

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.

[n.n.n] refers to the section in the API Specification available at www.khronos.org/opencvl.

The OpenCL Runtime

Command Queues [5.1]

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

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_command_queue_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
```

param_name: CL_QUEUE_CONTEXT,
CL_QUEUE_DEVICE,
CL_QUEUE_REFERENCE_COUNT,
CL_QUEUE_PROPERTIES

```
cl_int clSetCommandQueueProperty (cl_command_queue
    command_queue, cl_command_queue_properties
    properties, cl_bool enable,
    cl_command_queue_properties *old_properties)
```

properties:
CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE,
CL_QUEUE_PROFILING_ENABLE

Memory Objects

Memory objects include *buffer* objects, and *image* objects. Refer to the Graphic page for information about image objects.

A buffer object stores a one-dimensional collection of elements. Elements of a buffer object can be a scalar data type (such as int, float), vector data type, or a user-defined structure, and are stored in sequential fashion and can be accessed using a pointer by a kernel executing on a device. The data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

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

flags: CL_MEM_READ_WRITE,
CL_MEM_WRITE_ONLY,
CL_MEM_READ_ONLY,
CL_MEM_USE_HOST_PTR,
CL_MEM_ALLOC_HOST_PTR,
CL_MEM_COPY_HOST_PTR

Program Objects

Create Program Objects [5.4.1]

```
cl_program clCreateProgramWithSource (
    cl_context context, cl_uint count, const char **strings,
    const size_t *lengths, cl_int *errcode_ret)
```

```
cl_program clCreateProgramWithBinary (
    cl_context context, cl_uint num_devices,
    const cl_device_id *device_list, const size_t *lengths,
    const unsigned char **binaries, cl_int *binary_status,
    cl_int *errcode_ret)
```

```
cl_int clRetainProgram (cl_program program)
```

```
cl_int clReleaseProgram (cl_program program)
```

Build Program Executable [5.4.2]

```
cl_int clBuildProgram (cl_program program,
    cl_uint num_devices, const cl_device_id *device_list,
    const char *options, void (*pfn_notify)
    (cl_program, void *user_data), void *user_data)
```

The OpenCL Platform Layer

The OpenCL platform layer which implements platform-specific features that allow applications to query OpenCL devices, device configuration information, and to create OpenCL contexts using one or more devices.

Contexts [4.3]

```
cl_context clCreateContext (
    cl_context_properties *properties, cl_uint num_devices,
    const cl_device_id *devices, void (*pfn_notify)
    (const char *errinfo, const void *private_info, size_t cb,
    void *user_data), void *user_data, cl_int *errcode_ret)
```

```
cl_context clCreateContextFromType (
    cl_context_properties *properties,
    cl_device_type device_type, void (*pfn_notify)
    (const char *errinfo, const void *private_info,
    size_t cb, void *user_data), void *user_data,
    cl_int *errcode_ret)
```

```
cl_int clRetainContext (cl_context context)
```

```
cl_int clReleaseContext (cl_context context)
```

```
cl_int clGetContextInfo (cl_context context,
    cl_context_info param_name,
    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_PROPERTIES

Querying Platform Info and Devices [4.1, 4.2]

```
cl_int clGetPlatformInfo (cl_platform_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
```

param_name: CL_PLATFORM_PROFILE,
CL_PLATFORM_VERSION

```
cl_int clGetDeviceIDs (cl_device_type device_type,
    cl_uint num_entries, cl_device_id *devices,
    cl_uint *num_devices)
```

device_type: CL_DEVICE_TYPE_CPU,
CL_DEVICE_TYPE_GPU,
CL_DEVICE_TYPE_ACCELERATOR,
CL_DEVICE_TYPE_DEFAULT, CL_DEVICE_TYPE_ALL

```
cl_int clGetDeviceInfo (cl_device_id device,
    cl_device_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
```

