_KERNEL_EQUALIZE_HISTOGRAM = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0xA,
VX_KERNEL_ABSDIFF = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0xB,
VX_KERNEL_MEAN_STDDEV = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0xC,
VX_KERNEL_THRESHOLD = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0xD,
VX_KERNEL_INTEGRAL_IMAGE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0xE,
VX_KERNEL_DILATE_3x3 = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0xF,
VX_KERNEL_ERODE_3x3 = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x10,
VX_KERNEL_MEDIAN_3x3 = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x11,
VX_KERNEL_BOX_3x3 = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x12,
VX_KERNEL_GAUSSIAN_3x3 = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x13,
VX_KERNEL_CUSTOM_CONVOLUTION = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x14,
VX_KERNEL_GAUSSIAN_PYRAMID = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x15,
VX_KERNEL_ACCUMULATE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x16,
VX_KERNEL_ACCUMULATE_WEIGHTED = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x17,
VX_KERNEL_ACCUMULATE_SQUARE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x18,
VX_KERNEL_MINMAXLOC = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x19,
VX_KERNEL_CONVERTDEPTH = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x1A,
VX_KERNEL_CANNY_EDGE_DETECTOR = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x1B,
VX_KERNEL_AND = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x1C,
VX_KERNEL_OR = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x1D,
VX_KERNEL_XOR = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x1E,
VX_KERNEL_NOT = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x1F,
VX_KERNEL_MULTIPLY = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x20,
VX_KERNEL_ADD = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x21,
VX_KERNEL_SUBTRACT = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x22,
VX_KERNEL_WARP_AFFINE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x23,
VX_KERNEL_WARP_PERSPECTIVE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x24,
VX_KERNEL_HARRIS_CORNERS = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x25,
VX_KERNEL_FAST_CORNERS = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x26,
VX_KERNEL_OPTICAL_FLOW_PYR_LK = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x27,
VX_KERNEL_REMAP = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x28,
VX_KERNEL_MAX_1_0 } }

The standard list of available vision kernels.

Functions

- `vx_kernel vxGetKernelByEnum (vx_context context, vx_enum kernel)`
  Obtains a reference to the kernel using the `vx_kernel_e` enumeration.
- `vx_kernel vxGetKernelByName (vx_context context, vx_char *name)`
Obtains a reference to a kernel using a string to specify the name.

- **vx_status 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.

- **void vxReleaseKernel (vx_kernel *kernel)**
  
  Release the reference to the kernel. The object may not be garbage collected until its total reference count is zero.

### 3.67.2 Data Structure Documentation

**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 1301 of file vx_types.h.

**Data Fields**

<table>
<thead>
<tr>
<th>vx_enum</th>
<th>enumeration</th>
<th>The kernel enumeration value from vx_kernel_e (or an extension thereof)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_char</td>
<td>name[VX_MAX_KERNEL_NAME]</td>
<td>The kernel name in dotted hierarchical format. e.g. &quot;org.khronos.-openvx.sobel3x3&quot;.</td>
</tr>
</tbody>
</table>

### 3.67.3 Typedef Documentation

**typedef struct _vx_kernel* vx_kernel**

An opaque reference to the descriptor of a kernel.

See Also

- vxGetKernelByName
- vxGetKernelByEnum

Definition at line 152 of file vx_types.h.

### 3.67.4 Enumeration Type Documentation

**enum vx_kernel_attribute_e**

The kernel attributes list.

**Enumerator**

- **VX_KERNEL_ATTRIBUTE_NUMPARAMS** Used to query a kernel for the number of parameters the kernel supports. Use a vx_uint32 parameter.
- **VX_KERNEL_ATTRIBUTE_NAME** Used to query the name of the kernel. Not settable. Use a vx_char[VX_MAX_KERNEL_NAME] array (not a vx_array).
- **VX_KERNEL_ATTRIBUTE_ENUM** Used to query the enum of the kernel. Not settable. Use a vx_enum parameter.
- **VX_KERNEL_ATTRIBUTE_LOCAL_DATA_SIZE** The local data area allocated with each kernel when it becomes a node. Use a vx_size parameter.

Note

If not set it will default to zero.

- **VX_KERNEL_ATTRIBUTE_LOCAL_DATA_PTR** The local data pointer allocate with each kernel when it becomes a node. Use a void pointer parameter. Use a vx_size parameter.

Definition at line 694 of file vx_types.h.
enum 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 vxSetParameterByIndex *
- VX_TYPE_ARRAY for a vx_array in the size field of vxGetParameterByIndex or vxSetParameterByIndex *
- or other appropriate types in vx_type.e.

Note

All kernels in the lower level specification would be reflected here. These names are prone to changing before the specification is complete.

Enumerator

VX_KERNEL_INVALID The invalid kernel is used to for conformance failure in relation to some kernel operation (Get/Release). If the kernel is executed it shall always return an error. The kernel has no parameters. To address by name use "org.khronos.openvx.invalid".

VX_KERNEL_COLOR_CONVERT The Color Space conversion kernel. The conversions are based on the vx_fourcc_e code in the images.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>The input image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The output image.</td>
</tr>
</tbody>
</table>

See Also

Function: 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.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>The input image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vx_enum</td>
<td>The channel index. This is not dependant on the input channel order or packing. See vx_channel.e.</td>
</tr>
</tbody>
</table>

See Also

Function: Channel Extract

VX_KERNEL_CHANNEL_COMBINE The Generic Channel Combine Kernel. This kernel combine multiple individual planes into a single multiplanar image of the type specified in the output image.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>Plane 0, Must be FOURCC_U8</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vx_image</td>
<td>Plane 1, Must be FOURCC_U8</td>
</tr>
<tr>
<td>in</td>
<td>vx_image</td>
<td>Plane 2, [optional] Must be FOURCC_U8</td>
</tr>
<tr>
<td>in</td>
<td>vx_image</td>
<td>Plane 3, [optional] Must be FOURCC_U8</td>
</tr>
<tr>
<td>out</td>
<td>vx_image</td>
<td>Output Image. FOURCC_RGB, FOURCC_NV12, or FOURCC_IYUV.</td>
</tr>
</tbody>
</table>

See Also

Function: Channel Combine

**VX_KERNEL_SOBEL_3x3**  The Sobel 3x3 Filter Kernel.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>Input Image. Must be FOURCC_U8</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vx_image</td>
<td>Output Gradient X image. Must be FOURCC_S16.</td>
</tr>
<tr>
<td>out</td>
<td>vx_image</td>
<td>Output Gradient Y image. Must be FOURCC_S16.</td>
</tr>
</tbody>
</table>

See Also

Function: Sobel 3x3

**VX_KERNEL_MAGNITUDE**  The Magnitude Kernel. This kernel produces a magnitude plane from two input gradients.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>The input x image in FOURCC_S16</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vx_image</td>
<td>The input y image in FOURCC_S16</td>
</tr>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The output magnitude plane in FOURCC_U8.</td>
</tr>
</tbody>
</table>

See Also

Function: Magnitude

**VX_KERNEL_PHASE**  The Magnitude Kernel. This kernel produces a phase plane from two input gradients.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>The input x image in FOURCC_S16</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vx_image</td>
<td>The input y image in FOURCC_S16</td>
</tr>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The output phase plane in FOURCC_U8. 0-255 map to 0 to 2*PI.</td>
</tr>
</tbody>
</table>

See Also

Function: 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 but the relative sizes of the input and output.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>The input image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The output image. This must not be a virtual image.</td>
</tr>
<tr>
<td>in</td>
<td>vx_enum</td>
<td>The filtering type. VX_FILTER_DEFAULT is the default.</td>
</tr>
</tbody>
</table>

See Also

Function: Scale Image

**VX_KERNEL_TABLE_LOOKUP**  The Table Lookup kernel.

Parameters
CHAPTER 3. MODULE DOCUMENTATION

in vx_image The input image in FOURCC_U8.
in vx_lut The LUT which is of type VX_TYPE_UINT8.
out vx_image The output image of type FOURCC_U8.

See Also
Function: TableLookup

VX_KERNEL_HISTOGRAM The Histogram Kernel.
Parameters

<table>
<thead>
<tr>
<th>in vx_image</th>
<th>The input image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out vx_distribution</td>
<td>The distribution.</td>
</tr>
</tbody>
</table>

See Also
Function: Histogram

VX_KERNEL_EQUALIZE_HISTOGRAM The Histogram Equalization Kernel.
Parameters

<table>
<thead>
<tr>
<th>in vx_image</th>
<th>The grayscale input image in FOURCC_U8.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out vx_image</td>
<td>The grayscale output image of type FOURCC_U8 with equalized brightness and contrast.</td>
</tr>
</tbody>
</table>

See Also
Function: Equalize Histogram

VX_KERNEL_ABSDIFF The Absolute Difference Kernel.
Parameters

<table>
<thead>
<tr>
<th>in vx_image</th>
<th>An input image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in vx_image</td>
<td>An input image.</td>
</tr>
<tr>
<td>out vx_image</td>
<td>The output image.</td>
</tr>
</tbody>
</table>

See Also
Function: Absolute Difference

VX_KERNEL_MEAN_STDDEV The Mean and Standard Deviation Kernel.
Parameters

<table>
<thead>
<tr>
<th>in vx_image</th>
<th>The input image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out vx_scalar</td>
<td>A VX_TYPE_FLOAT32 value outputting the mean.</td>
</tr>
<tr>
<td>out vx_scalar</td>
<td>A VX_TYPE_FLOAT32 value outputting the standard deviation.</td>
</tr>
</tbody>
</table>

See Also
Function: Mean and Standard Deviation.

VX_KERNEL_THRESHOLD The Threshold Kernel.
Parameters

<table>
<thead>
<tr>
<th>in vx_image</th>
<th>A FOURCC_U8 image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in vx_threshold</td>
<td>A VX_THRESHOLD_TYPE_BINARY</td>
</tr>
<tr>
<td>out vx_image</td>
<td>A FOURCC_U8 with either 0 or 255 as values.</td>
</tr>
</tbody>
</table>

See Also
Function: Thresholding

VX_KERNEL_INTEGRAL_IMAGE The Integral Image Kernel.
### VX_KERNEL_DILATE_3x3
The dilate kernel.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>The FOURCC_U8 input image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The FOURCC_U8 output image.</td>
</tr>
</tbody>
</table>

See Also

- **Function: Integral Image**

### VX_KERNEL_ERODE_3x3
The erode kernel.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>The FOURCC_U8 input image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The FOURCC_U8 output image.</td>
</tr>
</tbody>
</table>

See Also

- **Function: Dilate Image**

### VX_KERNEL_MEDIAN_3x3
The median image filter.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>The FOURCC_U8 input image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The FOURCC_U8 output image.</td>
</tr>
</tbody>
</table>

See Also

- **Function: Median Filter**

### VX_KERNEL_BOX_3x3
The box filter kernel.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>The FOURCC_U8 input image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The FOURCC_U8 output image.</td>
</tr>
</tbody>
</table>

See Also

- **Function: Box Filter**

### VX_KERNEL_GAUSSIAN_3x3
The gaussian filter kernel.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>The FOURCC_U8 input image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The FOURCC_U8 output image.</td>
</tr>
</tbody>
</table>

See Also

- **Function: Gaussian Filter**

### VX_KERNEL_CUSTOM_CONVOLUTION
The custom convolution kernel.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>The FOURCC_U8 input image.</th>
</tr>
</thead>
</table>
### vx.convolution

**in** vx.convolution The vx.int16 symmetric matrix.

**out** vx.image The FOURCC_S16 output image.

**See Also**

Function: Custom Convolution

### VX KERNEL GAUSSIAN_PYRAMID

The gaussian image pyramid kernel.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx.image</th>
<th>The input FOURCC_U8 image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vx.pyramid</td>
<td>The pyramid object with the defined number of levels.</td>
</tr>
</tbody>
</table>

**See Also**

Function: Gaussian Image Pyramid

### VX KERNEL ACCUMULATE

The accumulation kernel.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx.image</th>
<th>The input FOURCC_U8 image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>vx.image</td>
<td>The FOURCC_U16 accumulation image.</td>
</tr>
</tbody>
</table>

