OpenCL (Open Computing Language) is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance compute servers, desktop computer systems and handheld devices.

The OpenCL Runtime

Command Queues [5.1]

cl_command_queue createCommandQueue ( cl_context context, cl_device_id device, cl_command_queue_properties properties, cl_int *errcode_ret )

cl_command_queue info

Properties: CL_QUEUE_PROFILING_ENABLE, CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE, cl_int clRetainCommandQueue ( cl_command_queue command_queue )

cl_int clReleaseCommandQueue ( cl_command_queue command_queue )

cl_int clGetCommandQueueInfo ( cl_command_queue command_queue, cl_int param_name, const size_t *param_value_size, size_t *param_value, void *user_data, cl_int *errcode_ret )

Properties: See clCreateContext

cl_int clRetainContext ( cl_context context )

cl_int clReleaseContext ( cl_context context )

cl_int clGetContextInfo ( cl_context context, cl_int context_info_param_name, const size_t *param_value_size, void *param_value, size_t *param_value_size_ret )

param name: CL_CONTEXT_REFERENCE_COUNT, CL_CONTEXT_DEVICES, CL_CONTEXT_NUM_DEVICES

Buffer Objects

Elements of a buffer object can be a scalar or vector data type or a user-defined structure. Elements are stored sequentially and are accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

cl_mem createBuffer ( cl_context context, cl_mem_flags flags, size_t size, void *host_ptr, cl_int *errcode_ret )

cl_mem createSubBuffer ( cl_mem buffer, cl_mem_flags flags, cl_mem_create_type_buffer create_type, cl_int *errcode_ret )

flags for clCreateBuffer and clCreateSubBuffer

cl_MEM_READ_WRITE, cl_MEM_WRITE_ONLY, cl_MEM_RDONLY, cl_MEM_ALLOC_HOST_PTR

Read, Write, Copy Buffer Objects [5.2.2]

cl_int enqueueReadBuffer ( cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_read, size_t offset, size_t cb, void *ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event )

cl_int enqueueWriteBuffer ( cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_write, size_t offset, size_t cb, void *ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event )

cl_int enqueueWriteBufferRect ( cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_write, size_t offset, size_t dest_size, size_t dest_row_pitch, size_t dest_slice_pitch, void *ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event )

Map Buffer Objects [5.2.3]

cl_int enqueueMapBuffer ( cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_map, cl_map_flags map_flags, size_t size, size_t offset, size_t cb, void *mapped_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event, cl_int *errcode_ret )

cl_int enqueueVariantMapBuffer ( cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_map, cl_mem_flags map_flags, cl_mem_map_info *map_info, void *mapped_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event, cl_int *errcode_ret )

cl_int enqueueUnmapMemObject ( cl_command_queue command_queue, cl_mem memobj, void *unmapped_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event )

Program Objects

Create Program Objects [5.6.4]

cl_program createProgramWithSource ( cl_context context, cl_int num_devices, const char **strings, void (*pfn_notify)(cl_device_id device, const char *source), void *user_data )

cl_program createProgramWithBinary ( cl_context context, cl_int num_devices, const cl_device_id *device_list, const size_t *lengths, const unsigned char **binaries, cl_int int _errcode_ret )

cl_int clRetainProgram ( cl_program program )

cl_int clReleaseProgram ( cl_program program )

Build Program Executable [5.6.2]

cl_int buildProgram ( cl_program program, cl_int num_devices, cl_device_id *device_list, const char **strings, void (*pfn_notify)(cl_device_id device, const char *source), void *user_data )

Build Options [5.6.3]

Preprocessor: (-D processed in order listed in clBuildProgram)

Optimization options:

- cl-opt-disable
- cl-md-enable
- cl-no-signed-zeros
- cl-finite-math-only
- cl-unsafe-math-optimizations

Map Buffer Objects [5.4.1-2]

cl_int clRetainMemObject ( cl_mem memobj )

cl_int clReleaseMemObject ( cl_mem memobj )

cl_int clSetMemObjectDestructorCallback ( cl_mem memobj, void (CL_CALLBACK *pfn_notify)(cl_mem memobj, void *user_data), void *user_data )

cl_int clEnqueueUnmapMemObject ( cl_command_queue command_queue, cl_mem memobj, void *unmapped_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event )