param_name: CL_DEVICE_TYPE,
CL_DEVICE_VENDOR_ID,
CL_DEVICE_MAX_COMPUTE_UNITS,
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS,
CL_DEVICE_MAX_WORK_ITEM_SIZES,
CL_DEVICE_MAX_WORK_GROUP_SIZE,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE,
CL_DEVICE_MAX_CLOCK_FREQUENCY,
CL_DEVICE_ADDRESS_BITS,
CL_DEVICE_MAX_MEM_ALLOC_SIZE,
CL_DEVICE_IMAGE_SUPPORT,
CL_DEVICE_MAX_READ_IMAGE_ARGS,
CL_DEVICE_MAX_WRITE_IMAGE_ARGS,
CL_DEVICE_IMAGE2D_MAX_WIDTH,
CL_DEVICE_IMAGE2D_MAX_HEIGHT,
CL_DEVICE_IMAGE3D_MAX_WIDTH,
CL_DEVICE_IMAGE3D_MAX_HEIGHT,
CL_DEVICE_IMAGE3D_MAX_DEPTH,
CL_DEVICE_MAX_SAMPLERS,
CL_DEVICE_MAX_PARAMETER_SIZE,
CL_DEVICE_MEM_BASE_ADDR_ALIGN,
CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE,
CL_DEVICE_SINGLE_FP_CONFIG,
CL_DEVICE_GLOBAL_MEM_CACHE_TYPE,
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE,
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE,
CL_DEVICE_GLOBAL_MEM_SIZE,
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE,
CL_DEVICE_MAX_CONSTANT_ARGS,
CL_DEVICE_LOCAL_MEM_TYPE,
CL_DEVICE_LOCAL_MEM_SIZE,
CL_DEVICE_ERROR_CORRECTION_SUPPORT,
CL_DEVICE_PROFILING_TIMER_RESOLUTION,
CL_DEVICE_ENDIAN_LITTLE,
CL_DEVICE_AVAILABLE,
CL_DEVICE_COMPILER_AVAILABLE,
CL_DEVICE_EXECUTION_CAPABILITIES,
CL_DEVICE_QUEUE_PROPERTIES,
CL_DEVICE_NAME,
CL_DEVICE_VENDOR,
CL_DRIVER_VERSION, CL_DEVICE_PROFILE,
CL_DEVICE_VERSION, CL_DEVICE_EXTENSIONS

Read, Write, Copy Buffer Objects [5.2.2 - 5.2.3]

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

```
cl_int clEnqueueCopyBuffer (
    cl_command_queue command_queue,
    cl_mem src_buffer, cl_mem dst_buffer, size_t src_offset,
    size_t dst_offset, size_t cb,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

```
cl_int clRetainMemObject (cl_mem memobj)
```

```
cl_int clReleaseMemObject (cl_mem memobj)
```

Map and Unmap Memory Objects [5.2.8]

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

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

Query Buffer Object [5.2.9]

```
cl_int clGetMemObjectInfo (cl_mem memobj,
    cl_mem_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
```

param_name: CL_MEM_TYPE,
CL_MEM_FLAGS, CL_MEM_SIZE,
CL_MEM_HOST_PTR, CL_MEM_MAP_COUNT,
CL_MEM_REFERENCE_COUNT, CL_MEM_CONTEXT

Build Options [5.4.3]

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

-D name,
-D name=definition,
-I dir

Math Intrinsics options:

-cl-single-precision-constant,
-cl-denorms-are-zero,

Optimization options:

-cl-opt-disable, -cl-strict-aliasing,
-cl-mad-enable, -cl-no-signed-zeros,
-cl-finite-math-only, -cl-fast-relaxed-math,
-cl-unsafe-math-optimizations

Warning request/suppress options:

-w, -Werror

Unload the OpenCL Compiler [5.4.4]

```
cl_int clUnloadCompiler (void)
```

Query Program Objects [5.4.5]

```
cl_int clGetProgramInfo (cl_program program,
    cl_program_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
```

param_value: CL_PROGRAM_REFERENCE_COUNT,
CL_PROGRAM_CONTEXT,
CL_PROGRAM_NUM_DEVICES,
CL_PROGRAM_DEVICES,
CL_PROGRAM_BINARY_SIZES,
CL_PROGRAM_BINARIES

```
cl_int clGetProgramBuildInfo (cl_program program,
    cl_device_id device,
    cl_program_build_info param_name,
    size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
```

param_name: CL_PROGRAM_BUILD_STATUS,
CL_PROGRAM_BUILD_OPTIONS,
CL_PROGRAM_BUILD_LOG

Kernel and Event Objects

Create Kernel Queries [5.5.1]

```
cl_kernel clCreateKernel (cl_program program,
    const char *kernel_name, cl_int *errcode_ret)
```

```
cl_int clCreateKernelsInProgram (cl_program program,
    cl_uint num_kernels, cl_kernel *kernels,
    cl_uint *num_kernels_ret)
```

```
cl_int clRetainKernel (cl_kernel kernel)
```

```
cl_int clReleaseKernel (cl_kernel kernel)
```

Kernel Arguments & Object Queries [5.5.2, 5.5.3]

```
cl_int clSetKernelArg (cl_kernel kernel, cl_uint arg_index,
    size_t arg_size, const void *arg_value)
```