**See Also**

Function: Accumulate

### VX KERNEL ACCUMULATE WEIGHTED

The weighted accumulation kernel.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx.image</th>
<th>The input FOURCC_U8 image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vx.scalar</td>
<td>The input VX_TYPE_FLOAT32 alpha value with the range $0.0 \leq \alpha \leq 1.0$.</td>
</tr>
<tr>
<td>in,out</td>
<td>vx.image</td>
<td>The FOURCC_U16 accumulation image.</td>
</tr>
</tbody>
</table>

**See Also**

Function: Accumulate Weighted

### VX KERNEL ACCUMULATE SQUARE

The squared accumulation kernel.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx.image</th>
<th>The input FOURCC_U8 image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vx.scalar</td>
<td>The input VX_TYPE_FLOAT32 scalar with the range $0.0 \leq \text{scalar} \leq 1.0$.</td>
</tr>
<tr>
<td>in,out</td>
<td>vx.image</td>
<td>The FOURCC_U16 accumulation image.</td>
</tr>
</tbody>
</table>

**See Also**

Function: Accumulate Squared

### VX KERNEL MINMAXLOC

The min and max location kernel.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx.image</th>
<th>The input.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vx.scalar</td>
<td>The minimum value.</td>
</tr>
<tr>
<td>out</td>
<td>vx.scalar</td>
<td>The maximum value.</td>
</tr>
<tr>
<td>out</td>
<td>vx.array</td>
<td>The minimum locations (if the input image has several minimums, the kernel will return all of them).</td>
</tr>
<tr>
<td>out</td>
<td>vx.array</td>
<td>The maximum locations (if the input image has several maximums, the kernel will return all of them).</td>
</tr>
</tbody>
</table>
CHAPTER 3. MODULE DOCUMENTATION

\begin{tabular}{|c|c|p{5.5in}|}
\hline
out & vx\_scalar & The total number of detected minimums in image (optional) \\
out & vx\_scalar & The total number of detected maximums in image (optional) \\
\hline
\end{tabular}

See Also

Function: Min, Max Location

**VX\_KERNEL\_CONVERTDEPTH** The bit-depth conversion kernel.

Parameters

\begin{tabular}{|c|p{5.5in}|}
\hline
in & vx\_image & The input image. Formats include FOURCC\_U8, FOURCC\_U16. \\
out & vx\_image & The output image. Formats include FOURCC\_U8, FOURCC\_U16. \\
in & vx\_scalar & The enumeration of the vx\_convert\_policy\_e. \\
in & vx\_scalar & The vx\_int32 shift value. \\
\hline
\end{tabular}

See Also

Function: Convert Bit depth

**VX\_KERNEL\_CANNY\_EDGE\_DETECTOR** The Canny Edge Detector.

Parameters

\begin{tabular}{|c|p{5.5in}|}
\hline
in & vx\_image & The input FOURCC\_U8 image. \\
in & vx\_threshold & The double threshold for hysteresis. \\
out & vx\_image & The output image in FOURCC\_U8 format. \\
\hline
\end{tabular}

See Also

Function: Canny Edge Detector

**VX\_KERNEL\_AND** The Bitwise And Kernel.

Parameters

\begin{tabular}{|c|p{5.5in}|}
\hline
in & vx\_image & Input image used as an operand. \\
in & vx\_image & Input image used as an operand. \\
out & vx\_image & The output image containing the result of the operation. \\
\hline
\end{tabular}

See Also

Function: Bitwise And

**VX\_KERNEL\_OR** The Bitwise Inclusive Or Kernel.

Parameters

\begin{tabular}{|c|p{5.5in}|}
\hline
in & vx\_image & Input image used as an operand. \\
in & vx\_image & Input image used as an operand. \\
out & vx\_image & The output image containing the result of the operation. \\
\hline
\end{tabular}

See Also

Function: Bitwise Inclusive Or

**VX\_KERNEL\_XOR** The Bitwise Exclusive Or Kernel.

Parameters

\begin{tabular}{|c|p{5.5in}|}
\hline
in & vx\_image & Input image used as an operand. \\
in & vx\_image & Input image used as an operand. \\
out & vx\_image & The output image containing the result of the operation. \\
\hline
\end{tabular}

See Also

Function: Bitwise Exclusive Or

**VX\_KERNEL\_NOT** The Bitwise Not Kernel.
**CHAPTER 3. MODULE DOCUMENTATION**

## Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>Input image used as the operand.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The output image containing the result of the operation.</td>
</tr>
</tbody>
</table>

See Also

- **Function: Bitwise Not**
- **VX_KERNEL_MULTIPLY** The Pixelwise Multiplication Kernel.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>Input image used as an operand.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vx_image</td>
<td>Input image used as an operand.</td>
</tr>
<tr>
<td>in</td>
<td>vx_scalar</td>
<td>A non-negative VX_TYPE_FLOAT32 scale multiplied to each product before overflow handling.</td>
</tr>
<tr>
<td>in</td>
<td>vx_enum</td>
<td>Overflow policy, an enumeration of the vx_convert_policy.e.</td>
</tr>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The output image containing the result of the operation.</td>
</tr>
</tbody>
</table>

See Also

- **Function: Pixel-wise Multiplication**
- **VX_KERNEL_ADD** The Addition Kernel.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>Input image used as an operand.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vx_image</td>
<td>Input image used as an operand.</td>
</tr>
<tr>
<td>in</td>
<td>vx_enum</td>
<td>Overflow policy, an enumeration of the vx_convert_policy.e.</td>
</tr>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The output image containing the result of the operation.</td>
</tr>
</tbody>
</table>

See Also

- **Function: Arithmetic Addition**
- **VX_KERNEL_SUBTRACT** The Subtraction Kernel.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>Input image used as an operand.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vx_image</td>
<td>Input image used as an operand.</td>
</tr>
<tr>
<td>in</td>
<td>vx_enum</td>
<td>Overflow policy, an enumeration of the vx_convert_policy.e.</td>
</tr>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The output image containing the result of the operation.</td>
</tr>
</tbody>
</table>

See Also

- **Function: Arithmetic Subtraction**
- **VX_KERNEL_WARP_AFFINE** The Warp Affine Kernel.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vx_image</th>
<th>The input image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vx_matrix</td>
<td>The 2x3 affine matrix.</td>
</tr>
<tr>
<td>in</td>
<td>vx_enum</td>
<td>The interpolation type from vx_interpolation_type.e.</td>
</tr>
<tr>
<td>out</td>
<td>vx_image</td>
<td>The output image.</td>
</tr>
</tbody>
</table>

See Also

- **Function: Warp Affine**
- **VX_KERNEL_WARP_PERSPECTIVE** The Warp Perspective Kernel.

### Parameters
CHAPTER 3. MODULE DOCUMENTATION

The input image.
in | vx_image       | The input image.
in | vx_matrix      | The 3x3 perspective matrix.
in | vx_enum        | The Interpolation type from vx_interpolation_type_e.

The output image.

See Also

Function: Warp Perspective

**VX_KERNEL_HARRIS_CORNERS**  The Harris Corners Kernel.

Parameters

| in   | vx_image       | The input image.
in | vx_scalar      | The sensitivity factor.
in | vx_scalar      | The minimum threshold which to eliminate Harris Corner scores.
in | vx_scalar      | The radial Euclidean distance for non-maximum suppression.
in | vx_scalar      | The VX_TYPE_FLOAT32 scalar sensitivity threshold \( k \) from the Harris--Stephens equation.
in | vx_scalar      | The gradient window size to use on the input. The implementation must support at least 3, 5, and 7.
in | vx_scalar      | The block window size used to compute the harris corner score. The implementation must support at least 3, 5, and 7.
out | vx_array       | The array of output corners.
out | vx_scalar      | The total number of detected corners in image (optional)

See Also

Function: Harris Corners

**VX_KERNEL_FAST_CORNERS**  The FAST Corners Kernel.

Parameters

| in   | vx_image       | input The input grayscale image (FOURCC_U8).
in | vx_scalar      | strength_thresh Threshold on difference between intensity of the central pixel and pixels on Bresenham's circle of radius 3 (VX_TYPE_FLOAT32 scalar)
in | vx_bool        | nonmax_supression If true, non-maximum suppression is applied to detected corners (keypoints)
out | vx_array       | corners Output corner array (vx_array of vx_keypoint_t)
out | vx_scalar      | num_corners The total number of detected corners in image (optional)

See Also

Function: Fast Corners

**VX_KERNEL_OPTICAL_FLOW_PYR_LK**  The Optical Flow Pyramid (LK) Kernel.

See Also

Function: Optical Flow Pyramid (LK)

**VX_KERNEL_REMAP**  The Remap Kernel.

Parameters

| in   | vx_image       | input The input image in FOURCC_U8 format.
in | vx_remap       | table The remap table.
in | vx_scalar      | policy The vx_interpolation_type_e type.
out | vx_image       | output The output image in FOURCC_U8 format.

See Also

Function: Remap

Definition at line 58 of file vx_kernels.h.
3.67.5 Function Documentation

`vx_kernel 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></th>
<th>context</th>
<th>The reference to the implementation context.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>kernel</td>
<td>A value from vx_kernel_e or a vendor or client defined value.</td>
</tr>
</tbody>
</table>

Returns

Returns a vx_kernel.

Return values

|   | 0 | The kernel enumeration was not found in the context. |

Precondition

vxLoadKernels if the kernel is not provided by the OpenVX implementation.

vx_kernel vxGetKernelByName ( vx_context context, vx_char * name )

Obtains a reference to a kernel using a string to specify the name.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>context</th>
<th>The reference to the implementation context.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>name</td>
<td>The string of the name of the kernel to get.</td>
</tr>
</tbody>
</table>

Returns

Returns a kernel reference or zero if an error occurred.

Return values

|   | 0 | The kernel name was not found in the context. |

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_status 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

<table>
<thead>
<tr>
<th></th>
<th>kernel</th>
<th>The kernel reference to query.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>attribute</td>
<td>The attribute to query. Use a vx_kernel_attribute_e.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The pointer to the location at which the resulting value will be stored.</td>
</tr>
<tr>
<td></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

|   | VX_SUCCESS | No errors |

void 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. |

Note

After returning from this function the reference will be zeroed.
3.68 Object: Parameter

3.68.1 Detailed Description

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_ATTRIBUTE_INDEX = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PARAMETER << 8)) + 0x0,
  VX_PARAMETER_ATTRIBUTE_DIRECTION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PARAMETER << 8)) + 0x1,
  VX_PARAMETER_ATTRIBUTE_TYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PARAMETER << 8)) + 0x2,
  VX_PARAMETER_ATTRIBUTE_STATE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PARAMETER << 8)) + 0x3,
  VX_PARAMETER_ATTRIBUTE_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 vxGetParameterByIndex (vx_node node, vx_uint32 index)
  
  Retrieves a vx_parameter from a vx_node.

- vx_status vxQueryParameter (vx_parameter param, vx_enum attribute, void *ptr, vx_size size)
  
  This allows the client to query a parameter to determine its meta-information.

- void 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 vxSetParameterByIndex (vx_node node, vx_uint32 index, vx_reference value)
  
  Sets the specified parameter data for a kernel on the node.
- **vx_status vxSetParameterByReference (vx_parameter parameter, vx_reference value)**
  Associates a parameter reference and a data reference with a kernel on a node.

### 3.68.2 Typedef Documentation

typedef struct _vx_parameter* vx_parameter

An opaque reference to a single parameter.

See Also

vxGetParameterByIndex

Definition at line 159 of file vx_types.h.

### 3.68.3 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 516 of file vx_types.h.

enum vx.parameter_attribute_e

The parameter attributes list.

**Enumerator**

- **VX_PARAMETER_ATTRIBUTE_INDEX** Used to query a parameter for its index value on the kernel it is associated with. Use a vx_uint32 parameter.
- **VX_PARAMETER_ATTRIBUTE_DIRECTION** Used to query a parameter for its direction value on the kernel it is associated with. Use a vx_enum parameter.
- **VX_PARAMETER_ATTRIBUTE_TYPE** Used to query a parameter for its size in bytes or if it is a vx_image or vx_array its vx_type_e will be returned. Use a vx_enum parameter.
- **VX_PARAMETER_ATTRIBUTE_STATE** Used to query a parameter for its state. A value in vx_parameter_state_e will be returned. Use a vx_enum parameter.
- **VX_PARAMETER_ATTRIBUTE_REF** Used to extract the reference contained in the parameter. Use a vx_reference parameter.

