The **OpenVX™** User Node Tiling Extension

Version 1.0

Document Revision: 1

Khronos Vision Working Group

Editor: Shorin Kyo
Editor: Thierry Lepley
Editor: Erik Rainey
Editor: Frank Brill

Copyright ©2013 The Khronos Group Inc.

Copyright ©2013 The Khronos Group Inc. All Rights Reserved.

This specification is protected by copyright laws and contains material proprietary to the Khronos Group, Inc. It or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast, or otherwise exploited in any manner without the express prior written permission of the Khronos Group. You may use this specification for implementing the functionality therein, without altering or removing any trademark, copyright or other notice from the specification, but the receipt or possession of this specification does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part. Khronos Group grants express permission to any current Promoter, Contributor or Adopter member of Khronos to copy and redistribute UNMODIFIED versions of this specification in any fashion, provided that NO CHARGE is made for the specification and the latest available update of the specification for any version of the API is used whenever possible. Such distributed specification may be reformatted AS LONG AS the contents of the specification are not changed in any way. The specification may be incorporated into a product that is sold as long as such product includes significant independent work developed by the seller. A link to the current version of this specification on the Khronos Group website should be included whenever possible with specification distributions.

Khronos Group makes no, and expressly disclaims any, representations or warranties, express or implied, regarding this specification, including, without limitation, any implied warranties of merchantability or fitness for a particular purpose or non-infringement of any intellectual property. Khronos Group makes no, and expressly disclaims any, warranties, express or implied, regarding the correctness, accuracy, completeness, timeliness, and reliability of the specification. Under no circumstances will the Khronos Group, or any of its Promoters, Contributors or Members or their respective partners, officers, directors, employees, agents or representatives be liable for any damages, whether direct, indirect, special or consequential damages for lost revenues, lost profits, or otherwise, arising from or in connection with these materials. SAMPLE CODE and EXAMPLES, as identified herein, are expressly depicted herein with a ”grey” watermark and are included for illustrative purposes only and are expressly outside of the Scope as defined in Attachment A - Khronos Group Intellectual Property (IP) Rights Policy of the Khronos Group Membership Agreement. A Member or Promoter Member shall have no obligation to grant any licenses under any Necessary Patent Claims covering SAMPLE CODE and EXAMPLES.

Contents

Chapter 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 target 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.

1.1 Purpose

The purpose of having a User Node Tiling Function is to:

- Provide a mechanism for user-node programmers to communicate enough information about their computation to enable OpenVX implementations to perform some optimizations that take advantage of locality in the user-node computation, and enable parallelism of execution across local data sets.
- Define a portable method to enable tiling on compatible platforms.
- Define a simple way to enable users to write image processing functions.

1.2 Basic Design

The user node tiling facility is largely composed of two components:

- User Node informs Graph Manager - The author of the tilable user kernel informs the graph manager about the data requirements of the algorithm implemented by the kernel. This is primarily done by setting attributes on the kernel (for all nodes of this type), or on the node (for that particular node). This is usually done at kernel definition time (for kernel attributes) or at node creation time (for node attributes).
- Graph Manager informs User Node - The graph manager informs the user kernel code about the sub-image or tile that the user kernel code will work on. This is primarily done via macros that the user kernel code can call to get this information at run time for a particular invocation of the user code.

1.3 Glossary

Terms and Definitions of the Extension:

- Input tileBlock - the minimum algorithmic size of the rectangle of source pixels needed to compute the function. In OpenVX 1.0, this value is unconditionally regarded to be same as the output tileBlock size.
- Output tileBlock - the minimum algorithmic size of the rectangle of destination pixels to be computed by the function.