```
cl_int clGetKernelInfo (cl_kernel kernel,
    cl_kernel_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
```

param_name: CL_KERNEL_FUNCTION_NAME,
CL_KERNEL_NUM_ARGS,
CL_KERNEL_REFERENCE_COUNT,
CL_KERNEL_CONTEXT, CL_KERNEL_PROGRAM

```
cl_int clGetKernelWorkGroupInfo (cl_kernel kernel,
    cl_device_id device,
    cl_kernel_work_group_info param_name,
    size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
```

param_name: CL_KERNEL_WORK_GROUP_SIZE,
CL_KERNEL_COMPILE_WORK_GROUP_SIZE

Execute Kernels [5.6]

```
cl_int clEnqueueNDRangeKernel (
    cl_command_queue command_queue, cl_kernel kernel,
    cl_uint work_dim, const size_t *global_work_offset,
    const size_t *global_work_size,
    const size_t *local_work_size,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

```
cl_int clEnqueueTask (
    cl_command_queue command_queue, cl_kernel kernel,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

```
cl_int clEnqueueNativeKernel (cl_command_queue
    command_queue, void (*user_func)(void *),
    void *args, size_t cb_args, cl_uint num_mem_objects,
    const cl_mem *mem_list, const void **args_mem_loc,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

Event Objects [5.7]

```
cl_int clWaitForEvents (
    cl_uint num_events, const cl_event *event_list)
```

```
cl_int clGetEventInfo (
    cl_event event, cl_event_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
```

```
cl_int clRetainEvent (cl_event event)
```

```
cl_int clReleaseEvent (cl_event event)
```

Out-of-order Execution of Kernels & Memory Object Commands [5.8]

```
cl_int clEnqueueMarker (
    cl_command_queue command_queue, cl_event *event)
```

```
cl_int clEnqueueWaitForEvents (
    cl_command_queue command_queue,
    cl_uint num_events, const cl_event *event_list)
```

```
cl_int clEnqueueBarrier (
    cl_command_queue command_queue)
```

Profile Operations on Memory Objects & Kernels [5.9]

```
cl_int clGetEventProfilingInfo (cl_event event,
    cl_profiling_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
```

param_name: CL_PROFILING_COMMAND_QUEUED,
CL_PROFILING_COMMAND_SUBMIT,
CL_PROFILING_COMMAND_START,
CL_PROFILING_COMMAND_END

Flush and Finish [5.10]

```
cl_int clFlush (cl_command_queue command_queue)
```

```
cl_int clFinish (cl_command_queue command_queue)
```

param_name: CL_EVENT_COMMAND_QUEUE,
CL_EVENT_COMMAND_TYPE,
CL_EVENT_COMMAND_EXECUTION_STATUS,
CL_EVENT_REFERENCE_COUNT

Supported Data Types

Built-in Scalar Data Types [6.1.1]

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	cl_char	8-bit signed
unsigned char, uchar	cl_uchar	8-bit unsigned
short	cl_short	16-bit signed
unsigned short, ushort	cl_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	cl_long	64-bit signed
unsigned long, ulong	cl_ulong	64-bit unsigned
float	cl_float	32-bit float
half	cl_half	16-bit float
size_t	--	32- or 64-bit unsigned integer
ptrdiff_t	--	32- or 64-bit signed integer
intptr_t	--	signed integer
uintptr_t	--	unsigned integer
void	--	void

Built-in Vector Data Types [6.1.2]

OpenCL Type	API Type	Description
char _n	cl_char _n	8-bit signed
uchar _n	cl_uchar _n	8-bit unsigned
short _n	cl_short _n	16-bit signed
ushort _n	cl_ushort _n	16-bit unsigned
int _n	cl_int _n	32-bit signed
uint _n	cl_uint _n	32-bit unsigned
long _n	cl_long _n	64-bit signed
ulong _n	cl_ulong _n	64-bit unsigned
float _n	cl_float _n	32-bit float

Other Built-in Data Types [6.1.3]

OpenCL Type	Description
image2d_t	2D image handle
image3d_t	3D image handle
sampler_t	sampler handle
event_t	event handle

Reserved Data Types [6.1.4]

OpenCL Type	Description
bool _n	boolean vector
double, double _n	64-bit float, vector
half _n	16-bit float, vector
quad, quad _n	128-bit float, vector
complex half, complex half _n imaginary half, imaginary half _n	16-bit complex, vector
complex float, complex float _n imaginary float, imaginary float _n	32-bit complex, vector
complex double, complex double _n imaginary double, imaginary double _n	64-bit complex, vector
complex quad, complex quad _n imaginary quad, imaginary quad _n	128-bit complex, vector
float _n x _m	<i>n</i> * <i>m</i> matrix of 32-bit floats
double _n x _m	<i>n</i> * <i>m</i> matrix of 64-bit floats
long double, long double _n	64 - 128-bit float, vector
long long, long long _n	128-bit signed
unsigned long long, unsigned long long, ulong long _n	128-bit unsigned

Vector Component Addressing [6.1.7]

The components of a vector may be addressed as shown below or as shown in the table of equivalencies.