Definition at line 744 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 will be returned.
- **VX_PARAMETER_STATE_OPTIONAL** The parameter may be unspecified. The kernel will take care not to deference optional parameters until is is certain they valid.

Definition at line 1036 of file vx_types.h.

### 3.68.4 Function Documentation

vx_parameter vxGetParameterByIndex ( vx_node node, vx_uint32 index )

Retrieves a vx_parameter from a vx_node.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>node</th>
<th>The node to extract the parameter from.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>index</td>
<td>The index of the parameter to get a reference to.</td>
</tr>
</tbody>
</table>

Returns

\texttt{vx\_parameter}

\texttt{vx\_status vxQueryParameter ( vx\_parameter param, vx\_enum attribute, void * ptr, vx\_size size )}

This allows the client to query a parameter to determine its meta-information.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>param</th>
<th>The reference to the parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>attribute</td>
<td>The attribute to query. Use a \texttt{vx_parameter_attribute_e}.</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>The location at which the resulting value will be stored.</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 \texttt{vx\_status\_e} enumeration.

\texttt{void 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

| in  | param | The pointer to the parameter to release. |

Note

After returning from this function the reference will be zeroed.

\texttt{vx\_status 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>in</th>
<th>node</th>
<th>The node which contains the kernel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>index</td>
<td>The index of the parameter desired.</td>
</tr>
<tr>
<td>in</td>
<td>value</td>
<td>The reference to the parameter.</td>
</tr>
</tbody>
</table>

Returns

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

See Also

\texttt{vxSetParameterByReference}

\texttt{vx\_status vxSetParameterByReference ( vx\_parameter parameter, vx\_reference value )}

Associates a parameter reference and a data reference with a kernel on a node.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>parameter</th>
<th>The reference to the kernel parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>value</td>
<td>The value to associate with the kernel parameter.</td>
</tr>
</tbody>
</table>

Returns

A `vx_status_e` enumeration.

See Also

`vxGetParameterByIndex`
3.69 Advanced Framework API

3.69.1 Detailed Description

Components in this set are considered to be advanced. Advanced topics include extensions through Client Defined Functions, Reflection and Introspection, Performance Tweaking through Hinting and Directives, and Debugging Callbacks.

Modules

- **Framework: Client Defined Functions**
  
  *Client Defined Functions are a method to extend OpenVX with new vision functions.*

- **Framework: Directives**
  
  *The Directives Interface.*

- **Framework: Graph Parameters**
  
  *The Graph Parameter API.*

- **Framework: Hints**
  
  *The Hints Interface.*

- **Framework: Log**
  
  *The debug logging interface.*
CHAPTER 3. MODULE DOCUMENTATION

3.70 Framework: Log

3.70.1 Detailed Description

The debug logging interface. The functions of the debugging interface allow clients to receive important debugging information about OpenVX.

See Also

vx_status_e for the list of possible errors.

Figure 3.2: Log messages only can be received after the callback is installed.

Typedefs

- typedef void(*vx_log_callback_f)(vx_context context, vx_reference ref, vx_status status, vx_char string[])

The log callback function.

Functions

- void vxAddLogEntry(vx_reference ref, vx_status status, const char *message,...)

  Adds a line to the log.

- void vxRegisterLogCallback(vx_context context, vx_log_callback_f callback, vx_bool reentrant)

  Registers a callback facility to the OpenVX implementation to receive error logs.

3.70.2 Function Documentation

void vxAddLogEntry (vx_reference ref, vx_status status, const char *message, ...)

Adds a line to the log.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>ref</th>
<th>The reference to add the log entry against. Some valid value must be provided.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>status</td>
<td>The status code. VX_SUCCESS status entries will be ignored and not added.</td>
</tr>
<tr>
<td></td>
<td>message</td>
<td>The human readable message to add to the log.</td>
</tr>
<tr>
<td></td>
<td>...</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.

void 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></th>
<th>context</th>
<th>The overall context to OpenVX.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>callback</td>
<td>The callback function. If NULL, the previous callback is removed.</td>
</tr>
<tr>
<td></td>
<td>reentrant</td>
<td>If reentrancy flag is vx_true_e, then the callback may be entered from multi-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ple simultaneous tasks or threads (if the host OS supports this).</td>
</tr>
</tbody>
</table>
3.71 Framework: Hints

3.71.1 Detailed Description

The Hints Interface. *Hints* are messages given to the OpenVX implementation which it may support (are optional).

**Enumerations**

- `enum vx_hint_t {
  VX_HINT_SERIALIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_HINT << 12)) + 0x0
}

These enumerations are given to the vxHint API to enable/disable platform optimizations and/or features. Hints are optional and usually will be vendor specific.

**Functions**

- `vx_status vxHint (vx_context context, vx_reference reference, vx_enum hint)`

  A generic API to give platform specific hints to the implementation.

3.71.2 Enumeration Type Documentation

`enum vx_hint_t`

These enumerations are given to the vxHint API to enable/disable platform optimizations and/or features. Hints are optional and usually will be vendor specific.

See Also

- `vxHint`

**Enumerator**

`VX_HINT_SERIALIZE`  This indicates to the implementation that the user wants to disable any parallelization techniques. Implementations may not be parallelized, so this is a hint only.

Definition at line 531 of file vx_types.h.

3.71.3 Function Documentation

`vx_status vxHint ( vx_context context, vx_reference reference, vx_enum hint )`

A generic API to give platform specific hints to the implementation.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>context</code> The reference to the implementation context.</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>reference</code> The reference to the object to hint at. This could be <code>vx_context; vx_graph; vx_node; vx_image; vx_array;</code> or any other reference.</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>hint</code> A <code>vx_hint_t</code> &quot;hint&quot; to give the OpenVX context. This is a platform specific optimization or implementation mechanism.</td>
</tr>
</tbody>
</table>

**Returns**

A `vx_status_t` enumeration.

**Return values**

<table>
<thead>
<tr>
<th><code>VX_SUCCESS</code></th>
<th>No error.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>VX_ERROR_INVALID_REFERENCE</code></td>
<td>if context or reference are invalid.</td>
</tr>
<tr>
<td><strong>VX_ERROR_NOT_SUPPORTED</strong></td>
<td>if the hint is not supported.</td>
</tr>
</tbody>
</table>
### 3.72 Framework: Directives

#### 3.72.1 Detailed Description

The Directives Interface. *Directives* are messages given the OpenVX implementation which it must support (are non-optional).

**Enumerations**

- enum vx_directive_t {
  
  **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

  These enumerations are given to the vxDirective API to enable/disable platform optimizations and/or features. Directives are not optional and usually will be vendor specific, by defining a vendor range of directives and starting their enumeration from there.

**Functions**

- vx_status vxDirective (vx_context context, vx_reference reference, vx_enum directive)

  A generic API to give platform specific directives to the implementations.

#### 3.72.2 Enumeration Type Documentation

**enum vx_directive_t**

These enumerations are given to the vxDirective API to enable/disable platform optimizations and/or features. Directives are not optional and usually will be 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.

Definition at line 546 of file vx_types.h.

#### 3.72.3 Function Documentation

**vx_status vxDirective ( vx_context context, vx_reference reference, vx_enum directive )**

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>context</td>
<td>The reference to the implementation context.</td>
</tr>
<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.</td>
</tr>
</tbody>
</table>

**Returns**

A **vx_status_t** enumeration.
Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>No error.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>If context or reference are invalid.</td>
</tr>
<tr>
<td>VX_ERROR_NOT_SUPPORTED</td>
<td>If the directive is not supported.</td>
</tr>
</tbody>
</table>
3.73 Framework: Client Defined Functions

3.73.1 Detailed Description

Client Defined Functions are a method to extend OpenVX with new vision functions. Client Defined Functions can be loaded by OpenVX and included as nodes in the graph or as immediate functions (if the Client supplies the interface). Client Defined Functions will typically be loaded and executed on HLOS/CPU compatible targets, not remote processors or other accelerators. This specification does not mandate what constitutes compatible platforms.

![Call sequence of CDF Installation](image)

Figure 3.3: Call sequence of CDF Installation
Figure 3.4: Call sequence of a Graph Verify and Release with Client Defined Functions.

Figure 3.5: Call sequence of a Graph Execution with Client Defined Functions

**Typedefs**

- typedef vx_status(*(vx_kernel_deinitialize_f)(vx_node node, 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 will be called, if not NULL.
CHAPTER 3. MODULE DOCUMENTATION

• typedef vx_status(vx_node node, vx_reference*parameters, vx_uint32 num)
  The pointer to the Host side kernel.

• typedef vx_status(vx_node node, 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 will be called, if not NULL.

• typedef vx_status(vx_node node, vx_uint32 index)
  The user defined kernel node input parameter validation function.

• typedef vx_status(vx_node node, vx_uint32 index, vx_meta_format meta)
  The user defined kernel node output parameter validation function. The function only needs to fill in the meta data structure.

• typedef struct vx_meta_format*vx_meta_format
  This structure is used to extract meta data from a validation function. If the data object between nodes is virtual, this will allow the framework to automatically create the data object, if needed.

• typedef vx_status(vx_context context)
  The entry point into modules loaded by vxLoadKernels.

Functions

• vx_kernel vxAddKernel(vx_context context, vx_char name[ VX_MAX_KERNEL_NAME], vx_enum enumeration, vx_kernel_f func_ptr, vx_uint32 numParams, vx_kernel_input_validate_f input, vx_kernel_output_validate_f output, vx_kernel_initialize_f init, vx_kernel_deinitialize_f deinit)
  This API allows users to add custom kernels to the known kernel database in OpenVX at runtime. This would primarily be used by the module function vxPublishKernels.

• vx_status vxAddParameterToKernel(vx_kernel kernel, vx_uint32 index, vx_enum dir, vx_enum data_type, vx_enum state)
  This API allows users to set the signatures of the custom kernel.

• vx_status 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.

• vx_status vxLoadKernels(vx_context context, vx_char*module)
  Loads one or more kernels into the OpenVX context. This is the interface by which OpenVX is extensible. Once the set of kernels is loaded new kernels and their parameters can be queried.

• vx_status vxRemoveKernel(vx_kernel kernel)
  Removes a non-finalized vx_kernel from the vx_context. Once a vx_kernel has been finalized it can not be removed.

• vx_status vxSetKernelAttribute(vx_kernel kernel, vx_enum attribute, void*ptr, vx_size size)
  The interface to set kernel attributes.

3.73.2 Typedef Documentation

typedef vx_status(vx_node node, vx_reference*parameters, vx_uint32 num)

The pointer to the kernel deinitialize. If the host code requires a call to deinitialize data during a node garbage collection, this function will be called, if not NULL.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th></th>
<th>node</th>
<th>The handle to the node which contains this kernel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td></td>
<td>parameters</td>
<td>The array of parameter references.</td>
</tr>
<tr>
<td>in</td>
<td></td>
<td>num</td>
<td>The number of parameters.</td>
</tr>
</tbody>
</table>

Definition at line 1166 of file vx_types.h.

typedef vx_status(vx_node node, vx_reference*parameters, vx_uint32 num)

The pointer to the Host side kernel.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>node</td>
<td>The handle to the node which contains this kernel.</td>
</tr>
<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 1144 of file vx_types.h.

typedef vx_status( ∗ vx_kernel_initialize_f)(vx_node node, 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 will be called, if not NULL.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>node</td>
<td>The handle to the node which contains this kernel.</td>
</tr>
<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 1155 of file vx_types.h.

typedef vx_status( ∗ vx_kernel_input_validate_f)(vx_node node, vx_uint32 index)

The user defined kernel node input parameter validation function.