- Input neighborhood - the additional pixels surrounding the tile in the input images needed to compute the output tile.
- Output neighborhood - the additional (previously computed) pixels surrounding the tile in the output image needed to compute the output tile. In OpenVX 1.0, this value is unconditionally regarded as zero in all four directions (top, bottom, left, right).
- Tile - the rectangle of destination pixels to be computed by the function, whose size is greater than or equal to the output tileBlock and less than or equal to the rectangle of the entire image. This value is determined by the OpenVX runtime.
- Border mode - defines both the referred pixel values at the image border (i.e. outside the image boundary), and the behavior of the OpenVX runtime against the tile size determination when the image size happens to be not a multiple of the output tileBlock size.
- Tile memory size - a facility for per-tile memory allocation. The amount allocated will be the amount requested times the number of tileBlocks in the given tile. Lifetime of the given pointer is per invocation of the user function that processes a tile.

1.4 Adding a Tiling Kernel Function

```
vx_status status = VX_SUCCESS;
vx_uint32 k = 0;
for (k = 0; k < dimof(tiling_kernels); k++)
{
    vx_kernel kernel = vxAddTilingKernel(context,
            tiling_kernels[k].name,
            tiling_kernels[k].enumeration,
            tiling_kernels[k].function,
            tiling_kernels[k].num_params,
            tiling_kernels[k].input_validator,
            tiling_kernels[k].output_validator);
    if (kernel)
    {
        vx_uint32 p = 0;
        for (p = 0; p < tiling_kernels[k].num_params; p++)
        {
            status |= vxAddParameterToKernel(kernel, p,
                        tiling_kernels[k].parameters[p].direction,
                        tiling_kernels[k].parameters[p].type,
                        tiling_kernels[k].parameters[p].state);
        }
        status |= vxSetKernelAttribute(kernel,
  VX_KERNEL_ATTRIBUTE_INPUT_NEIGHBORHOOD, &tiling_kernels[k].nbhd,
  sizeof(vx_neighborhood_size_t));
        status |= vxSetKernelAttribute(kernel,
  VX_KERNEL_ATTRIBUTE_OUTPUT_TILE_BLOCK_SIZE, &tiling_kernels[k].
  block, sizeof(vx_tile_block_size_t));
        status |= vxSetKernelAttribute(kernel, VX_KERNEL_ATTRIBUTE_BORDER, &
  tiling_kernels[k].border, sizeof(vx_border_mode_t));
        if (status != VX_SUCCESS)
        {
            vxRemoveKernel(kernel);
        }
        else
        {
            status = vxFinalizeKernel(kernel);
        }
        if (status != VX_SUCCESS)
        {
            printf("Failed to publish kernel %s\n", tiling_kernels[k].name);
            break;
        }
    }
}
```

1.5 Example Kernels

Users can add simple filter-style kernels similar to this example of a Gaussian Blur.

```
void gaussian_image_tiling(void * VX_RESTRICT parameters[VX_RESTRICT],
                           void * VX_RESTRICT tile_memory,
                           vx_size tile_memory_size)
{
    vx_uint32 x, y;
    vx_tile_t *in = (vx_tile_t *)parameters[0];
    vx_tile_t *out = (vx_tile_t *)parameters[1];

    for (y = 0; y < vxTileHeight(out, 0); y += vxTileBlockHeight(out))
    {
        for (x = 0; x < vxTileWidth(out, 0); x += vxTileBlockWidth(out))
        {
            vx_uint32 sum = 0;
            /* this math covers a {-1,1},{-1,1} neighborhood and a block of 1x1.
             * an internal set of for loop could have been placed here instead.
             */
            sum += vxImagePixel(vx_uint8, in, 0, x, y, -1, -1);
            sum += vxImagePixel(vx_uint8, in, 0, x, y,  0, -1) << 1;
            sum += vxImagePixel(vx_uint8, in, 0, x, y, +1, -1);
            sum += vxImagePixel(vx_uint8, in, 0, x, y, -1,  0) << 1;
            sum += vxImagePixel(vx_uint8, in, 0, x, y,  0,  0) << 2;
            sum += vxImagePixel(vx_uint8, in, 0, x, y, +1,  0) << 1;
            sum += vxImagePixel(vx_uint8, in, 0, x, y, -1, +1);
            sum += vxImagePixel(vx_uint8, in, 0, x, y,  0, +1) << 1;
            sum += vxImagePixel(vx_uint8, in, 0, x, y, +1, +1);
            sum >>= 4;
            if (sum > 255)
                sum = 255;
            vxImagePixel(vx_uint8, out, 0, x, y, 0, 0) = (vx_uint8)sum;
        }
    }
}
```