Vector Components

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
float2 v;	v.x, v.s0	v.y, v.s1														
float4 v;	v.x, v.s0	v.y, v.s1	v.z, v.s2	v.w, v.s3												
float8 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7								
float16 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7	v.s8	v.s9	v.sa, v.sA	v.sb, v.sB	v.sc, v.sC	v.sd, v.sD	v.se, v.sE	v.sf, v.sF

Vector Addressing Equivalencies

	v.lo	v.hi	v.odd	v.even
float2	v.x, v.s0	v.y, v.s1	v.y, v.s1	v.x, v.s0
float4	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz
float8	v.s0123	v.s4567	v.s1357	v.s0246
float16	v.s01234567	v.s89abcdef	v.s13579bdf	v.s02468ace

When addressing vector components by numeric indices, they must be preceded by the letter s or S, e.g.: s1.

Swizzling, duplication, and nesting are allowed, e.g.: vxy, vxx, v.lo.x

Conversions and Type Casting Examples [6.2]

```
T a = (T)b; // scalar types only
T a = convert_T(b);
T a = convert_T_R(b);
T a = convert_T_sat_R(b);
T a = as_T(b);
```

Rounding Modes [6.2.3.2]

R can be:
_rte Round to nearest even
_rtz Round toward zero
_rtp Round toward positive infinity
_rtn Round toward negative infinity

Operators [6.3]

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

```
+ - * % / -- ++ == != &
~ ^ > < >= <= | ! && ||
?: >> << , = op= sizeof
```

Address Space Qualifiers [6.5]

```
__global __local
__constant __private
```

Function Qualifiers [6.7]

```
__kernel
__attribute__((vec_type_hint(int)))
__attribute__((work_group_size_hint(X, Y, Z)))
__attribute__((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
```

Predefined Macro Names

```
__FILE__ Current source file
__LINE__ Line number
__OPENCL_VERSION__ Integer version number
__ENDIAN_LITTLE__ 1 if device is little endian
__ROUNDING_MODE__ Current rounding mode (default "rte")
__kernel_exec(X, typen) Same as __kernel_attribute__((work_group_size_hint(X, 1, 1))) \
__attribute__((vec_type_hint(typen)))
__IMAGE_SUPPORT__ 1 if images are supported,
__FAST_RELAXED_MATH__ 1 if -cl-fast-relaxed-math optimization option is specified
```

OpenCL™ API 1.0 Quick Reference Card

Work-Item Built-in Functions [6.11.1]

D is dimension index.

<code>uint get_work_dim ()</code>	Num. of dimensions in use
<code>size_t get_global_size (uint <i>D</i>)</code>	Num. of global work-items
<code>size_t get_global_id (uint <i>D</i>)</code>	Global work-item ID value
<code>size_t get_local_size (uint <i>D</i>)</code>	Num. of local work-items
<code>size_t get_local_id (uint <i>D</i>)</code>	Local work-item ID
<code>size_t get_num_groups (uint <i>D</i>)</code>	Num. of work-groups
<code>size_t get_group_id (uint <i>D</i>)</code>	Returns the work-group ID

Floating Point Math Constants [6.11.2]

MAXFLOAT	Value of maximum non-infinite single-precision floating-point number.
HUGE_VALF	Positive float constant expression. HUGE_VALF evaluates to +infinity. Used as an error value.
INFINITY	Constant expression of type float representing positive or unsigned infinity.
NAN	Constant expression of type float representing a quiet NaN.

Integer Built-in Functions [6.11.3]

T is type char, charn, uchar, ucharn, short, shortn, ushort, ushortn, int, intrn, uint, uintn, long, longn, ulong, ulongn, or ulongn. *U* refers to the unsigned version of *T*.