Note

This function will be called once for each VX_INPUT or VI_BIDIRECTIONAL parameter index.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>node</td>
<td>The handle to the node which is being validated.</td>
</tr>
<tr>
<td>in</td>
<td>index</td>
<td>The index of the parameter being validated.</td>
</tr>
</tbody>
</table>

Returns

Returns an error code describing the validation status on this parameter.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_FORMAT</td>
<td>The parameter format was incorrect.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_VALUE</td>
<td>The value of the parameter was incorrect.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_DIMENSION</td>
<td>The dimensionality of the parameter was incorrect.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>The index was out of bounds.</td>
</tr>
</tbody>
</table>

Definition at line 1190 of file vx_types.h.

typedef vx_status( ∗ vx_kernel_output_validate_f)(vx_node node, vx_uint32 index, vx_meta_format meta)

The user defined kernel node output parameter validation function. The function only needs to fill in the meta data structure.

Note

This function will be called once for each VX_OUTPUT parameter index.

Parameters
in  node  The handle to the node which is being validated.
in  index  The index of the parameter being validated.
in  ptr  A pointer to a preallocated structure that the system holds. The validation function will fill in the correct type, format and dimensionality for the system to use either create memory or check against existing memory.

Returns

Returns an error code describing the validation status on this parameter.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_PARAMETERS</td>
<td>The index was out of bounds.</td>
</tr>
</tbody>
</table>

Definition at line 1206 of file vx_types.h.

typedef vx_status(vx_publish_kernels_f)(vx_context context)

The entry point into modules loaded by vxLoadKernels.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>The handle to the implementation context.</td>
</tr>
</tbody>
</table>

Note

The symbol exported from the user module must be "vxPublishKernels" in extern "C" format.

Definition at line 1135 of file vx_types.h.

3.73.3 Function Documentation

vx_kernel vxAddKernel (vx_context context, vx_char name[VX_MAX_KERNEL_NAME], vx_enum enumeration, vx_kernel_f func_ptr, vx_uint32 numParams, vx_kernel_input_validate_f input, vx_kernel_output_validate_f output, vx_kernel_initialize_f init, vx_kernel_deinitialize_f deinit)

This API allows users to add custom kernels to the known kernel database in OpenVX at runtime. This would primarily be used by the module function vxPublishKernels.

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>name</td>
<td>The string which is to be used to match the kernel.</td>
</tr>
<tr>
<td>enumeration</td>
<td>The enumerated value of the kernel to be used by clients.</td>
</tr>
<tr>
<td>func_ptr</td>
<td>The process-local function pointer to be invoked</td>
</tr>
<tr>
<td>numParams</td>
<td>The number of parameters for this kernel.</td>
</tr>
<tr>
<td>input</td>
<td>The pointer to vx_kernel_input_validate_f which will validate the input parameters to this kernel.</td>
</tr>
<tr>
<td>output</td>
<td>The pointer to vx_kernel_output_validate_f which will validate the output parameters to this kernel.</td>
</tr>
<tr>
<td>init</td>
<td>The kernel initialization function.</td>
</tr>
<tr>
<td>deinit</td>
<td>The kernel de-initialization function.</td>
</tr>
</tbody>
</table>

Returns

vx_kernel

Return values
0 | Indicates that an error occurred when adding the kernel.
+ | Kernel added to OpenVX.

vx_status vxAddParameterToKernel ( vx_kernel kernel, vx_uint32 index, vx_enum dir, vx_enum data_type, vx_enum state )

This API allows users to set the signatures of the custom kernel.

Parameters

| in | kernel | The reference to the kernel added with `vxAddKernel`. |
| in | index | The index of the parameter to add. |
| in | dir | The direction of the parameter. This must be a value from `vx_direction_e`. |
| 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

| VX_SUCCESS | Parameter set on kernel. |
| VX_ERROR_INVALIDREFERENCE | The value passed as kernel was not a vx_kernel. |

Precondition

`vxAddKernel`

vx_status 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.

Parameters

| in | kernel | The reference to the loaded kernel from `vxAddKernel`. |

Returns

A `vx_status_e` enumeration. If an error occurs, the kernel will not be available for usage by the clients of OpenVX. Typically this is due to a mismatch between the number of parameter requested and given.

Precondition

`vxAddKernel` and `vxAddParameterToKernel`

vx_status vxLoadKernels ( vx_context context, vx_char * module )

Loads one or more kernels into the OpenVX context. This is the interface by which OpenVX is extensible. Once the set of kernels is loaded new kernels and their parameters can be queried.

Note

When all references to loaded kernels are released, the module may be automatically unloaded.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in context</td>
<td>The reference to the implementation context.</td>
</tr>
<tr>
<td>in 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: &quot;libxyz.so&quot; should be passed as just &quot;xyz&quot; and the implementation will &quot;do the right thing&quot; that the platform requires.</td>
</tr>
</tbody>
</table>

Note

This API will use the system pre-defined paths for modules.

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>Code</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 context is not a vx_context.</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 vxRemoveKernel ( vx_kernel kernel )

Removes a non-finalized vx_kernel from the vx_context. Once a vx_kernel has been finalized it can not be removed.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in kernel</td>
<td>The reference to the kernel to be removed. Returned from vxAddKernel.</td>
</tr>
</tbody>
</table>

Note

Any kernel enumerated in the base standard can not be removed. Only kernels added through vxAddKernel can be removed.

Returns

A vx_status_e enumeration.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>if an invalid kernel was passed in.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETER</td>
<td>if a base kernel was passed in.</td>
</tr>
</tbody>
</table>

vx_status vxSetKernelAttribute ( vx_kernel kernel, vx_enum attribute, void * ptr, vx_size size )

The interface to set kernel attributes.

Parameters
in      kernel          The reference to the kernel.
in      attribute       The enumeration of the attributes. See vx_kernel_attribute_e.
in      ptr             The pointer to the location to read the attribute from.
in      size            The size of the data area indicated by ptr in bytes.

Note

After a kernel has been passed to vxFinalizeKernel, no attributes can be altered.

Returns

A vx_status_e enumeration.
CHAPTER 3. MODULE DOCUMENTATION

3.74 Framework: Graph Parameters

3.74.1 Detailed Description

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 typically use the "manual" method via \texttt{vxCreateNode}.

```c
vx_graph vxCornersGraphFactory(vx_context context) {  
  vx_status status = VX_SUCCESS;
  vx_uint32 i;
  vx_float32 strength_thesh = 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);
  if (graph) {  
    vx_image virts[] = {
      vxCreateVirtualImage(graph, 0, 0, FOURCC_VIRT),
      vxCreateVirtualImage(graph, 0, 0, FOURCC_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)] = {
      vxCreateNode(graph, kernels[0]),
      vxCreateNode(graph, kernels[1]),
      vxCreateNode(graph, kernels[2]),
    };
    vx_scalar scalars[] = {
      vxCreateScalar(context, VX_TYPE_ENUM, &channel),
      vxCreateScalar(context, VX_TYPE_FLOAT32, &strength_thesh),
      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]);
    status |= vxAddParameterToGraph(graph, parameters[1]);
    for (i = 0; i < dimof(scalars); i++) {  
      vxReleaseScalar(&scalars[i]);
    }
    for (i = 0; i < dimof(virts); i++) {  
      vxReleaseImage(&virts[i]);
    }
  }
}
```
for (i = 0; i < dimof(kernels); i++)
{
    vxReleaseKernel(&kernels[i]);
}
for (i = 0; i < dimof(nodes);i++)
{
    vxReleaseNode(&nodes[i]);
}
for (i = 0; i < dimof(parameters); i++)
{
    vxReleaseParameter(&parameters[i]);
}
return graph;
}

Some data is contained in these Graphs and does not become exposed to Clients of the factory. This allows ISVs or Vendors to create custom IP or IP sensitive factories which Clients can use but may not be able to determine what is inside the factory. Since the graph contains internal references to the data, the objects will not be freed until the graph itself is released.

Functions

- **vx_status vxAddParameterToGraph (vx_graph graph, vx_parameter parameter)**
  
  Add the given parameter extracted from a **vx_node** to the graph.

- **vx_parameter vxGetGraphParameterByIndex (vx_graph graph, vx_uint32 index)**
  
  Retrieves a **vx_parameter** from a **vx_graph**.

- **vx_status 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.

### 3.74.2 Function Documentation

**vx_status vxAddParameterToGraph ( vx_graph graph, vx_parameter parameter )**

Add 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 which 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**

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 added to Graph.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>The parameter was not a valid <strong>vx_parameter</strong></td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETER</td>
<td>The parameter was of a node not in this graph.</td>
</tr>
</tbody>
</table>

**vx_parameter 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>
</table>
in | index | The index of the parameter.

Returns

Returns `vx_parameter` reference.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>If the index was out of bounds.</td>
</tr>
<tr>
<td>*</td>
<td>The parameter reference.</td>
</tr>
</tbody>
</table>

`vx_status 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

Returns a `vx_status` enumeration.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_SUCCESS</td>
<td>Parameter set to Graph.</td>
</tr>
<tr>
<td>VX_ERROR_INVALID_REFERENCE</td>
<td>The value was not a valid <code>vx_reference</code></td>
</tr>
<tr>
<td>VX_ERROR_INVALID_PARAMETER</td>
<td>The parameter index was out of bounds or the dir parameter was incorrect</td>
</tr>
</tbody>
</table>
Bibliography


Index

Advanced Features, 190
Advanced Framework API, 217
Advanced Objects, 191

Basic Features, 102
FOURCC_IYUV, 111
FOURCC_NV12, 111
FOURCC_NV21, 111
FOURCC_RGB, 111
FOURCC_RGBA, 111
FOURCC_S16, 111
FOURCC_S32, 111
FOURCC_U16, 111
FOURCC_U32, 111
FOURCC_U8, 111
FOURCC_UYVY, 111

e CHANNEL 0, 110
FOURCC VIRT, 111
FOURCC_YUV, 111
FOURCC_YUV4, 111
FOURCC_YUVV, 111
VX_CHANNEL_0, 110
VX_CHANNEL_1, 110
VX_CHANNEL_2, 110
VX_CHANNEL_3, 110
VX_CHANNEL_A, 110
VX_CHANNEL_B, 110
VX_CHANNEL_C, 110
VX_CHANNEL_D, 110
VX_CHANNEL_E, 110
VX_CHANNEL_F, 110
VX_CONVERT_POLICY_SATURATE, 110
VX_CONVERT_POLICY_TRUNCATE, 110
VX_ENUM_ACCESSOR, 111
VX_ENUM_ACTION, 110
VX_ENUM_BORDER_MODE, 110
VX_ENUM_CHANNEL, 110
VX_ENUM_COLOR_RANGE, 110
VX_ENUM_COLOR_SPACE, 110
VX_ENUM_COMPARISON, 110
VX_ENUM_CONVERT_POLICY, 110
VX_ENUM_DIRECTION, 110
VX_ENUM_DIRECTIVE, 110
VX_ENUM_HINT, 110
VX_ENUM_IMPORT_MEM, 110
VX_ENUM_INTERPOLATION, 110
VX_ENUM_NORM_TYPE, 111
VX_ENUM_OVERFLOW, 110
VX_ENUM_PARAMETER_STATE, 110
VX_ENUM_ROUND_POLICY, 111
VX_ENUM_TERM_CRITERIA, 111

VX_ENUM_THRESHOLD_TYPE, 110
VX_ERROR_GRAPH_ABANDONED, 112
VX_ERROR_GRAPH_SCHEDULED, 112
VX_ERROR_INVALID_DIMENSION, 112
VX_ERROR_INVALID_FORMAT, 112
VX_ERROR_INVALID_GRAPH, 112
VX_ERROR_INVALID_LINK, 112
VX_ERROR_INVALID_MODULE, 112
VX_ERROR_INVALID_NODE, 112
VX_ERROR_INVALID_PARAMETERS, 112
VX_ERROR_INVALID_REFERENCE, 112
VX_ERROR_INVALID_SCOPE, 112
VX_ERROR_INVALID_TYPE, 112
VX_ERROR_INVALID_VALUE, 112
VX_ERROR_MULTIPLE_WRITERS, 112
VX_ERROR_NO_MEMORY, 113
VX_ERROR_NO_RESOURCES, 113
VX_ERROR_NOT_ALLOCATED, 113
VX_ERROR_NOT_COMPATIBLE, 113
VX_ERROR_NOT_IMPLEMENTED, 113
VX_ERROR_NOT_SUFFICIENT, 113
VX_ERROR_NOT_SUPPORTED, 113
VX_ERROR_OPTIMIZED_AWAY, 113
VX_ERROR_REFERENCE_NONZERO, 112
VX_FAILURE, 113
VX_ID_AMD, 115
VX_ID_ARM, 114
VX_ID_AXIS, 115
VX_ID_BDTI, 114
VX_ID_BROADCOM, 115
VX_ID_CEVA, 115
VX_ID_DEFAULT, 115
VX_ID_Freescale, 115
VX_ID_INTEL, 115
VX_ID_KHRONOS, 114
VX_ID_MARVELL, 115
VX_ID_MEDIATEK, 115
VX_ID_MOVIDIUS, 115
VX_ID_NVIDIA, 114
VX_ID_QUALCOMM, 114
VX_ID_RENESAS, 114
VX_ID_SAMSUNG, 115
VX_ID_ST, 115
VX_ID_TI, 114
VX_ID_VIVANTE, 114
VX_ID_XILINX, 115
VX_INTERPOLATION_TYPE_AREA, 112
VX_INTERPOLATION_TYPE_BILINEAR, 112