Query Buffer Object [5.4.3]

cl_int clGetMemObjectInfo ( cl_mem memobj, cl_mem_info info_size, void *param_value_size_ret, cl_mem_info info_value_size, void *param_value, void *user_data )

Map Program Objects [5.6.5]

cl_int clGetProgramInfo ( cl_program program, cl_int info_type, cl_mem_info info_size, void *param_value_size, void *param_value, size_t *param_value_size_ret )

Program Objects Contine >
### Built-in Scalar Data Types [6.1.1]

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>--</td>
<td>true (1) or false (0)</td>
</tr>
<tr>
<td>char</td>
<td>cl_char</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>unsigned char</td>
<td>cl_uchar</td>
<td>8-bit unsigned</td>
</tr>
<tr>
<td>short</td>
<td>cl_int</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>unsigned short</td>
<td>cl_ushort</td>
<td>16-bit unsigned</td>
</tr>
<tr>
<td>int</td>
<td>cl_int</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>unsigned int</td>
<td>cl_uint</td>
<td>32-bit unsigned</td>
</tr>
<tr>
<td>long</td>
<td>cl_long</td>
<td>64-bit signed</td>
</tr>
<tr>
<td>unsigned long</td>
<td>cl_ulong</td>
<td>64-bit unsigned</td>
</tr>
<tr>
<td>float</td>
<td>cl_float</td>
<td>32-bit float</td>
</tr>
<tr>
<td>half</td>
<td>cl_half</td>
<td>16-bit float (for storage only)</td>
</tr>
<tr>
<td>size_t</td>
<td>--</td>
<td>32- or 64-bit signed integer</td>
</tr>
<tr>
<td>ptrdiff_t</td>
<td>--</td>
<td>32- or 64-bit signed integer</td>
</tr>
<tr>
<td>intptr_t</td>
<td>--</td>
<td>signed integer</td>
</tr>
<tr>
<td>uintptr_t</td>
<td>--</td>
<td>unsigned integer</td>
</tr>
</tbody>
</table>

### Built-in Vector Data Types [6.1.2]

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>cl_char</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>wchar_t</td>
<td>cl_wchar</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>short</td>
<td>cl_short</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>USHORT</td>
<td>cl_ushort</td>
<td>16-bit unsigned</td>
</tr>
<tr>
<td>int</td>
<td>cl_int</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>UINT</td>
<td>cl_uint</td>
<td>32-bit unsigned</td>
</tr>
<tr>
<td>long</td>
<td>cl_long</td>
<td>64-bit signed</td>
</tr>
<tr>
<td>ULONG</td>
<td>cl_ulong</td>
<td>64-bit unsigned</td>
</tr>
<tr>
<td>float</td>
<td>cl_float</td>
<td>32-bit float</td>
</tr>
<tr>
<td>half</td>
<td>cl_half</td>
<td>16-bit float</td>
</tr>
</tbody>
</table>

### Other Built-in Data Types [6.1.3]

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image2d_t</td>
<td>2D image handle</td>
</tr>
<tr>
<td>image3d_t</td>
<td>3D image handle</td>
</tr>
<tr>
<td>sampler_t</td>
<td>sampler handle</td>
</tr>
<tr>
<td>event_t</td>
<td>event handle</td>
</tr>
</tbody>
</table>

### Reserved Data Types [6.1.4]

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>boolean vector</td>
</tr>
<tr>
<td>double, double</td>
<td>64-bit float, vector</td>
</tr>
<tr>
<td>half</td>
<td>16-bit vector</td>
</tr>
<tr>
<td>quad, quadn</td>
<td>128-bit float, vector</td>
</tr>
<tr>
<td>complex half, complex halfn</td>
<td>16-bit complex, vector</td>
</tr>
<tr>
<td>complex float, complex floatn</td>
<td>32-bit complex, vector</td>
</tr>
<tr>
<td>complex double, complex doublen, complex double imaginary, complex double imaginaryn</td>
<td>64-bit complex, vector</td>
</tr>
<tr>
<td>complex quad, complex quadn, complex quad imaginary, complex quad imaginaryn</td>
<td>128-bit complex, vector</td>
</tr>
<tr>
<td>float3x3n</td>
<td>n*n matrix of 32-bit floats</td>
</tr>
<tr>
<td>double3x3n</td>
<td>n*n matrix of 64-bit floats</td>
</tr>
<tr>
<td>long double, long doublen</td>
<td>64-128-bit float, vector</td>
</tr>
<tr>
<td>long long, long longn</td>
<td>128-bit signed</td>
</tr>
<tr>
<td>unsigned long long, unsigned long longn</td>
<td>128-bit unsigned</td>
</tr>
</tbody>
</table>