<code>U abs (T x)</code>	<i>x</i>
<code>U abs_diff (T x, T y)</code>	<i>x</i> - <i>y</i> without modulo overflow
<code>T add_sat (T x, T y)</code>	<i>x</i> + <i>y</i> and saturates the result
<code>T hadd (T x, T y)</code>	(<i>x</i> + <i>y</i>) >> 1 without modulo overflow
<code>T rhadd (T x, T y)</code>	(<i>x</i> + <i>y</i> + 1) >> 1
<code>T clz (T x)</code>	Number of leading 0-bits in <i>x</i>
<code>T mad_hi (T a, T b, T c)</code>	mul_hi(<i>a</i> , <i>b</i>) + <i>c</i>
<code>T mad24 (T a, T b, T c)</code>	OPT Multiply 24-bit integer values <i>a</i> and <i>b</i> and add the 32-bit integer result to 32-bit integer <i>c</i>
<code>T mad_sat (T a, T b, T c)</code>	<i>a</i> * <i>b</i> + <i>c</i> and saturates the result
<code>T max (T x, T y)</code>	<i>y</i> if <i>x</i> < <i>y</i> , otherwise it returns <i>x</i>

<code>T min (T x, T y)</code>	<i>y</i> if <i>y</i> < <i>x</i> , otherwise it returns <i>x</i>
<code>T mul_hi (T x, T y)</code>	high half of the product of <i>x</i> and <i>y</i>
<code>T mul24 (T a, T b)</code>	OPT Multiply 24-bit integer values <i>a</i> and <i>b</i>
<code>T rotate (T v, T i)</code>	result[<i>indx</i>] = v[<i>indx</i>] << i[<i>indx</i>]
<code>T sub_sat (T x, T y)</code>	<i>x</i> - <i>y</i> and saturates the result
<code>shortn upsample (charn hi, ucharn lo)</code>	result[<i>i</i>] = ((short)hi[<i>i</i>] << 8) lo[<i>i</i>]
<code>ushortn upsample (ucharn hi, ucharn lo)</code>	result[<i>i</i>] = ((ushort)hi[<i>i</i>] << 8) lo[<i>i</i>]
<code>intrn upsample (shortn hi, ushortn lo)</code>	result[<i>i</i>] = ((int)hi[<i>i</i>] << 16) lo[<i>i</i>]
<code>uintn upsample (ushortn hi, ushortn lo)</code>	result[<i>i</i>] = ((uint)hi[<i>i</i>] << 16) lo[<i>i</i>]
<code>longn upsample (intrn hi, uintn lo)</code>	result[<i>i</i>] = ((long)hi[<i>i</i>] << 32) lo[<i>i</i>]
<code>ulongnn upsample (uintn hi, uintn lo)</code>	result[<i>i</i>] = ((ulong)hi[<i>i</i>] << 32) lo[<i>i</i>]

Common Built-in Functions [6.11.4]

T is type float or floatn (or optionally double, doublen, half, or halfn).

<code>T clamp (T x, T min, T max)</code> <code>floatn clamp (floatn x, float min, float max)</code> <code>doublen clamp (doublen x, double min, double max)</code> <code>halfn clamp (halfn x, half min, half max)</code>	Clamp <i>x</i> to range given by <i>min</i> , <i>max</i>
<code>T degrees (T radians)</code>	radians to degrees
<code>T max (T x, T y)</code> <code>floatn max (floatn x, float y)</code> <code>doublen max (doublen x, double y)</code> <code>halfn max (halfn x, half y)</code>	Max of <i>x</i> and <i>y</i>
<code>T min (T x, T y)</code> <code>floatn min (floatn x, float y)</code> <code>doublen min (doublen x, double y)</code> <code>halfn min (halfn x, half y)</code>	Min of <i>x</i> and <i>y</i>
<code>T mix (T x, T y)</code> <code>floatn mix (floatn x, float y)</code> <code>doublen mix (doublen x, double y)</code> <code>halfn mix (halfn x, half y)</code>	Linear blend of <i>x</i> and <i>y</i>
<code>T radians (T degrees)</code>	degrees to radians
<code>T step (T edge, T x)</code> <code>floatn step (float edge, floatn x)</code> <code>doublen step (double edge, doublen x)</code> <code>halfn step (half edge, halfn x)</code>	0.0 if <i>x</i> < <i>edge</i> , else 1.0
<code>T smoothstep (T edge0, T edge1, T x)</code> <code>floatn smoothstep (float edge0, float edge1, floatn x)</code> <code>doublen smoothstep (double edge0, double edge1, doublen x)</code> <code>halfn smoothstep (half edge0, half edge1, halfn x)</code>	Step and interpolate
<code>T sign (T x)</code>	Sign of <i>x</i>

Math Built-in Functions [6.11.5]

T is type float or floatn (or optionally double, doublen, half, or halfn). *intrn*, *uintn*, and *ulongn* must be scalar when *T* is scalar. The symbol **HN** indicates that Half and Native variants are available by prepending "half_" or "native_" to the function name, as in half_cos() and native_cos().