235
INDEX

VX_INTERPOLATION_TYPE_NEARNEIGHBOR, 112
VX_STATUS_MIN, 112
VX_SUCCESS, 113
VX_TYPE_ARRAY, 114
VX_TYPE_BOOL, 114
VX_TYPE_CHAR, 113
VX_TYPE_CONTEXT, 114
VX_TYPE_CONVOLUTION, 114
VX_TYPE_COORDINATES2D, 114
VX_TYPE_COORDINATES3D, 114
VX_TYPE_DELAY, 114
VX_TYPE_DISTRIBUTION, 114
VX_TYPE_ENUM, 114
VX_TYPE_ERROR, 114
VX_TYPE_FLOAT32, 113
VX_TYPE_FLOAT64, 114
VX_TYPE_FOURCC, 114
VX_TYPE_GRAPH, 114
VX_TYPE_IMAGE, 114
VX_TYPE_INT16, 113
VX_TYPE_INT32, 113
VX_TYPE_INT64, 113
VX_TYPE_INT8, 113
VX_TYPE_INVALID, 113
VX_TYPE_KERNEL, 114
VX_TYPE_KEYPOINT, 114
VX_TYPE_LUT, 114
VX_TYPE_MATRIX, 114
VX_TYPE_META_FORMAT, 114
VX_TYPE_NODE, 114
VX_TYPE_OBJECT_MAX, 114
VX_TYPE_PARAMETER, 114
VX_TYPE_PYRAMID, 114
VX_TYPE_RECTANGLE, 114
VX_TYPE_REFNUM, 114
VX_TYPE_REMAP, 114
VX_TYPE_SCALAR, 114
VX_TYPE_SCALAR_MAX, 114
VX_TYPE_SIZE, 114
VX_TYPE_STRUCT_MAX, 114
VX_TYPE_THRESHOLD, 114
VX_TYPE_UINT16, 113
VX_TYPE_UINT32, 113
VX_TYPE_UINT64, 113
VX_TYPE_UINT8, 113
VX_CHANNEL_E, 109
VX_CHANNELF, 109
VX_CONVERT_POLICY_E, 110
VX_CONVERT_POLICYF, 110
VX_ENUM, 109
VX_ENUMF, 110
VX_FOURCC_E, 111
VX_FOURCCF, 111
VX_INTERPOLATION_TYPE_E, 111
VX_INTERPOLATION_TYPEF, 111
VX_STATUS, 109
VX_STATUSF, 112
VX_TYPE_E, 113
VX_VENDOR_ID_E, 114
VX_VENDOR_IDF, 114
vxGetStatus, 115

Basic Framework, 184
FOURCC_JYUV
Basic Features, 111
FOURCC_NV12
Basic Features, 111
FOURCC_NV21
Basic Features, 111
FOURCC_RGB
Basic Features, 111
FOURCC_RGBX
Basic Features, 111
FOURCC_S16
Basic Features, 111
FOURCC_S32
Basic Features, 111
FOURCC_U16
Basic Features, 111
FOURCC_U32
Basic Features, 111
FOURCC_U8
Basic Features, 111
FOURCC_UYYV
Basic Features, 111
FOURCC_VIRT
Basic Features, 111
FOURCC_YUV4
Basic Features, 111
FOURCC_YUVV
Basic Features, 111

Framework: Directives
VX_DIRECTIVE_DISABLE_LOGGING, 221
VX_DIRECTIVE_ENABLE_LOGGING, 221

Framework: Hints
VX_HINT_SERIALIZE, 219

Framework: Node Callbacks
VX_ACTION_ABANDON, 187
VX_ACTION_CONTINUE, 187
VX_ACTION_RESTART, 187

Framework: Client Defined Functions, 223
vx_kernel_deinitKernel, 225
vx_kernel, 225
vx_kernel_initiate, 226
vx_kernel_input_validate, 226
vx_kernel_output_validate, 226
vx_publish_kernels, 227
vxAddKernel, 227
vxAddParameterToKernel, 228
vxFinalizeKernel, 228
vxLoadKernels, 228
vxRemoveKernel, 229
vxSetKernelAttribute, 229
Framework: Directives, 221
vx Directive, 221
Framework: Graph Parameters, 231
vxAddParameterToGraph, 232
vxGetGraphParameterByIndex, 232
vxSetGraphParameterByIndex, 233
Framework: Hints, 219
vx Hint, 219
Framework: Log, 218
vxAddLogEntry, 218
vxRegisterLogCallback, 218
Framework: Node Callbacks, 185
vx Action, 187
vx Action, 187
vxNodeComplete, 187
vxAssignNodeCallback, 187
vxRetrieveNodeCallback, 188
Framework: Performance Measurement, 189
Function: Canny Edge Detector
VX NORM L1, 46
VX NORM L2, 46
Function: Absolute Difference, 27
vxAbsDiffNode, 27
vxuAbsDiff, 27
Function: Accumulate, 28
vxAccumulateImageNode, 28
vxuAccumulateImage, 28
Function: Accumulate Squared, 29
vxAccumulateSquareImageNode, 29
vxuAccumulateSquareImage, 29
Function: Accumulate Weighted, 31
vxAccumulateWeightedImageNode, 31
vxuAccumulateWeightedImage, 31
Function: Arithmetic Addition, 33
vxAddNode, 33
vxuAdd, 33
Function: Arithmetic Subtraction, 35
vxSubtractNode, 35
vxuSubtract, 35
Function: Bitwise And, 37
vxAndNode, 37
vxuAnd, 37
Function: Bitwise Exclusive Or, 39
vxXorNode, 39
vxuXor, 39
Function: Bitwise Inclusive Or, 41
vxOrNode, 41
vxuOr, 41
Function: Bitwise Not, 43
vxNotNode, 43
vxuNot, 43
Function: Box Filter, 44
vxBox3x3Node, 44
vxuBox3x3, 44
Function: Canny Edge Detector, 45
vx norm type e, 46
vxCannyEdgeDetectorNode, 46
vxuCannyEdgeDetector, 46
Function: Channel Combine, 48
vxChannelCombineNode, 48
vxuChannelCombine, 48
Function: Channel Extract, 50
vxChannelExtractNode, 50
vxuChannelExtract, 50
Function: Color Convert, 52
vxColorConvertNode, 54
vxuColorConvert, 54
Function: Convert Bit depth, 56
vxConvertDepthNode, 56
vxuConvertDepth, 57
Function: Custom Convolution, 58
vxConvolveNode, 58
vxuConvolve, 59
Function: Dilate Image, 60
vxDilate3x3Node, 60
vxuDilate3x3, 60
Function: Equalize Histogram, 61
vxuEqualizeHist, 61
Function: Erode Image, 62
vxuErode3x3Node, 62
vxuErode3x3, 62
Function: Fast Corners, 63
vxFastCornersNode, 64
vxuFastCorners, 64
Function: Gaussian Filter, 66
vxuGaussian3x3, 66
Function: Gaussian Image Pyramid, 71
vxuGaussianPyramid, 71
Function: Harris Corners, 67
vxuHarrisCornersNode, 68
vxuHarrisCorners, 68
Function: Histogram, 70
vxuHistogramNode, 70
vxuHistogram, 70
Function: Integral Image, 73
vxIntegralImageNode, 73
vxuIntegralImage, 73
Function: Magnitude, 74
vxuMagnitudeNode, 74
vxuMagnitude, 74
Function: Mean and Standard Deviation, 76
vxuMeanStdDevNode, 76
vxuMeanStdDev, 76
Function: Median Filter, 78
vxMedian3x3Node, 78
vxuMedian3x3, 78
Function: Min, Max Location, 79
INDEX

VX_KERNEL_AND, 207
VX_KERNEL_ATTRIBUTE_ENUM, 201
VX_KERNEL_ATTRIBUTE_LOCAL_DATA_PTR, 201
VX_KERNEL_ATTRIBUTE_LOCAL_DATA_SIZE, 201
VX_KERNEL_ATTRIBUTE_NAME, 201
VX_KERNEL_ATTRIBUTE_NUMPARAMS, 201
VX_KERNEL_BOX_3x3, 205
VX_KERNEL_CANNY_EDGE_DETECTOR, 207
VX_KERNEL_CHANNEL_COMBINE, 202
VX_KERNEL_CHANNEL_EXTRACT, 202
VX_KERNEL_COLOR_CONVERT, 202
VX_KERNEL_CONVERTDEPTH, 207
VX_KERNEL_CUSTOM_CONVOLUTION, 205
VX_KERNEL_DILATE_3x3, 205
VX_KERNEL_EQUALIZE_HISTOGRAM, 204
VX_KERNEL_ERODE_3x3, 205
VX_KERNEL_FAST_CORNERS, 209
VX_KERNEL_GAUSSIAN_3x3, 205
VX_KERNEL_GAUSSIAN_PYRAMID, 206
VX_KERNEL_HARRIS_CORNERS, 209
VX_KERNEL_HISTOGRAM, 204
VX_KERNEL_INTEGRAL_IMAGE, 204
VX_KERNEL_INVALID, 202
VX KERNEL_MAGNITUDE, 203
VX_KERNEL_MEAN_STDDEV, 204
VX_KERNEL_MEDIAN_3x3, 205
VX_KERNEL_MINMAXLOC, 206
VX_KERNEL_MULTIPLY, 208
VX_KERNEL_NOT, 207
VX_KERNEL_OPTICAL_FLOW_PYR_LK, 209
VX_KERNEL_OR, 207
VX_KERNEL_PHASE, 203
VX_KERNEL_REMAP, 209
VX_KERNEL_SCALE_IMAGE, 203
VX_KERNEL_SOBEL_3x3, 205
VX_KERNEL_SUBTRACT, 208
VX_KERNEL_TABLE_LOOKUP, 203
VX_KERNEL_THRESHOLD, 204
VX_KERNEL_WARP_AFFINE, 208
VX_KERNEL_WARP_PERSPECTIVE, 208
VX_KERNEL_XOR, 207

Object: Parameter
VX_KERNEL_ATTRIBUTE_DIRECTION, 214
VX_KERNEL_ATTRIBUTE_INDEX, 214
VX_KERNEL_ATTRIBUTE_REF, 214
VX_KERNEL_ATTRIBUTE_STATE, 214
VX_KERNEL_ATTRIBUTE_TYPE, 214
VX_KERNEL_ATTRIBUTE_STATE_OPTIONAL, 214
VX_KERNEL_ATTRIBUTE_STATE_REQUIRED, 214

Object: Pyramid
VX_PYRAMID_ATTRIBUTE_FORMAT, 171
VX_PYRAMID_ATTRIBUTE_HEIGHT, 171
VX_PYRAMID_ATTRIBUTE_LEVELS, 171
VX_PYRAMID_ATTRIBUTE_SCALE, 171
VX_PYRAMID_ATTRIBUTE_WIDTH, 171

Object: Reference
VX_REF_ATTRIBUTE_COUNT, 117
VX_REF_ATTRIBUTE_TYPE, 117

Object: Remap
VX_REMAP_ATTRIBUTE_DESTINATION_HEIGHT, 174
VX_REMAP_ATTRIBUTE_DESTINATION_WIDTH, 174
VX_REMAP_ATTRIBUTE_SOURCE_HEIGHT, 174
VX_REMAP_ATTRIBUTE_SOURCE_WIDTH, 174