### Vector Component Addressing [6.1.7]

## Vector Components

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>float2 v;</td>
<td>vx, vx0</td>
<td>vy, vx1</td>
<td>vx0, vx1</td>
<td>vx0, vx2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float3 v;</td>
<td>vx, vx0</td>
<td>vy, vx1</td>
<td>vz, vx0</td>
<td>vz, vx1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float4 v;</td>
<td>vx, vx0</td>
<td>vy, vx1</td>
<td>vz, vx0</td>
<td>vz, vx1</td>
<td>vz, vx2</td>
<td>vz, vx3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float8 v;</td>
<td>vx, vx0</td>
<td>vx, vx1</td>
<td>vx, vx0</td>
<td>vx, vx1</td>
<td>vx, vx0</td>
<td>vx, vx1</td>
<td>vx, vx0</td>
<td>vx, vx1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float16 v;</td>
<td>vx, vx0</td>
<td>vx, vx1</td>
<td>vx, vx0</td>
<td>vx, vx1</td>
<td>vx, vx0</td>
<td>vx, vx1</td>
<td>vx, vx0</td>
<td>vx, vx1</td>
<td>vx, vx0</td>
<td>vx, vx1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Vector Addressing Equivalencies

Numeric indices are preceded by the letter s or S, e.g.: s1. Swizzling, duplication, and nesting are allowed, e.g.: v.yx, v.xx, v.lo.x

### Conversions & Type Casting Examples [6.2]

\[ T \ = \ convert\_T\_sat\_f\_b(\text{f}); \] \text{R} \text{ is rounding mode} 
\text{R} \text{ can be one of the following rounding modes:} 
\text{rte} \text{ to nearest even} \text{ rtp toward + infinity} 
\text{rtz toward zero} \text{ rtn toward -infinity}

### Operators [6.3]

These operators behave similarly as in C99 except that operands may include vector types when possible:

\[ + \quad - \quad \% \quad / \quad \div \quad \% \quad = \quad \& \quad \&\& \quad \text{|} \quad \text{^} \quad \text{>} \quad < \quad \geq \quad \leq \quad \neq \quad \text{op} \quad \text{sizeof} \]

### Address Space Qualifiers [6.5]

\text{global, global local, local} 
\text{constant, constant local} 
\text{private, private local}

### Function Qualifiers [6.7]

\text{__kernel, kernel} 
\text{__attribute__} \text{([vec\_type\_hint\(h\text{type}\)])} 
\text{type defaults to int} 
\text{__attribute__} \text{([work\_group\_size\(X, Y, Z\)])} 
\text{__attribute__} \text{([reqd\_work\_group\_size\(X, Y, Z\)])}
Preprocessor Directives & Macros [6.9]  
#pragma OPENCL_FP_CONTRACT on-off-switch  
-on-off-switch; ON, OFF, DEFAULT  

_attribute (aligned()) | __attribute__((aligned()))
_attribute (packed())   | __attribute__((packed()))
_attribute (endianness) | __attribute__((endianness))

### Specify Type Attributes [6.10.1]  
Use to specify attributes of enum, struct, and union types.

#### Math Constants [6.12.2]  
The values of the following symbolic constants are type float and are accurate within the precision of a single precision floating-point number.

- **HUGE_VAL**  
  Positive double expression, evaluates to infinity. Used as error value. 
  
- **HUGE_VALF**  
  Positive float expression, evaluates to infinity. Used as error value.

#### MAXFLOAT  
Value of max. non-infinite single-precision floating-point number.

### Common Built-in Functions [6.11.4]  
_T_ is type float or float (or optionally double, double, or half). 
Optional extensions enable double, double, and half types.

#### Interger Built-in Functions [6.11.3]  
_T_ is type char, char, uchar, short, short, ushort, ushort, int, int, uint, long, ulong, or ulong.  
_U_ is the unsigned version of _T_ and the scalar version of _T_.

### Math Built-in Functions [6.11.2]  
_T_ is type float or float (or optionally double, double, or half),  
int, uint, and ulong may be scalar when _T_ is scalar or qualifier _global, local, or private_. 
_H_ indicates that Half and Native variants are available by prepending “half” or “native_” to function name. 
Prototypes shown in purple are half_ and native_. 
Optional extensions enable double, double, and half types.