<code>T acos (T)</code>	Arc cosine
<code>T acosh (T)</code>	Inverse hyperbolic cosine
<code>T acospi (T x)</code>	acos (<i>x</i>) / π
<code>T asin (T)</code>	Arc sine
<code>T asinh (T)</code>	Inverse hyperbolic sine
<code>T asinpi (T x)</code>	asin (<i>x</i>) / π
<code>T atan (T y_over_x)</code>	Arc tangent
<code>T atan2 (T y, T x)</code>	Arc tangent of <i>y</i> / <i>x</i>
<code>T atanh (T)</code>	Hyperbolic arc tangent
<code>T atanpi (T x)</code>	atan (<i>x</i>) / π
<code>T atan2pi (T x, T y)</code>	atan2 (<i>x</i> , <i>y</i>) / π
<code>T cbrt (T)</code>	cube root
<code>T ceil (T)</code>	Round to integer toward + infinity
<code>T copysign (T x, T y)</code>	<i>x</i> with sign changed to sign of <i>y</i>
<code>T cos (T)</code>	HN cosine
<code>T cosh (T)</code>	hyperbolic cosine
<code>T cospi (T x)</code>	cos (π <i>x</i>)
<code>T half_divide (T x, T y)</code> <code>T native_divide (T x, T y)</code>	<i>x</i> / <i>y</i>
<code>T erfc (T)</code>	Complementary error function
<code>T erf (T)</code>	Calculates error function of <i>T</i>
<code>T exp (T x)</code>	HN Exponential base e
<code>T exp2 (T)</code>	HN Exponential base 2

<code>T exp10 (T)</code>	HN Exponential base 10
<code>T expm1 (T x)</code>	e ^{<i>x</i>} - 1.0
<code>T fabs (T)</code>	Absolute value
<code>T fdim (T x, T y)</code>	"Positive difference" between <i>x</i> and <i>y</i>
<code>T floor (T)</code>	Round to integer toward - infinity
<code>T fma (T a, T b, T c)</code>	Multiply and add, then round
<code>T fmax (T x, T y)</code> <code>halfn fmax (halfn x, half y)</code>	Return <i>y</i> if <i>x</i> < <i>y</i> , otherwise it returns <i>x</i>
<code>T fmin (T x, T y)</code> <code>halfn fmin (halfn x, half y)</code>	Return <i>y</i> if <i>y</i> < <i>x</i> , otherwise it returns <i>x</i>
<code>T fmod (T x, T y)</code>	Modulus. Returns <i>x</i> - <i>y</i> * trunc (<i>x</i> / <i>y</i>)
<code>T fract (T x, T *iptr)</code>	Fractional value in <i>x</i>
<code>T frexp (T x, intrn *exp)</code>	Extract mantissa and exponent
<code>T hypot (T x, T y)</code>	square root of <i>x</i> ² + <i>y</i> ²
<code>intrn flogb (T x)</code>	Return exponent as an integer value
<code>T ldexp (T x, intrn n)</code> <code>T ldexp (T x, int n)</code>	<i>x</i> * 2 ^{<i>n</i>}
<code>T lgamma (T x)</code> <code>T lgamma_r (T x, intrn *signp)</code>	Log gamma function
<code>T log (T)</code>	HN Natural logarithm
<code>T log2 (T)</code>	HN Base 2 logarithm
<code>T log10 (T)</code>	HN Base 10 logarithm
<code>T log1p (T x)</code>	ln (1.0 + <i>x</i>)
<code>T logb (T x)</code>	exponent of <i>x</i>
<code>T mad (T a, T b, T c)</code>	Approximates <i>a</i> * <i>b</i> + <i>c</i>
<code>T modf (T x, T *iptr)</code>	Decompose a floating-point number