Object: Scalar
VX_SCALAR_ATTRIBUTE_TYPE, 178

Object: Threshold
VX_THRESHOLD_ATTRIBUTE_LOWER, 181
VX_THRESHOLD_ATTRIBUTE_TYPE, 181
VX_THRESHOLD_ATTRIBUTE_UPPER, 181
VX_THRESHOLD_ATTRIBUTE_VALUE, 181
VX_THRESHOLD_ATTRIBUTE_TYPE_BINARY, 182
VX_THRESHOLD_ATTRIBUTE_TYPE_RANGE, 182

Object: Array
vx_array_attribute_e, 135
vxAccessArrayRange, 135
vxAddArrayItems, 136
vxArrayItem, 134
vxCommitArrayRange, 136
vxCreateArray, 137
vxCreateVirtualArray, 137
vxCreateArrayPointer, 135
vxQueryArray, 138
vxReleaseArray, 138
vxTruncateArray, 138

Object: Array (Advanced)
vxRegisterUserStruct, 192

Object: Context
vx_context, 120
vx_context_attribute_e, 120
vx_import_type_e, 121
vxRound_policy_e, 121
vxtermination_criteria_e, 122
<table>
<thead>
<tr>
<th><strong>Object:</strong> Convolution, 140</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_getContext, 122</td>
</tr>
<tr>
<td>vx_getQueryContext, 123</td>
</tr>
<tr>
<td>vx_releaseContext, 123</td>
</tr>
<tr>
<td>vx_setContextAttribute, 123</td>
</tr>
<tr>
<td>vx_accessConvolution_attribute_e, 140</td>
</tr>
<tr>
<td>vx_commitConvolution_coefficients, 141</td>
</tr>
<tr>
<td>vx_createConvolution, 141</td>
</tr>
<tr>
<td>vx_queryConvolution, 142</td>
</tr>
<tr>
<td>vx_releaseConvolution, 142</td>
</tr>
<tr>
<td>vx_setConvolution_attribute, 142</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Object:</strong> Delay, 195</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_delay, 195</td>
</tr>
<tr>
<td>vx_delay_attribute_e, 196</td>
</tr>
<tr>
<td>vx_ageDelay, 196</td>
</tr>
<tr>
<td>vx_associateDelay_with_node, 196</td>
</tr>
<tr>
<td>vx_createDelay, 197</td>
</tr>
<tr>
<td>vx_dissociateDelay_from_node, 197</td>
</tr>
<tr>
<td>vx_queryDelay, 198</td>
</tr>
<tr>
<td>vx_releaseDelay, 198</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Object:</strong> Distribution, 143</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_distribution_attribute_e, 143</td>
</tr>
<tr>
<td>vx_accessDistribution, 144</td>
</tr>
<tr>
<td>vx_commitDistribution, 144</td>
</tr>
<tr>
<td>vx_createDistribution, 144</td>
</tr>
<tr>
<td>vx_queryDistribution, 145</td>
</tr>
<tr>
<td>vx_releaseDistribution, 145</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Object:</strong> Graph, 125</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_graph, 126</td>
</tr>
<tr>
<td>vx_graph_attribute_e, 126</td>
</tr>
<tr>
<td>vx_createGraph, 126</td>
</tr>
<tr>
<td>vx_isGraph_verified, 126</td>
</tr>
<tr>
<td>vx_processGraph, 127</td>
</tr>
<tr>
<td>vx_queryGraph, 127</td>
</tr>
<tr>
<td>vx_releaseGraph, 127</td>
</tr>
<tr>
<td>vx_scheduleGraph, 128</td>
</tr>
<tr>
<td>vx_setGraph_attribute, 128</td>
</tr>
<tr>
<td>vx_verifyGraph, 128</td>
</tr>
<tr>
<td>vx_waitGraph, 129</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Object:</strong> Image, 146</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_channel_range_e, 149</td>
</tr>
<tr>
<td>vx_color_space_e, 149</td>
</tr>
<tr>
<td>vx_image, 149</td>
</tr>
<tr>
<td>vx_image_attribute_e, 150</td>
</tr>
<tr>
<td>vx_accessImage_patch, 150</td>
</tr>
<tr>
<td>vx_commitImage_patch, 152</td>
</tr>
<tr>
<td>vx_computeImage_patch_size, 154</td>
</tr>
<tr>
<td>vx_createImage, 154</td>
</tr>
<tr>
<td>vx_createImage_from_handle, 154</td>
</tr>
<tr>
<td>vx_createImage_from_roi, 156</td>
</tr>
<tr>
<td>vx_createUniformImage, 156</td>
</tr>
<tr>
<td>vx_createVirtualImage, 157</td>
</tr>
<tr>
<td>vx_formatImage_patch_address_1d, 157</td>
</tr>
<tr>
<td>vx_formatImage_patch_address_2d, 159</td>
</tr>
<tr>
<td>vx_getValid_region_image, 160</td>
</tr>
<tr>
<td>vx_queryImage, 161</td>
</tr>
<tr>
<td>vx_releaseImage, 161</td>
</tr>
<tr>
<td>vx_setImage_attribute, 162</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Object:</strong> Kernel, 199</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_kernel, 201</td>
</tr>
<tr>
<td>vx_kernel_attribute_e, 201</td>
</tr>
<tr>
<td>vx_kernel_e, 201</td>
</tr>
<tr>
<td>vx_getKernel_by_enum, 210</td>
</tr>
<tr>
<td>vx_getKernel_by_name, 211</td>
</tr>
<tr>
<td>vx_queryKernel, 211</td>
</tr>
<tr>
<td>vx_releaseKernel, 212</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Object:</strong> LUT, 163</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_lut_attribute_e, 163</td>
</tr>
<tr>
<td>vx_accessLUT, 163</td>
</tr>
<tr>
<td>vx_commitLUT, 164</td>
</tr>
<tr>
<td>vx_createLUT, 164</td>
</tr>
<tr>
<td>vx_queryLUT, 164</td>
</tr>
<tr>
<td>vx_releaseLUT, 166</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Object:</strong> Matrix, 167</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_matrix_attribute_e, 167</td>
</tr>
<tr>
<td>vx_accessMatrix, 167</td>
</tr>
<tr>
<td>vx_commitMatrix, 168</td>
</tr>
<tr>
<td>vx_createMatrix, 168</td>
</tr>
<tr>
<td>vx_queryMatrix, 168</td>
</tr>
<tr>
<td>vx_releaseMatrix, 169</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Object:</strong> Node, 130</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_node, 130</td>
</tr>
<tr>
<td>vx_node_attribute_e, 131</td>
</tr>
<tr>
<td>vx_queryNode, 131</td>
</tr>
<tr>
<td>vx_releaseNode, 131</td>
</tr>
<tr>
<td>vx_removeNode, 131</td>
</tr>
<tr>
<td>vx_setNode_attribute, 132</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Object:</strong> Node (Advanced), 193</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_createNode, 193</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Object:</strong> Parameter, 213</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_direction_e, 214</td>
</tr>
<tr>
<td>vx_parameter, 214</td>
</tr>
<tr>
<td>vx_parameter_attribute_e, 214</td>
</tr>
<tr>
<td>vx_parameter_state_e, 214</td>
</tr>
<tr>
<td>vx_getParameter_by_index, 214</td>
</tr>
<tr>
<td>vx_queryParameter, 215</td>
</tr>
<tr>
<td>vx_releaseParameter, 215</td>
</tr>
<tr>
<td>vx_setParameter_by_index, 215</td>
</tr>
<tr>
<td>vx_setParameter_by_reference, 215</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Object:</strong> Pyramid, 170</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_pyramid_attribute_e, 171</td>
</tr>
<tr>
<td>vx_createPyramid, 171</td>
</tr>
<tr>
<td>vx_createVirtualPyramid, 171</td>
</tr>
<tr>
<td>vx_getPyramid_level, 172</td>
</tr>
<tr>
<td>vx_queryPyramid, 172</td>
</tr>
<tr>
<td>vx_releasePyramid, 172</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Object:</strong> Reference, 117</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_reference, 117</td>
</tr>
<tr>
<td>vx_reference_attribute_e, 117</td>
</tr>
<tr>
<td>vx_queryReference, 117</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Object:</strong> Remap, 174</th>
</tr>
</thead>
<tbody>
<tr>
<td>vx_remap_attribute_e, 174</td>
</tr>
<tr>
<td>vx_createRemap, 175</td>
</tr>
<tr>
<td>vx_getRemap_point, 176</td>
</tr>
</tbody>
</table>
VX_ID_AMD Basic Features, 115
VX_ID_ARM Basic Features, 114
VX_ID_AXIS Basic Features, 115
VX_ID_BDTI Basic Features, 114
VX_ID_BROADCOM Basic Features, 115
VX_ID_GEVA Basic Features, 115
VX_ID_DEFAULT Basic Features, 115
VX_ID_FREESCALE Basic Features, 115
VX_ID_INTEL Basic Features, 114
VX_ID_KHRONOS Basic Features, 115
VX_ID_MARVELL Basic Features, 115
VX_ID_MEDIATEK Basic Features, 115
VX_ID_Movidius Basic Features, 115
VX_IMAGE_ATTRIBUTE_FORMAT Object: Image, 150
VX_IMAGE_ATTRIBUTE_HEIGHT Object: Image, 150
VX_IMAGE_ATTRIBUTE_PLANES Object: Image, 150
VX_IMAGE_ATTRIBUTE_RANGE Object: Image, 150
VX_IMAGE_ATTRIBUTE_SIZE Object: Image, 150
VX_IMAGE_ATTRIBUTE_SPACE Object: Image, 150
VX_IMAGE_ATTRIBUTE_WIDTH Object: Image, 150
VX_IMPORT_TYPE_HOST Object: Context, 121
VX_IMPORT_TYPE_NONE Object: Context, 121
VX_INPUT Object: Parameter, 214
VX_KERNEL ABSDIFF Object: Kernel, 204
VX_KERNEL ACCUMULATE Object: Kernel, 206
VX_KERNEL ACCUMULATE SQUARE Object: Kernel, 206
VX_KERNEL ACCUMULATE WEIGHTED Object: Kernel, 206
VX_KERNEL ADD Object: Kernel, 208
VX_KERNEL AND Object: Kernel, 207
VX_KERNEL ATTRIBUTE ENUM Object: Kernel, 201
VX_KERNEL ATTRIBUTE LOCAL DATA PTR Object: Kernel, 201
VX_KERNEL ATTRIBUTE LOCAL DATA SIZE Object: Kernel, 201
VX_KERNEL ATTRIBUTE NAME Object: Kernel, 201
VX_KERNEL ATTRIBUTE NUMPARAMS Object: Kernel, 201
VX_KERNEL BOX 3x3 Object: Kernel, 205
VX_KERNEL CANNY EDGE DETECTOR Object: Kernel, 207
VX_KERNEL CHANNEL COMBINE Object: Kernel, 202
VX_KERNEL CHANNEL EXTRACT Object: Kernel, 202
VX_KERNEL COLOR CONVERT Object: Kernel, 202
VX_KERNEL CONV2DEPTH Object: Kernel, 207
VX_KERNEL CUSTOM_CONVOLUTION Object: Kernel, 207
VX_KERNEL DILATE 3x3 Object: Kernel, 205
VX_KERNEL EQUALIZE HISTOGRAM Object: Kernel, 204
VX_KERNEL ERODE 3x3 Object: Kernel, 205
VX_KERNEL FAST CORNERS Object: Kernel, 209
VX_KERNEL GAUSSIAN 3x3 Object: Kernel, 205
VX_KERNEL GAUSSIAN PYRAMID Object: Kernel, 206
INDEX