---

**OpenCL API 1.1 Quick Reference**

©2015 Khronos Group - Rev. 0715
Geometric Built-in Functions [6.11.5]

Vector types may have 3, or 4 components. Optional extensions enable double, half, and double.half.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>float dot (float p0, float p1)</td>
<td>Dot product</td>
<td>Vector distance</td>
</tr>
<tr>
<td>double dot (double p0, double p1)</td>
<td>Dot product</td>
<td>Vector distance</td>
</tr>
<tr>
<td>short dot (short p0, short p1)</td>
<td>Dot product</td>
<td>Vector distance</td>
</tr>
<tr>
<td>Vector distance (float p0, float p1)</td>
<td>Dot product</td>
<td>Vector distance</td>
</tr>
<tr>
<td>Double distance (double p0, double p1)</td>
<td>Dot product</td>
<td>Vector distance</td>
</tr>
<tr>
<td>Half distance (half p0, half p1)</td>
<td>Dot product</td>
<td>Vector distance</td>
</tr>
</tbody>
</table>

Relational Built-in Functions [6.11.6]

T is type float, char, char, uchar, uchar, short, short, ushort, ushort, int, int, uint, uint, long, long, ulong, or ulong (and optionally double, double, and half). U is type uchar, uchar, uchar, ushort, ushort, uint, uint, ulong, or ulong (and optionally double, double, and half).

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>int isequal (float x, float y)</td>
<td>Compare of x == y</td>
<td>Test for x == y and x != y</td>
</tr>
<tr>
<td>int isinalt (float x, float y)</td>
<td>Compare of x != y</td>
<td>Test for a NaN</td>
</tr>
<tr>
<td>int isinalt (short x, short y)</td>
<td>Compare of x != y</td>
<td>Test for a NaN</td>
</tr>
<tr>
<td>int isnormal (float x, float y)</td>
<td>Is normal</td>
<td>Test for a normal value</td>
</tr>
<tr>
<td>int isnormal (short x, short y)</td>
<td>Is normal</td>
<td>Test for a normal value</td>
</tr>
<tr>
<td>int isordered (float x, float y)</td>
<td>Is ordered</td>
<td>Test if arguments are ordered</td>
</tr>
<tr>
<td>int isordered (short x, short y)</td>
<td>Is ordered</td>
<td>Test if arguments are ordered</td>
</tr>
<tr>
<td>int isunordered (float x, float y)</td>
<td>Is unordered</td>
<td>Test if arguments are unordered</td>
</tr>
<tr>
<td>int isunordered (short x, short y)</td>
<td>Is unordered</td>
<td>Test if arguments are unordered</td>
</tr>
<tr>
<td>int isless (float x, float y)</td>
<td>Is less</td>
<td>Test for finite values</td>
</tr>
<tr>
<td>int isless (short x, short y)</td>
<td>Is less</td>
<td>Test for finite values</td>
</tr>
<tr>
<td>int islesseq (float x, float y)</td>
<td>Is less or equal</td>
<td>Test for finite values</td>
</tr>
<tr>
<td>int islesseq (short x, short y)</td>
<td>Is less or equal</td>
<td>Test for finite values</td>
</tr>
<tr>
<td>int islessgreater (float x, float y)</td>
<td>Is less than greater</td>
<td>Compare of x &lt; y</td>
</tr>
<tr>
<td>int islessgreater (short x, short y)</td>
<td>Is less than greater</td>
<td>Compare of x &lt; y</td>
</tr>
<tr>
<td>int isinfinite (float x)</td>
<td>Is infinite</td>
<td>Each bit of result is</td>
</tr>
<tr>
<td>int isinfinite (short x)</td>
<td>Is infinite</td>
<td>Each bit of result is</td>
</tr>
</tbody>
</table>

Atomic Functions [6.11.11, 9.4]

T is type int or unsigned int. T may also be type float for atomic _char, and type long or ulong for extended 64-bit atomic functions. Q is volatile _global or volatile _local, except Q must be volatile _global atomic _char when T is float.

The built-in atomic functions for 32-bit values begin with atomic _int, int, uint, uint, long, long, ulong, float, and optionally optionally double, double, and half. Optional extensions enable double, half, and double.half.