Users can also add more complex addressing functionality in order to optimize their code in this example which uses two inputs.

```
void add_image_tiling(void * VX_RESTRICT parameters[VX_RESTRICT],
                      void * VX_RESTRICT tile_memory,
                      vx_size tile_memory_size)
{
    vx_uint32 i,j;
    vx_tile_t *in0 = (vx_tile_t *)parameters[0];
    vx_tile_t *in1 = (vx_tile_t *)parameters[1];
    vx_tile_t *out = (vx_tile_t *)parameters[2];

    for (j = 0u; j < vxTileHeight(out,0); j+=vxTileBlockHeight(out))
    {
        for (i = 0u; i < vxTileWidth(out,0); i+=vxTileBlockWidth(out))
        {
            /* this math happens to cover a 1x1 block and has no neighborhood */
            vx_uint16 pixel = vxImagePixel(vx_uint8, in0, 0, i, j, 0, 0) +
                              vxImagePixel(vx_uint8, in1, 0, i, j, 0, 0);
            if (pixel > INT16_MAX)
                pixel = INT16_MAX;
            vxImagePixel(vx_int16, out, 0, i, j, 0, 0) = (vx_int16)pixel;
        }
    }
}
```

1.6 Informative Coding Guidelines

User node tiles should be coded in a very specific manner to maintain the requirement of *data independence* in that all blocks should be computed in a *functionally independent* manner. This allows the implement greater flexibility in optimization choices. These informal requirements include:

- Avoid using global shared memory.
- Avoid using blocking access protected mechanisms (semaphores, etc).
- Avoid using dynamic memory allocation functions at runtime.
- Avoid calling other functions internally in the tile function.
- Do try to write the your loops in an efficient manner for your compiler environment.

Chapter 2

Module Documentation

2.1 Extension: User Tiling API

The Khronos Extension for User Defined Tiling Functions.

Data Structures

- struct vx_image_description_t

  A structure which describes the parent image. More...
- struct vx_neighborhood_size_t

  The User Tiling Function Neighborhood declaration. More...
- struct vx_tile_block_size_t

  The User Tiling Function tile block size declaration. More...
- struct vx_tile_t

  The tile structure declaration. More...

Macros

- #define VX_MAX_TILING_PLANES (4)

  The maximum number of planes in a tiled image.
- #define VX_RESTRICT restrict

  A platform wrapper for the restrict keyword.
- #define vxImageHeight(ptile) ((ptile))->image.height)

  The full height of the tile's source image in pixels.
- #define vxImageOffset(ptile, i, x, y, ox, oy)

  Computes the offset within an image.
- #define vxImagePixel(type, ptile, i, x, y, ox, oy) ∗((type ∗)(&((**vx_uint8** ∗)(ptile)->base[i])[vxImageOffset(ptile, i, x, y, ox, oy)]))

  Accesses an image pixel as a type-cast indexed pointer dereference.
- #define vxImageWidth(ptile) ((ptile))->image.width)

  The full width of the tile's source image in pixels.
- #define vxNeighborhoodBottom(ptile) ((ptile)->neighborhood.bottom)

  The simple wrapper to access each images neighborhood +Y value.
- #define vxNeighborhoodLeft(ptile) ((ptile)->neighborhood.left)

  The simple wrapper to access each images neighborhood -X value.
- #define vxNeighborhoodRight(ptile) ((ptile)->neighborhood.right)

  The simple wrapper to access each images neighborhood +X value.
- #define vxNeighborhoodTop(ptile) ((ptile)->neighborhood.top)

  The simple wrapper to access each images neighborhood -Y value.
- #define vxTileBlockHeight(ptile) ((ptile)->tile_block.height)

The algorithmic minimum tile block height.