<code>float nan (uintn nancode)</code> <code>floatn nan (uintn nancode)</code> <code>halfn nan (ushortn nancode)</code> <code>doublen nan (ulongn nancode)</code> <code>doublen nan (uintn nancode)</code>	Quiet NaN
<code>T nextafter (T x, T y)</code>	Next representable floating-point value following <i>x</i> in the direction of <i>y</i>
<code>T pow (T x, T y)</code>	Compute <i>x</i> to the power of <i>y</i> (<i>x</i> ^{<i>y</i>})
<code>T pown (T x, intrn y)</code>	Compute <i>x</i> ^{<i>y</i>} , where <i>y</i> is an integer
<code>T powr (T x, T y)</code>	HN Compute <i>x</i> ^{<i>y</i>} , where <i>x</i> is >= 0
<code>T half_recip (T x)</code> <code>T native_recip (T x)</code>	1 / <i>x</i>
<code>T remainder (T x, T y)</code>	Floating point remainder function
<code>T remquo (T x, T y, intrn *quo)</code>	Floating point remainder and quotient function
<code>T rint (T)</code>	Round integer to nearest even integer
<code>T rootn (T x, intrn y)</code>	Compute <i>x</i> to the power of 1/ <i>y</i>
<code>T round (T x)</code>	Integral value nearest to <i>x</i> rounding
<code>T rsqrt (T)</code>	HN Inverse square root
<code>T sin (T)</code>	HN sine
<code>T sincos (T x, T *cosval)</code>	sine and cosine of <i>x</i>
<code>T sinh (T)</code>	hyperbolic sine
<code>T sinpi (T x)</code>	sin (π <i>x</i>)
<code>T sqrt (T)</code>	HN square root
<code>T tan (T)</code>	HN tangent
<code>T tanh (T)</code>	hyperbolic tangent
<code>T tanpi (T x)</code>	tan (π <i>x</i>)
<code>T tgamma (T)</code>	gamma function
<code>T trunc (T)</code>	Round to integer toward zero

Geometric Built-in Functions [6.11.5]

Vector types may have 2 or 4 components.

<code>float4 cross (float4 p0, float4 p1)</code> <code>double4 cross (double4 p0, double4 p1)</code> <code>half4 cross (half4 p0, half4 p1)</code>	Cross product
<code>float dot (float p0, float p1)</code> <code>float dot (floatn p0, floatn p1)</code> <code>double dot (double p0, double p1)</code> <code>double dot (doublen p0, doublen p1)</code> <code>half dot (half p0, half p1)</code> <code>half dot (halfn p0, halfn p1)</code>	Dot product

<code>float distance (float p0, float p1)</code> <code>float distance (floatn p0, floatn p1)</code> <code>double distance (double p0, double p1)</code> <code>double distance (doublen p0, doublen p1)</code> <code>half distance (half p0, half p1)</code> <code>half distance (halfn p0, halfn p1)</code>	Vector distance
<code>float length (float p)</code> <code>float length (floatn p)</code> <code>double length (double p)</code> <code>double length (doublen p)</code> <code>half length (half p)</code> <code>half length (halfn p)</code>	Vector length

<code>float normalize (float p)</code> <code>floatn normalize (floatn p)</code> <code>double normalize (double p)</code> <code>doublen normalize (doublen p)</code> <code>half normalize (half p)</code> <code>halfn normalize (halfn p)</code>	Normal vector length 1
<code>float fast_distance (float p0, float p1)</code> <code>float fast_distance (floatn p0, floatn p1)</code>	Vector distance
<code>float fast_length (float p)</code> <code>float fast_length (floatn p)</code>	Vector length
<code>float fast_normalize (float p)</code> <code>floatn fast_normalize (floatn p)</code>	Normal vector length 1

Each occurrence of *T* within a function call must be the same. In vector types, *n* is 2, 4, 8, or 16 unless otherwise specified.

HN= Half and Native variants are available. **half_** and **native_** variants are shown in purple.

OPT = Optional function.

More built-in functions >

Relational Built-in Functions [6.11.6]

T is type float, floatn, char, charn, uchar, uchar_n, short, shortn, ushort, ushortn, int, intr, uint, uintn, long, longn, ulong, or ulongn and optionally double, doublen. *S* is type char, charn, short, shortn, int, intr, long, or longn. *U* is type uchar, uchar_n, ushort, ushortn, uint, uintn, ulong, or ulongn.