Asyc Copies and Prefetch Functions [6.11.10]

T is type char, char, uchar, uchar, short, short, ushort, ushort, int, int, uint, uint, long, long, ulong, float, and optionally optionally double, double, and half. Optional extensions enable double, half, and double.half.

Vector Data Load/Store Functions [6.11.7]

Q is an Address Space Qualifier listed in 6.5 unless otherwise noted. T defaults to the current rounding mode, or is one of the Rounding Modes listed in 6.2.3.2. T is type char, uchar, short, uchar, int, uint, uint, long, long, ulong, float, and optionally optionally double, double, and half. Optionally refers to the vector type of T. Optional extensions enable the double, half, and double.half.
OpenCL 1.1 Quick Reference

Page 5

Miscellaneous Vector Built-in Functions [6.11.12]

T and M mean the 2, 4, 8, or 16-component vectors of char, uchar, short, ushort, half, int, uint, long, ulong, float, double. Un means the built-in unsigned integer data types. For vec_step(), Tn also includes char1, uchar1, short1, ushort1, half1, int1, uint1, long1, ulong1, float1, double1.

- int vec_step(T n)
- int vec_step(Typename)

Takes a built-in scalar or vector data type argument and returns an integer value representing the number of elements in the scalar or vector.

- Tri Shuffle(T n, Un mask)
- Tri Shuffle(T x, Tri y, Un mask)

Constructs permutation of elements from one or two input vectors, return a vector with same element type as input & length that is the same as the shuffle mask.

OpenCL Graphics: Following is a subset of the OpenCL API specification that pertains to graphics.

Image Read and Write Built-in Functions [6.11.13, 9.5.9.6.8]

The built-in functions defined in this section can only be used with image memory objects created with clCreateImage2D or clCreateImage3D. sampler specifies the addressing and filtering mode to use. H = to enable read_image and write_image, enable extension cl_khr_fp64.

3D = To enable type image3d_t in write_image4, l, w, enable extension cl_khr_3d_image_writes.

- float4 read_imagef(image2d_t image, sampler_t sampler, int2 coord)
- float4 read_imagef(image2d_t image, sampler_t sampler, float2 coord)
- int read_imagei(image2d_t image, sampler_t sampler, int2 coord)
- int read_imagei(image2d_t image, sampler_t sampler, float2 coord)
- uint4 read_imageui(image2d_t image, sampler_t sampler, int2 coord)
- uint4 read_imageui(image2d_t image, sampler_t sampler, float2 coord)

Read an element from a 2D image

- float4 read_imagef(image3d_t image, sampler_t sampler, int3 coord)
- float4 read_imagef(image3d_t image, sampler_t sampler, float3 coord)
- int read_imagei(image3d_t image, sampler_t sampler, int3 coord)
- int read_imagei(image3d_t image, sampler_t sampler, float3 coord)
- uint4 read_imageui(image3d_t image, sampler_t sampler, int3 coord)
- uint4 read_imageui(image3d_t image, sampler_t sampler, float3 coord)

Read an element from a 3D image

- void write_image4f(image2d_t image, int2 coord, float4 color)
- void write_image4f(image2d_t image, int2 coord, uint4 color)

Write color value to (x, y) location specified by coord in the 2D image

- void write_image4f(image3d_t image, int3 coord, float4 color)
- void write_image4f(image3d_t image, int3 coord, uint4 color)

Write color at coord in the 3D image

Image Objects

Create Image Objects [5.3.1.3]

- cl_mem clCreateImage2D(cl context, cl_mem_flags flags, cl integer image_format, size_t image_width, size_t image_height, size_t image_row_pitch, void *host_ptr, cl integer *errcode_ret)

flags: also for clCreateImage3D, clGetSupportedImageFormats

- cl_mem clCreateImage3D(cl context, cl_mem_flags flags, cl integer image_format, size_t image_width, size_t image_height, size_t image_depth, size_t image_row_pitch, size_t image_slice_pitch, void *host_ptr, cl integer *errcode_ret)

flags: See clCreateImage2D

Query List of Supported Image Formats [5.3.2.3]

- cl_int clGetSupportedImageFormats(cl context, cl_mem mem, cl_mem_flags type, size_t *num_entries, cl integer *image_format, cl integer *num_image_formats)

flags: See clCreateImage2D

Copy Between Image, Buffer Objects [5.3.4]