- #define vxTileBlockWidth(ptile) ((ptile)->tile_block.width)

  The algorithmic minimum tile block width.
- #define vxTileHeight(ptile, index) ((ptile)->addr[index].dim_y)

  The height of the tile in pixels.
- #define vxTileWidth(ptile, index) ((ptile)->addr[index].dim_x)

  The width of the tile in pixels.
- #define vxTileX(ptile) ((ptile)->tile_x)

  The tile index in the x direction.
- #define vxTileY(ptile) ((ptile)->tile_y)

  The tile index in the Y direction.

Typedefs

- typedef void(∗ vx_tiling_kernel_f)(void ∗restrict parameters[restrict], void ∗restrict tile_memory, **vx_size** tile_-memory_size)

  Tiling Kernel function typedef for User Defined Tiling Kernels.

Enumerations

- enum vx_border_mode_tiling_e { VX_BORDER_MODE_SELF = (((VX_ID_KHRONOS) << 20) | (VX_ENUM-_BORDER_MODE << 12)) + 0x3 }

  The tiling border mode extensions.
- enum vx_kernel_attribute_tiling_e { VX_KERNEL_ATTRIBUTE_INPUT_NEIGHBORHOOD = (((VX_ID_KHRO-NOS) << 20) | (VX_TYPE_KERNEL << 8)) + 0x7, VX_KERNEL_ATTRIBUTE_OUTPUT_TILE_BLOCK_SIZE = (((VX_ID_KHRONOS) << 20) | (VX_TYPE_KERNEL << 8)) + 0x8, VX_KERNEL_ATTRIBUTE_BORDER = (((VX_ID_KHRONOS) << 20) | (VX_TYPE_KERNEL << 8)) + 0x9, VX_KERNL_ATTRIBUTE_TILE_MEM-ORY_SIZE = (((VX_ID_KHRONOS) << 20) | (VX_TYPE_KERNEL << 8)) + 0xA }

  The User Kernel Tiling Attributes.
- enum vx_node_attribute_tiling_e { VX_NODE_ATTRIBUTE_INPUT_NEIGHBORHOOD = (((VX_ID_KHRONOS) << 20) | (VX_TYPE_NODE << 8)) + 0x7, VX_NODE_ATTRIBUTE_OUTPUT_TILE_BLOCK_SIZE = (((VX-_ID_KHRONOS) << 20) | (VX_TYPE_NODE << 8)) + 0x8, VX_NODE_ATTRIBUTE_TILE_MEMORY_SIZE = (((VX_ID_KHRONOS) << 20) | (VX_TYPE_NODE << 8)) + 0xA }

  The User Node Tiling Attributes.

Functions

- **vx_kernel** vxAddTilingKernel (**vx_context** context, **vx_char** name[(256)], **vx_enum** enumeration, vx_tiling_-kernel_f func_ptr, **vx_uint32** num_params, vx_kernel_input_validate_f input, vx_kernel_output_validate_f output)

  Allows a user to add a tile-able kernel to the OpenVX system.

2.1.1 Detailed Description

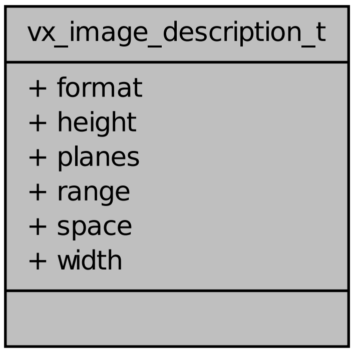
The Khronos Extension for User Defined Tiling Functions.

2.1.2 Data Structure Documentation

struct vx_image_description_t

A structure which describes the parent image.
Definition at line 81 of file vx_khr_tiling.h.

Collaboration diagram for vx_image_description_t:



Data Fields

vx_fourcc	format	The **vx_fourcc_e** of the image.
vx_uint32	height	Height of the image.
vx_uint32	planes	The number of planes in the image.
vx_enum	range	The `vx_channel_range_e` enumeration.
vx_enum	space	The `vx_color_space_e` enumeration.
vx_uint32	width	Width of the image.