VX_KERNEL_HARRIS_CORNERS
Object: Kernel, 209

VX_KERNEL_HISTOGRAM
Object: Kernel, 204

VX_KERNEL_INTEGRAL_IMAGE
Object: Kernel, 204

VX_KERNEL_INVALID
Object: Kernel, 202

VX_KERNEL_MAGNITUDE
Object: Kernel, 203

VX_KERNEL_MEAN_STDDEV
Object: Kernel, 204

VX_KERNEL_MEDIAN_3x3
Object: Kernel, 205

VX_KERNEL_MINMAXLOC
Object: Kernel, 206

VX_KERNEL_MULTIPLY
Object: Kernel, 208

VX_KERNEL_NOT
Object: Kernel, 207

VX_KERNEL_OPTICAL_FLOW_PYR_LK
Object: Kernel, 209

VX_KERNEL_OR
Object: Kernel, 207

VX_KERNEL_PHASE
Object: Kernel, 203

VX_KERNEL_REMAP
Object: Kernel, 209

VX_KERNEL_SCALE_IMAGE
Object: Kernel, 203

VX_KERNEL_SOBEL_3x3
Object: Kernel, 203

VX_KERNEL_SUBTRACT
Object: Kernel, 208

VX_KERNEL_TABLE_LOOKUP
Object: Kernel, 203

VX_KERNEL_THRESHOLD
Object: Kernel, 204

VX_KERNEL_WARP_AFFINE
Object: Kernel, 208

VX_KERNEL_WARP_PERSPECTIVE
Object: Kernel, 208

VX_KERNEL_XOR
Object: Kernel, 207

VX_LUT_ATTRIBUTE_COUNT
Object: LUT, 163

VX_LUT_ATTRIBUTE_SIZE
Object: LUT, 163

VX_LUT_ATTRIBUTE_TYPE
Object: LUT, 163

VX_MATRIX_ATTRIBUTE_COLUMNS
Object: Matrix, 167

VX_MATRIX_ATTRIBUTE_ROWS
Object: Matrix, 167

VX_MATRIX_ATTRIBUTE_SIZE
Object: Matrix, 167

VX_MATRIX_ATTRIBUTE_TYPE
Object: Matrix, 167

VX_NODE_ATTRIBUTE_BORDER_MODE
Object: Node, 131

VX_NODE_ATTRIBUTE_LOCAL_DATA_PTR
Object: Node, 131

VX_NODE_ATTRIBUTE_LOCAL_DATA_SIZE
Object: Node, 131

VX_NODE_ATTRIBUTE_PERFORMANCE
Object: Node, 131

VX_NODE_ATTRIBUTE_STATUS
Object: Node, 131

VX_NORM_L1
Function: Canny Edge Detector, 46

VX_NORM_L2
Function: Canny Edge Detector, 46

VX_OUTPUT
Object: Parameter, 214

VX_PARAMETER_ATTRIBUTE_DIRECTION
Object: Parameter, 214

VX_PARAMETER_ATTRIBUTE_INDEX
Object: Parameter, 214

VX_PARAMETER_ATTRIBUTE_REF
Object: Parameter, 214

VX_PARAMETER_ATTRIBUTE_STATE
Object: Parameter, 214

VX_PARAMETER_ATTRIBUTE_TYPE
Object: Parameter, 214

VX_PARAMETER_STATE_OPTIONAL
Object: Parameter, 214

VX_PARAMETER_STATE_REQUIRED
Object: Parameter, 214

VX_PYRAMID_ATTRIBUTE_FORMAT
Object: Pyramid, 171

VX_PYRAMID_ATTRIBUTE_HEIGHT
Object: Pyramid, 171

VX_PYRAMID_ATTRIBUTE_LEVELS
Object: Pyramid, 171

VX_PYRAMID_ATTRIBUTE_SCALE
Object: Pyramid, 171

VX_PYRAMID_ATTRIBUTE_WIDTH
Object: Pyramid, 171

VX_READ_AND_WRITE
Object: Context, 120

VX_READ_ONLY
Object: Context, 120

VX_REF_ATTRIBUTE_COUNT
Object: Reference, 117

VX_REF_ATTRIBUTE_TYPE
Object: Reference, 117

VX_REMAP_ATTRIBUTE_DESTINATION_HEIGHT
Object: Remap, 174

VX_REMAP_ATTRIBUTE_DESTINATION_WIDTH
Object: Remap, 174

VX_REMAP_ATTRIBUTE_SOURCE_HEIGHT
Object: Remap, 174

VX_REMAP_ATTRIBUTE_SOURCE_WIDTH
Object: Remap, 174

VX_ROUND_POLICY_TO_NEAREST_EVEN
Object: Context, 122
| VX_ROUND_POLICY_TO_ZERO       | VX_TYPE_INT16                        |
|                                | Basic Features, 113                  |
| VX_SCALAR_ATTRIBUTE_TYPE       | VX_TYPE_INT32                        |
|                                | Basic Features, 113                  |
| VX_STATUS_MIN                  | VX_TYPE_INT64                        |
|                                | Basic Features, 113                  |
| VX_SUCCESS                     | VX_TYPE_INVALID                      |
|                                | Basic Features, 113                  |
| VX_TERM_CRITERIA_BOTH          | VX_TYPE_KERNEL                       |
|                                | Basic Features, 114                  |
| VX_TERM_CRITERIA_EPSILON       | VX_TYPE_KEYPOINT                     |
|                                | Basic Features, 114                  |
| VX_TERM_CRITERIA_ITERATIONS    | VX_TYPE_LUT                          |
|                                | Basic Features, 114                  |
| VX_THRESHOLD_ATTRIBUTE_LOWER   | VX_TYPE_OBJECT_MAX                   |
|                                | Basic Features, 114                  |
| VX_THRESHOLD_ATTRIBUTE_TYPE    | VX_TYPE_PARAMETER                    |
|                                | Basic Features, 114                  |
| VX_THRESHOLD_ATTRIBUTE_UPPER   | VX_TYPE_PYRAMID                      |
|                                | Basic Features, 114                  |
| VX_THRESHOLD_ATTRIBUTE_VALUE   | VX_TYPE_RECTANGLE                    |
|                                | Basic Features, 114                  |
| VX_THRESHOLD_TYPE_BINARY       | VX_TYPE_REFERENCE                    |
|                                | Basic Features, 114                  |
| VX_THRESHOLD_TYPE_RANGE        | VX_TYPE_REMAP                        |
|                                | Basic Features, 114                  |
| VX_TYPE_ARRAY                  | VX_TYPE_SCALAR                       |
|                                | Basic Features, 114                  |
| VX_TYPE_BOOL                   | VX_TYPE_SCALAR_MAX                   |
|                                | Basic Features, 114                  |
| VX_TYPE_CHAR                   | VX_TYPE_SIZE                         |
|                                | Basic Features, 114                  |
| VX_TYPE_CONTEXT                | VX_TYPE_THRESHOLD                    |
|                                | Basic Features, 114                  |
| VX_TYPE_CONVOLUTION            | VX_TYPE_UINT16                       |
|                                | Basic Features, 113                  |
| VX_TYPE_COORDINATES2D          | VX_TYPE_UINT32                       |
|                                | Basic Features, 113                  |
| VX_TYPE_COORDINATES3D          | VX_TYPE_UINT64                       |
|                                | Basic Features, 113                  |
| VX_TYPE_DELAY                  | VX_TYPE_UINT8                        |
|                                | Basic Features, 113                  |
| VX_TYPE_DISTRIBUTION           | VX_WRITE_ONLY                        |
|                                | Object: Context, 120                 |
| VX_TYPE_ENUM                   | VX_ENUM_BASE                         |
|                                | Basic Features, 108                  |
| VX_TYPE_ERROR                  | VX_FMT_REF                           |
|                                | Basic Features, 108                  |
Basic Features, 108