- cl_int clEnqueueCopyImageToBuffer(cl context, cl_mem src_image, cl_mem dst_image, size_t src_offset, size_t dst_offset, size_t num_events_in_wait_list, cl integer *event_wait_list, cl integer *event_id)

- cl_int clEnqueueCopyBufferToImage(cl context, cl_mem dst_image, cl_mem src_image, size_t src_offset, size_t dst_offset, size_t num_events_in_wait_list, cl integer *event_wait_list, cl integer *event_id)

Map and Unmap Image Objects [5.3.5.5]

- void *clEnqueueMapImage(cl context, cl_mem image, cl_command_queue command_queue, cl_addressing_mode address_mode, cl_map_flags map_flags, size_t num_events_in_wait_list, cl integer *event_wait_list, cl integer *event_id)

- cl_int clRetainSampler(cl context, cl_sampler sampler)

Sampler Objects [5.3]

- cl_int clCreateSampler(cl context, cl_boolean normalized_coords, cl_addressing_mode addressing_mode, cl_filter_mode filter_mode, cl integer *errcode_ret)

Apply to image2d_t and image3d_t types to declare if the image memory object is being read or written by a kernel.

The default qualifier is __read_only.

- __read_only
- __write_only
- __read_write

Access Qualifiers [6.6]

Image Formats [5.3.1.1, 9.5.9.6.8]

Supported image formats: image_channel_order with image_channel_data_type

Built-in support:

- CL_RGBA: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT8, CL_SIGNED_INT8, CL_UNSIGNED_INT8
- CL_BGR: CL_UNORM_INT8

Optional support:

- CL_R: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT8, CL_SIGNED_INT8, CL_UNSIGNED_INT8
- CL_DEPTH: CL_UNORM_INT16, CL_HALF_FLOAT, CL_FLOAT, CL_HALF_FLOAT
- CL_EXTENSIONS: CL_HALF_FLOAT, CL_FLOAT, CL_HALF_FLOAT
- CL_SRGB: CL_HALF_FLOAT, CL_FLOAT, CL_HALF_FLOAT
- CL_NORMALIZED_COORDS: CL_HALF_FLOAT

Sampler Objects [5.3]

- cl_sampler clCreateSampler(cl context, cl_boolean normalized_coords, cl_addressing_mode addressing_mode, cl_filter_mode filter_mode, cl integer *errcode_ret)

- cl_int clCreateSampler(cl context, cl_sampler sampler)

- cl_int clGetSamplerInfo(cl context, cl_sampler sampler, cl_sampler_info info_name, size_t *param_value_size, void *param_value, size_t *param_value_size_ret)

- cl_int crileaseSampler(cl context, cl_sampler sampler)

- cl_sampler clCreateSampler(cl context, cl_boolean normalized_coords, cl_addressing_mode addressing_mode, cl_filter_mode filter_mode, cl integer *errcode_ret)

- cl_sampler clCreateSampler(cl context, cl_boolean normalized_coords, cl_addressing_mode addressing_mode, cl_filter_mode filter_mode, cl integer *errcode_ret)

- cl_sampler clCreateSampler(cl context, cl_boolean normalized_coords, cl_addressing_mode addressing_mode, cl_filter_mode filter_mode, cl integer *errcode_ret)
Sampler Declaration Fields [6.11.13.1]
The sampler can be passed as an argument to the kernel using cslSetKernelArg, or it can be a constant variable of type sampler_t declared in the program source.

const sampler_t <sampler-name> = <normalized-mode> | <address-mode> | <filter-mode>
normalized-mode:
CLK_NORMALIZED_COORDS (TRUE, FALSE)

address-mode:
CLK_ADDRESS (REPEAT, CLAMP, NONE)
CLK_ADDRESS (CLAMP_TO_EDGE, MIRRORED_REPEAT)
filter-mode:
CLK_FILTER_NEAREST, CLK_FILTER_LINEAR

OpenCL Class Diagram [5.13]
The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language (UML) notation. The diagram shows both nodes and edges which are classes and their relationships. As a simplification it shows only classes, and no attributes or operations.

OpenCL Device Architecture Diagram [3.3]
The table below shows memory regions with allocation and memory access capabilities.

OpenCL/OpenGL Sharing APIs
Creating OpenCL memory objects from OpenGL objects using clCreateFromGLBuffer, clCreateFromGLTexture2D, clCreateFromGLTexture3D, and clCreateFromGLRenderbuffer ensure that the storage of the OpenGL object will not be deleted while the corresponding OpenCL memory object exists.