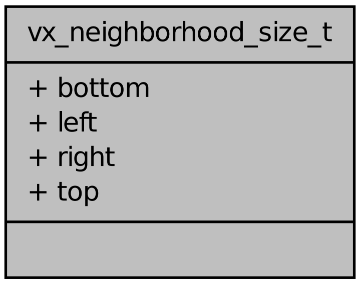
struct vx_neighborhood_size_t

The User Tiling Function Neighborhood declaration.

The author of a User Tiling Kernel will use this structure to define the neighborhood surrounding the tile block.

Definition at line 71 of file vx_khr_tiling.h.

Collaboration diagram for vx_neighborhood_size_t:



Data Fields

vx_int32	bottom	Bottom of the tile block.
vx_int32	left	Left of the tile block.
vx_int32	right	Right of the tile block.
vx_int32	top	Top of the tile block.

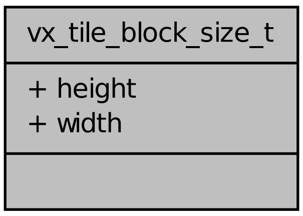
struct vx_tile_block_size_t

The User Tiling Function tile block size declaration.

The author of a User Tiling Kernel will use this structure to define the dimensionality of the algorithm tile block.

Definition at line 61 of file vx_khr_tiling.h.

Collaboration diagram for vx_tile_block_size_t:



Data Fields

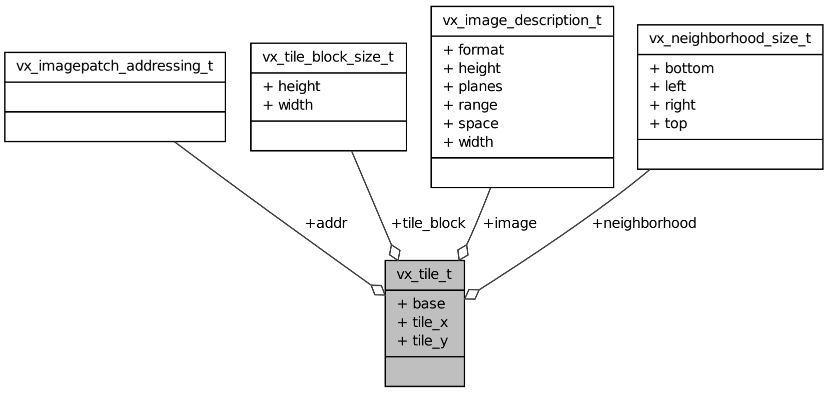
vx_int32	height	tile block height in pixels.
vx_int32	width	tile block width in pixels

struct vx_tile_t

The tile structure declaration.

Definition at line 98 of file vx_khr_tiling.h.

Collaboration diagram for vx_tile_t:



Data Fields

vx_imagepatch-_addressing_t	addr[(4)]	The array of addressing structure to describe each plane.
vx_uint8 ∗restrict	base[(4)]	The array of pointers to the tile's image plane.
vx_image-description_t	image	The description and attributes of the image.
vx-neighborhood-size_t	neighborhood	The neighborhood definition.
vx_tile_block-size_t	tile_block	The output block size structure.
vx_uint32	tile_x	The top left X pixel index within the width dimension of the image.
vx_uint32	tile_y	The top left Y pixel index within the height dimension of the image.

2.1.3 Macro Definition Documentation

#define vxImageHeight(***ptile*** **) ((ptile))->image.height)**

The full height of the tile's source image in pixels.

Parameters

in	*ptile*	The pointer to the vx_tile_t structure.

Definition at line 122 of file vx_khr_tiling.h.