vx_fourcc
Basic Features, 108

vx_scaleUnity
Basic Features, 108

vx_typeMask
Basic Features, 108

vx_version
Basic Features, 108

vx_versionMajor
Basic Features, 109

vx_versionMinor
Basic Features, 109

Vision Functions, 24

vx_border_mode
Basic Features, 109

vx_coordinates2d
Basic Features, 109

vx_coordinates3d
Basic Features, 109

vx_false
Basic Features, 109

vx_imagepatch_addressing
Object: Imagepatch, 147

vx_kernelInfo
Object: Kernel, 201

vx_keypoint
Object: Keypoint, 107

vx_perf
Object: Performance, 189

vx_rectangle
Object: Rectangle, 108

vx_true
Basic Features, 109

vx_accessor
Object: Accessor, 120

vx_action
Framework: Node Callbacks, 187

vx_array
Object: Array, 135

vx_bool
Basic Features, 109

vx_border_mode
Node: Border Modes, 194

vx_channel
Basic Features, 109

vx_channel_range
Object: Channel Range, 149

vx_color_space
Object: Color Space, 149

vx_context
Object: Context, 120

vx_context_attribute
Object: Context Attribute, 120

vx_convertPolicy
Basic Features, 110

vx_convolution
Object: Convolution, 140

vx_delay
Object: Delay, 195

vx_delay_attribute
Object: Delay Attribute, 196

vx_direction
Object: Direction, 196

vx_directive
Framework: Directives, 221

vx_distribution_attribute
Object: Distribution Attribute, 143

vx_enum
Basic Features, 109

vx_enum_e
Basic Features, 110

vx_fourcc_e
Basic Features, 111

vx_graph
Object: Graph, 126

vx_graph_attribute
Object: Graph Attribute, 126

vx_hint
Framework: Hints, 219

vx_image
Object: Image, 149

vx_image_attribute
Object: Image Attribute, 150

vx_import_type
Object: Import Type, 121

vx_interpolation_type
Basic Features, 111

vx_kernel
Object: Kernel, 201

vx_kernel_attribute
Object: Kernel Attribute, 201

vx_kernel_deinitialize
Framework: Client Defined Functions, 225

vx_kernel_end
Framework: Client Defined Functions, 225

vx_kernel_f
Framework: Client Defined Functions, 225

vx_kernel_initialize
Framework: Client Defined Functions, 226

vx_kernel_input_validate
Framework: Client Defined Functions, 226

vx_kernel_output_validate
Framework: Client Defined Functions, 226

vx_lut_attribute
Object: LUT, 163

vx_matrix_attribute
Object: Matrix, 167

vx_node
Object: Node, 130

vx_node_attribute
Object: Node Attribute, 131

vx_nodecomplete
Framework: Node Callbacks, 187

vx_norm
Function: Canny Edge Detector, 46

vx_parameter
Object: Parameter, 214

vx_parameter_attribute
Object: Parameter Attribute, 214

vx_parameter_state
<table>
<thead>
<tr>
<th>Object: Graph, 126</th>
</tr>
</thead>
<tbody>
<tr>
<td>vxCreateImage</td>
</tr>
<tr>
<td>Object: Image, 154</td>
</tr>
<tr>
<td>vxCreateImageFromHandle</td>
</tr>
<tr>
<td>Object: Image, 154</td>
</tr>
<tr>
<td>vxCreateImageFromROI</td>
</tr>
<tr>
<td>Object: Image, 156</td>
</tr>
<tr>
<td>vxCreateLUT</td>
</tr>
<tr>
<td>Object: LUT, 164</td>
</tr>
<tr>
<td>vxCreateMatrix</td>
</tr>
<tr>
<td>Object: Matrix, 168</td>
</tr>
<tr>
<td>vxCreateNode</td>
</tr>
<tr>
<td>Object: Node (Advanced), 193</td>
</tr>
<tr>
<td>vxCreatePyramid</td>
</tr>
<tr>
<td>Object: Pyramid, 171</td>
</tr>
<tr>
<td>vxCreateRemap</td>
</tr>
<tr>
<td>Object: Remap, 175</td>
</tr>
<tr>
<td>vxCreateScalar</td>
</tr>
<tr>
<td>Object: Scalar, 179</td>
</tr>
<tr>
<td>vxCreateThreshold</td>
</tr>
<tr>
<td>Object: Threshold, 182</td>
</tr>
<tr>
<td>vxCreateUniformImage</td>
</tr>
<tr>
<td>Object: Image, 156</td>
</tr>
<tr>
<td>vxCreateVirtualArray</td>
</tr>
<tr>
<td>Object: Array, 137</td>
</tr>
<tr>
<td>vxCreateVirtualImage</td>
</tr>
<tr>
<td>Object: Image, 157</td>
</tr>
<tr>
<td>vxCreateVirtualPyramid</td>
</tr>
<tr>
<td>Object: Pyramid, 171</td>
</tr>
<tr>
<td>vxDilate3x3Node</td>
</tr>
<tr>
<td>Function: Dilate Image, 60</td>
</tr>
<tr>
<td>vxDirective</td>
</tr>
<tr>
<td>Framework: Directives, 221</td>
</tr>
<tr>
<td>vxDissociateDelayFromNode</td>
</tr>
<tr>
<td>Object: Delay, 197</td>
</tr>
<tr>
<td>vxEqualizeHistNode</td>
</tr>
<tr>
<td>Function: Equalize Histogram, 61</td>
</tr>
<tr>
<td>vxErode3x3Node</td>
</tr>
<tr>
<td>Function: Erode Image, 62</td>
</tr>
<tr>
<td>vxFastCornersNode</td>
</tr>
<tr>
<td>Function: Fast Corners, 64</td>
</tr>
<tr>
<td>vxFinalizeKernel</td>
</tr>
<tr>
<td>Framework: Client Defined Functions, 228</td>
</tr>
<tr>
<td>vxFormatArrayPointer</td>
</tr>
<tr>
<td>Object: Array, 135</td>
</tr>
<tr>
<td>vxFormatImagePatchAddress1d</td>
</tr>
<tr>
<td>Object: Image, 157</td>
</tr>
<tr>
<td>vxFormatImagePatchAddress2d</td>
</tr>
<tr>
<td>Object: Image, 159</td>
</tr>
<tr>
<td>vxGaussian3x3Node</td>
</tr>
<tr>
<td>Function: Gaussian Filter, 66</td>
</tr>
<tr>
<td>vxGaussianPyramidNode</td>
</tr>
<tr>
<td>Function: Gaussian Image Pyramid, 71</td>
</tr>
<tr>
<td>vxGetContext</td>
</tr>
<tr>
<td>Object: Context, 122</td>
</tr>
<tr>
<td>vxGetGraphParameterByIndex</td>
</tr>
<tr>
<td>Framework: Graph Parameters, 232</td>
</tr>
<tr>
<td>vxGetKernelByEnum</td>
</tr>
<tr>
<td>Object: Kernel, 210</td>
</tr>
<tr>
<td>vxGetKernelByName</td>
</tr>
<tr>
<td>Object: Kernel, 211</td>
</tr>
<tr>
<td>vxgetParameterByIndex</td>
</tr>
<tr>
<td>Object: Parameter, 214</td>
</tr>
<tr>
<td>vxGetPyramidLevel</td>
</tr>
<tr>
<td>Object: Pyramid, 172</td>
</tr>
<tr>
<td>vxGetReferenceFromDelay</td>
</tr>
<tr>
<td>Object: Delay, 198</td>
</tr>
<tr>
<td>vxGetRemapPoint</td>
</tr>
<tr>
<td>Object: Remap, 176</td>
</tr>
<tr>
<td>vxGetStatus</td>
</tr>
<tr>
<td>Basic Features, 115</td>
</tr>
<tr>
<td>vxGetValidRegionImage</td>
</tr>
<tr>
<td>Object: Image, 160</td>
</tr>
<tr>
<td>vxHarrisCornersNode</td>
</tr>
<tr>
<td>Function: Harris Corners, 68</td>
</tr>
<tr>
<td>vxHint</td>
</tr>
<tr>
<td>Framework: Hints, 219</td>
</tr>
<tr>
<td>vxHistogramNode</td>
</tr>
<tr>
<td>Function: Histogram, 70</td>
</tr>
<tr>
<td>vxIntegralImageNode</td>
</tr>
<tr>
<td>Function: Integral Image, 73</td>
</tr>
<tr>
<td>vxIsGraphVerified</td>
</tr>
<tr>
<td>Object: Graph, 126</td>
</tr>
<tr>
<td>vxLoadKernels</td>
</tr>
<tr>
<td>Framework: Client Defined Functions, 228</td>
</tr>
<tr>
<td>vxMagnitudeNode</td>
</tr>
<tr>
<td>Function: Magnitude, 74</td>
</tr>
<tr>
<td>vxMeanStdDevNode</td>
</tr>
<tr>
<td>Function: Mean and Standard Deviation, 76</td>
</tr>
<tr>
<td>vxMedian3x3Node</td>
</tr>
<tr>
<td>Function: Median Filter, 78</td>
</tr>
<tr>
<td>vxMinMaxLocNode</td>
</tr>
<tr>
<td>Function: Min, Max Location, 79</td>
</tr>
<tr>
<td>vxMultiplyNode</td>
</tr>
<tr>
<td>Function: Pixel-wise Multiplication, 87</td>
</tr>
<tr>
<td>vxNotNode</td>
</tr>
<tr>
<td>Function: Bitwise Not, 43</td>
</tr>
<tr>
<td>vxOpticalFlowPyrLKNode</td>
</tr>
<tr>
<td>Function: Optical Flow Pyramid (LK), 82</td>
</tr>
<tr>
<td>vxOrNode</td>
</tr>
<tr>
<td>Function: Bitwise Inclusive Or, 41</td>
</tr>
<tr>
<td>vxPhaseNode</td>
</tr>
<tr>
<td>Function: Phase, 85</td>
</tr>
<tr>
<td>vxProcessGraph</td>
</tr>
<tr>
<td>Object: Graph, 127</td>
</tr>
<tr>
<td>vxQueryArray</td>
</tr>
<tr>
<td>Object: Array, 138</td>
</tr>
<tr>
<td>vxQueryContext</td>
</tr>
<tr>
<td>Object: Context, 123</td>
</tr>
<tr>
<td>vxQueryConvolution</td>
</tr>
<tr>
<td>Object: Convolution, 142</td>
</tr>
<tr>
<td>vxQueryDelay</td>
</tr>
<tr>
<td>Object: Delay, 198</td>
</tr>
<tr>
<td>vxQueryDistribution</td>
</tr>
<tr>
<td>Object: Distribution, 145</td>
</tr>
<tr>
<td>vxQueryGraph</td>
</tr>
<tr>
<td>Function/Property</td>
</tr>
<tr>
<td>-----------------------------------</td>
</tr>
<tr>
<td>vxQueryImage</td>
</tr>
<tr>
<td>vxQueryKernel</td>
</tr>
<tr>
<td>vxQueryLUT</td>
</tr>
<tr>
<td>vxQueryMatrix</td>
</tr>
<tr>
<td>vxQueryNode</td>
</tr>
<tr>
<td>vxQueryParameter</td>
</tr>
<tr>
<td>vxQueryPyramid</td>
</tr>
<tr>
<td>vxQueryReference</td>
</tr>
<tr>
<td>vxQueryRemap</td>
</tr>
<tr>
<td>vxQueryScalar</td>
</tr>
<tr>
<td>vxQueryThreshold</td>
</tr>
<tr>
<td>vxRegisterLogCallback</td>
</tr>
<tr>
<td>vxRegisterUserStruct</td>
</tr>
<tr>
<td>vxReleaseArray</td>
</tr>
<tr>
<td>vxReleaseContext</td>
</tr>
<tr>
<td>vxReleaseConvolution</td>
</tr>
<tr>
<td>vxReleaseDelay</td>
</tr>
<tr>
<td>vxReleaseDistribution</td>
</tr>
<tr>
<td>vxReleaseGraph</td>
</tr>
<tr>
<td>vxReleaseImage</td>
</tr>
<tr>
<td>vxReleaseKernel</td>
</tr>
<tr>
<td>vxReleaseLUT</td>
</tr>
<tr>
<td>vxReleaseMatrix</td>
</tr>
<tr>
<td>vxReleaseNode</td>
</tr>
<tr>
<td>vxReleaseParameter</td>
</tr>
<tr>
<td>vxReleasePyramid</td>
</tr>
<tr>
<td>vxReleaseRemap</td>
</tr>
<tr>
<td>vxReleaseScalar</td>
</tr>
<tr>
<td>vxReleaseThreshold</td>
</tr>
<tr>
<td>vxRemapNode</td>
</tr>
<tr>
<td>vxRemoveKernel</td>
</tr>
<tr>
<td>vxRemoveNode</td>
</tr>
<tr>
<td>vxRetrieveNodeCallback</td>
</tr>
<tr>
<td>vxScaleImageNode</td>
</tr>
<tr>
<td>vxScheduleGraph</td>
</tr>
<tr>
<td>vxSetContextAttribute</td>
</tr>
<tr>
<td>vxSetConvolutionAttribute</td>
</tr>
<tr>
<td>vxSetGraphAttribute</td>
</tr>
<tr>
<td>vxSetGraphParameterByIndex</td>
</tr>
<tr>
<td>vxSetImageAttribute</td>
</tr>
<tr>
<td>vxSetKernelAttribute</td>
</tr>
<tr>
<td>vxSetNodeAttribute</td>
</tr>
<tr>
<td>vxSetParameterByIndex</td>
</tr>
<tr>
<td>vxSetParameterByReference</td>
</tr>
<tr>
<td>vxSetRemapPoint</td>
</tr>
<tr>
<td>vxSetThresholdAttribute</td>
</tr>
<tr>
<td>vxSobel3x3Node</td>
</tr>
<tr>
<td>vxSubtractNode</td>
</tr>
<tr>
<td>vxTableLookupNode</td>
</tr>
<tr>
<td>vxThresholdNode</td>
</tr>
<tr>
<td>vxTruncateArray</td>
</tr>
<tr>
<td>vxVerifyGraph</td>
</tr>
<tr>
<td>vxWaitGraph</td>
</tr>
<tr>
<td>vxWarpAffineNode</td>
</tr>
<tr>
<td>vxWarpPerspectiveNode</td>
</tr>
<tr>
<td>vxXorNode</td>
</tr>
<tr>
<td>vxuAbsDiff</td>
</tr>
<tr>
<td>vxuAccumulateImage</td>
</tr>
</tbody>
</table>
Function: Accumulate, 28
vxuAccumulateSquareImage
  Function: Accumulate Squared, 29
vxuAccumulateWeightedImage
  Function: Accumulate Weighted, 31
vxuAdd
  Function: Arithmetic Addition, 33
vxuAnd
  Function: Bitwise And, 37
vxuBox3x3
  Function: Box Filter, 44
vxuCannyEdgeDetector
  Function: Canny Edge Detector, 46
vxuChannelCombine
  Function: Channel Combine, 48
vxuChannelExtract
  Function: Channel Extract, 50
vxuColorConvert
  Function: Color Convert, 54
vxuConvertDepth
  Function: Convert Bit depth, 57
vxuConvolve
  Function: Custom Convolution, 59
vxuDilate3x3
  Function: Dilate Image, 60
vxuEqualizeHist
  Function: Equalize Histogram, 61
vxuErode3x3
  Function: Erode Image, 62
vxuFastCorners
  Function: Fast Corners, 64
vxuGaussian3x3
  Function: Gaussian Filter, 66
vxuGaussianPyramid
  Function: Gaussian Image Pyramid, 71
vxuHalfScaleGaussian3x3
  Function: Scale Image, 91
vxuHarrisCorners
  Function: Harris Corners, 68
vxuHistogram
  Function: Histogram, 70
vxuIntegralImage
  Function: Integral Image, 73
vxuMagnitude
  Function: Magnitude, 74
vxuMeanStdDev
  Function: Mean and Standard Deviation., 76
vxuMedian3x3
  Function: Median Filter, 78
vxuMinMaxLoc
  Function: Min, Max Location, 79
vxuMultiply
  Function: Pixel-wise Multiplication, 88
vxuNot
  Function: Bitwise Not, 43
vxuOpticalFlowPyrLK
  Function: Optical Flow Pyramid (LK), 83
vxuOr
  Function: Bitwise Inclusive Or, 41
vxuPhase
  Function: Phase, 85
vxuRemap
  Function: Remap, 89
vxuScaleImage
  Function: Scale Image, 92
vxuSobel3x3
  Function: Sobel 3x3, 93
vxuSubtract
  Function: Arithmetic Subtraction, 35
vxuTableLookup
  Function: TableLookup, 95
vxuThreshold
  Function: Thresholding, 96
vxuWarpAffine
  Function: Warp Affine, 98
vxuWarpPerspective
  Function: Warp Perspective, 100
vxuXor
  Function: Bitwise Exclusive Or, 39