CL Buffer Objects > GL Buffer Objects [8.9.2]
cl_mem clCreateFromGLBuffer(cl_context context, cl_mem_flags flags, GLuint bufferobj, int *errcode_ret)
flags: CL_MEM_READ, CL_MEM_READ_WRITE

CL Image Objects > GL Textures [8.8.3]
ccl_mem clCreateFromGLTexture2D(cl_context context, cl_mem_flags flags, GLenum texture_target, GLuint texture, cl_int *errcode_ret)
flags: See clCreateFromGLBuffer
texture_target: GL_TEXTURE_2D, GL_TEXTURE_RECTANGLE

CL Image Objects > GL Texture 3D [8.8.3]
ccl_mem clCreateFromGLTexture3D(cl_context context, cl_mem_flags flags, GLenum texture_target, GLuint texture, cl_int *errcode_ret)
flags: See clCreateFromGLBuffer
texture_target: GL_TEXTURE_3D

Share Objects [8.8.6]
ccl_int clEnqueueAcquireGLObjects(cl_int command_queue command_queue, cl_uint num_objects, const cl_mem *mem_objects, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_int *event)

OpenCL/Direct3D 10 Sharing APIs [9.10]
Creating OpenCL memory objects from Direct3D objects using clCreateFromD3D10Buffer, clCreateFromD3D10Texture2D, clCreateFromD3D10Texture3D, and clCreateFromD3D10Renderbuffer ensures that the storage of that Direct3D object will not be deleted while the corresponding OpenCL memory object exists.

cl_int clGetDeviceIDsFromD3D10KHR (cl_platform_id platform, cl_d3d10_device_device khr d3d device source, void *d3d_object, cl_d3d10_device_set khr d3d device set, cl_uint num_entries, cl_device_id *device devices, cl_uint *num_devices)
d3d_device_source: CL_D3D10_DEVICE_KHR, CL_D3D10_D3D_ADAPTER_KHR
d3d_object: ID3D10Device, ID3D10Adapter
d3d_device_set: CL_ALL_DEVICES, CL_D3D10, CL_PREFERRED_DEVICES

cl_int clCreateFromD3D10BufferKHR (cl_context context, cl_mem_flags flags, ID3D10Buffer *resource, cl_int *errcode_ret)
flags: CL_MEM_READ, CL_MEM_READ_WRITE

cl_int clCreateFromD3D10Texture2DKHR (cl_context context, cl_mem_flags flags, ID3D10Texture2D *resource, UINT *subresource, cl_int *errcode_ret)
flags: See clCreateFromD3D10BufferKHR

cl_int clCreateFromD3D10Texture3DKHR (cl_context context, cl_mem_flags flags, ID3D10Texture3D *resource, UINT *subresource, cl_int *errcode_ret)
flags: See clCreateFromD3D10BufferKHR

cl_int clCreateFromD3D10BufferKHR (cl_context context, cl_mem_flags flags, ID3D10Buffer *resource, cl_int *errcode_ret)
flags: CL_MEM_READ, CL_MEM_READ_WRITE

cl_int clCreateFromD3D10Texture2DKHR (cl_context context, cl_mem_flags flags, ID3D10Texture2D *resource, UINT *subresource, cl_int *errcode_ret)
flags: See clCreateFromD3D10BufferKHR

cl_int clCreateFromD3D10Texture3DKHR (cl_context context, cl_mem_flags flags, ID3D10Texture3D *resource, UINT *subresource, cl_int *errcode_ret)
flags: See clCreateFromD3D10BufferKHR

cl_int clCreateFromD3D10BufferKHR (cl_context context, cl_mem_flags flags, ID3D10Buffer *resource, cl_int *errcode_ret)
flags: CL_MEM_READ, CL_MEM_READ_WRITE

cl_int clCreateFromD3D10Texture2DKHR (cl_context context, cl_mem_flags flags, ID3D10Texture2D *resource, UINT *subresource, cl_int *errcode_ret)
flags: See clCreateFromD3D10BufferKHR

cl_int clCreateFromD3D10Texture3DKHR (cl_context context, cl_mem_flags flags, ID3D10Texture3D *resource, UINT *subresource, cl_int *errcode_ret)
flags: See clCreateFromD3D10BufferKHR