#define vxImageOffset(***ptile, i, x, y, ox, oy*** **)**

Value:

```
((ptile)->addr[i].stride_y * (vx_int32)(((vx_int32)((oy)+(y)) * (vx_int32)(ptile)->addr[i].scale_y)/(
     vx_int32)VX_SCALE_UNITY)) + \
       ((ptile)->addr[i].stride_x * (vx_int32)(((vx_int32)((ox)+(x)) * (
     vx_int32)(ptile)->addr[i].scale_x)/(vx_int32)VX_SCALE_UNITY))
```

Computes the offset within an image.

Parameters

in	*ptile*	The pointer to the vx_tile_t structure.
in	*i*	The plance index.
in	*x*	The Width Coordinates.
in	*y*	The Height Coordinates.
in	*ox*	The X offset.
in	*oy*	The Y offset.

Definition at line 311 of file vx_khr_tiling.h.

#define vxImagePixel(***type, ptile, i, x, y, ox, oy*** **) ∗((type ∗)(&((vx_uint8 ∗)(ptile)->base[i])[vxImageOffset(ptile, i, x, y, ox, oy)]))**

Accesses an image pixel as a type-cast indexed pointer dereference.

Parameters

in	*type*	The type of the image pixel. Example values are **vx_uint8**, **vx_uint16**, **vx_uint32**, etc.
in	*ptile*	The pointer to the vx_tile_t structure.
in	*i*	The plance index.

`in`	*x*	The Center Pixel in Width Coordinates.
`in`	*y*	The Center Pixel in Height Coordinates.
`in`	*ox*	The X offset.
`in`	*oy*	The Y offset.

Definition at line 327 of file vx_khr_tiling.h.

#define vxImageWidth(*ptile*) ((ptile))->image.width)

The full width of the tile's source image in pixels.

Parameters

`in`	*ptile*	The pointer to the vx_tile_t structure.

Definition at line 129 of file vx_khr_tiling.h.

#define vxNeighborhoodBottom(*ptile*) ((ptile)->neighborhood.bottom)

The simple wrapper to access each images neighborhood +Y value.

Parameters

`in`	*ptile*	The pointer to the vx_tile_t structure.

Definition at line 202 of file vx_khr_tiling.h.

#define vxNeighborhoodLeft(*ptile*) ((ptile)->neighborhood.left)

The simple wrapper to access each images neighborhood -X value.

Parameters

`in`	*ptile*	The pointer to the vx_tile_t structure.

Definition at line 181 of file vx_khr_tiling.h.

#define vxNeighborhoodRight(*ptile*) ((ptile)->neighborhood.right)

The simple wrapper to access each images neighborhood +X value.

Parameters

`in`	*ptile*	The pointer to the vx_tile_t structure.

Definition at line 188 of file vx_khr_tiling.h.

#define vxNeighborhoodTop(*ptile*) ((ptile)->neighborhood.top)

The simple wrapper to access each images neighborhood -Y value.

Parameters

`in`	*ptile*	The pointer to the vx_tile_t structure.

Definition at line 195 of file vx_khr_tiling.h.

#define vxTileBlockHeight(*ptile*) ((ptile)->tile_block.height)

The algorithmic minimum tile block height.

Parameters

`in`	*ptile*	The pointer to the vx_tile_t structure.

Definition at line 167 of file vx_khr_tiling.h.

#define vxTileBlockWidth(*ptile*) ((ptile)->tile_block.width)

The algorithmic minimum tile block width.

Parameters

`in`	*ptile*	The pointer to the vx_tile_t structure.

Definition at line 174 of file vx_khr_tiling.h.

#define vxTileHeight(*ptile,* *index*) ((ptile)->addr[index].dim_y)

The height of the tile in pixels.

Parameters

`in`	*ptile*	The pointer to the vx_tile_t structure.
`in`	*index*	The plane index.

Definition at line 160 of file vx_khr_tiling.h.

#define vxTileWidth(*ptile,* *index*) ((ptile)->addr[index].dim_x)

The width of the tile in pixels.

Parameters

`in`	*ptile*	The pointer to the vx_tile_t structure.
`in`	*index*	The plane index.

Definition at line 152 of file vx_khr_tiling.h.

#define vxTileX(*ptile*) ((ptile)->tile_x)

The tile index in the x direction.

Parameters

`in`	*ptile*	The pointer to the vx_tile_t structure.

Definition at line 137 of file vx_khr_tiling.h.

#define vxTileY(*ptile*) ((ptile)->tile_y)

The tile index in the Y direction.

Parameters

`in`	*ptile*	The pointer to the vx_tile_t structure.

Definition at line 144 of file vx_khr_tiling.h.

2.1.4 Typedef Documentation

typedef void(∗ vx_tiling_kernel_f)(void ∗restrict parameters[restrict], void ∗restrict tile_memory, vx_size tile_memory_size)

Tiling Kernel function typedef for User Defined Tiling Kernels.

Note

Tiles may come in any dimension and are not guaranteed to be delivered in any particular order.

Parameters

`in`	*parameters*	The array abstract pointers to parameters.
`in`	*tile_memory*	The local tile memory pointer if requested, otherwise NULL.
`in`	*tile_memory_size*	The size of the local tile memory, if not requested, 0.

Definition at line 295 of file vx_khr_tiling.h.

2.1.5 Enumeration Type Documentation

enum vx_border_mode_tiling_e

The tiling border mode extensions.

Enumerator

VX_BORDER_MODE_SELF This value indicates that the author of the tiling kernel wrote code to handle border conditions into the kernel itself. If this mode is set, it can not be overriden by a call to the **vxSetNodeAttribute** with **VX_NODE_ATTRIBUTE_BORDER_MODE**.

Definition at line 278 of file vx_khr_tiling.h.

enum vx_kernel_attribute_tiling_e

The User Kernel Tiling Attributes.

Enumerator

VX_KERNEL_ATTRIBUTE_INPUT_NEIGHBORHOOD This allows a tiling mode kernel to set its input neighborhood.

VX_KERNEL_ATTRIBUTE_OUTPUT_TILE_BLOCK_SIZE This allows a tiling mode kernel to set its output tile block size.

VX_KERNEL_ATTRIBUTE_BORDER This allows the author to set the border mode on the tiling kernel.

VX_KERNL_ATTRIBUTE_TILE_MEMORY_SIZE This determines the per tile memory allocation.

Definition at line 239 of file vx_khr_tiling.h.

enum vx_node_attribute_tiling_e

The User Node Tiling Attributes.

Note

These are largely unusable by the tiling function, as it doesn't give you the node reference!

Enumerator

VX_NODE_ATTRIBUTE_INPUT_NEIGHBORHOOD This allows a tiling mode node to get its input neighborhood.

VX_NODE_ATTRIBUTE_OUTPUT_TILE_BLOCK_SIZE This allows a tiling mode node to get its output tile block size.

VX_NODE_ATTRIBUTE_TILE_MEMORY_SIZE This is the size of the tile local memory area.

Definition at line 260 of file vx_khr_tiling.h.

2.1.6 Function Documentation

vx_kernel vxAddTilingKernel (vx_context *context,* vx_char *name[(256)],* vx_enum *enumeration,* vx_tiling_kernel_f *func_ptr,* vx_uint32 *num_params,* vx_kernel_input_validate_f *input,* vx_kernel_output_validate_f *output*)

Allows a user to add a tile-able kernel to the OpenVX system.

Parameters

`in`	*context*	The handle to the implementation context.
`in`	*name*	The string which is to be used to match the kernel.

`in`	*enumeration*	The enumerated value of the kernel to be used by clients.
`in`	*func_ptr*	The process-local function pointer to be invoked.
`in`	*num_params*	The number of parameters for this kernel.
`in`	*input*	The pointer to a function which will validate the input parameters to this kernel.
`in`	*output*	The pointer to a function which will validate the output parameters to this kernel.

Note

Tiling Kernels do not have access to any of the normal node attributes listed in **vx_node_attribute_e**.

Postcondition

Call `vxAddParameterToKernel` for as many parameters as the function has, then call `vxFinalizeKernel`.

Return values

0	Indicates that an error occurred when adding the kernel.

Index