int isequal (float <i>x</i> , float <i>y</i>) intrn isequal (floatn <i>x</i> , floatn <i>y</i>) int isequal (double <i>x</i> , double <i>y</i>) longn isequal (doublen <i>x</i> , doublen <i>y</i>) int isequal (half <i>x</i> , half <i>y</i>) shortn isequal (halfn <i>x</i> , halfn <i>y</i>)	Compare of $x == y$
int isnotequal (float <i>x</i> , float <i>y</i>) intrn isnotequal (floatn <i>x</i> , floatn <i>y</i>) int isnotequal (double <i>x</i> , double <i>y</i>) longn isnotequal (doublen <i>x</i> , doublen <i>y</i>) int isnotequal (half <i>x</i> , half <i>y</i>) shortn isnotequal (halfn <i>x</i> , halfn <i>y</i>)	Compare of $x != y$
int isgreater (float <i>x</i> , float <i>y</i>) intrn isgreater (floatn <i>x</i> , floatn <i>y</i>) int isgreater (double <i>x</i> , double <i>y</i>) longn isgreater (doublen <i>x</i> , doublen <i>y</i>) int isgreater (half <i>x</i> , half <i>y</i>) shortn isgreater (halfn <i>x</i> , halfn <i>y</i>)	Compare of $x > y$
int isgreaterequal (float <i>x</i> , float <i>y</i>) intrn isgreaterequal (floatn <i>x</i> , floatn <i>y</i>) int isgreaterequal (double <i>x</i> , double <i>y</i>) longn isgreaterequal (doublen <i>x</i> , doublen <i>y</i>) int isgreaterequal (half <i>x</i> , half <i>y</i>) shortn isgreaterequal (halfn <i>x</i> , halfn <i>y</i>)	Compare of $x >= y$
int isless (float <i>x</i> , float <i>y</i>) intrn isless (floatn <i>x</i> , floatn <i>y</i>) int isless (double <i>x</i> , double <i>y</i>) longn isless (doublen <i>x</i> , doublen <i>y</i>) int isless (half <i>x</i> , half <i>y</i>) shortn isless (halfn <i>x</i> , halfn <i>y</i>)	Compare of $x < y$
int islessequal (float <i>x</i> , float <i>y</i>) intrn islessequal (floatn <i>x</i> , floatn <i>y</i>) int islessequal (double <i>x</i> , double <i>y</i>) longn islessequal (doublen <i>x</i> , doublen <i>y</i>) int islessequal (half <i>x</i> , half <i>y</i>) shortn islessequal (halfn <i>x</i> , halfn <i>y</i>)	Compare of $x <= y$
int islessgreater (float <i>x</i> , float <i>y</i>) intrn islessgreater (floatn <i>x</i> , floatn <i>y</i>) int islessgreater (double <i>x</i> , double <i>y</i>) longn islessgreater (doublen <i>x</i> , doublen <i>y</i>) int islessgreater (half <i>x</i> , half <i>y</i>) shortn islessgreater (halfn <i>x</i> , halfn <i>y</i>)	Compare of $(x < y) (x > y)$
int isfinite (float) intrn isfinite (floatn) int isfinite (double) longn isfinite (doublen) int isfinite (half) shortn isfinite (halfn)	Test for finite value

int isinf (float) intrn isinf (floatn) int isinf (double) longn isinf (doublen) int isinf (half) shortn isinf (halfn)	Test for +ve or -ve infinity
int isnan (float) intrn isnan (floatn) int isnan (double) longn isnan (doublen) int isnan (half) shortn isnan (halfn)	Test for a NaN
int isnormal (float) intrn isnormal (floatn) int isnormal (double) longn isnormal (doublen) int isnormal (half) shortn isnormal (halfn)	Test for a normal value
int isordered (float <i>x</i> , float <i>y</i>) intrn isordered (floatn <i>x</i> , floatn <i>y</i>) int isordered (double <i>x</i> , double <i>y</i>) longn isordered (doublen <i>x</i> , doublen <i>y</i>) int isordered (half <i>x</i> , half <i>y</i>) shortn isordered (halfn <i>x</i> , halfn <i>y</i>)	Test if arguments are ordered
int isunordered (float <i>x</i> , float <i>y</i>) intrn isunordered (floatn <i>x</i> , floatn <i>y</i>) int isunordered (double <i>x</i> , double <i>y</i>) longn isunordered (doublen <i>x</i> , doublen <i>y</i>) int isunordered (half <i>x</i> , half <i>y</i>) shortn isunordered (halfn <i>x</i> , halfn <i>y</i>)	Test if arguments are unordered
int signbit (float) intrn signbit (floatn) int signbit (double) longn signbit (doublen) int signbit (half) shortn signbit (halfn)	Test for sign bit
int any (<i>S x</i>)	1 if MSB in any component of <i>x</i> is set; else 0
int all (<i>S x</i>)	1 if MSB in all components of <i>x</i> are set; else 0
<i>T</i> bitselect (<i>T a</i> , <i>T b</i> , <i>T c</i>) halfn bitselect (halfn <i>a</i> , halfn <i>b</i> , halfn <i>c</i>) doublen bitselect (doublen <i>a</i> , doublen <i>b</i> , doublen <i>c</i>)	Each bit of result is corresponding bit of <i>a</i> if corresponding bit of <i>c</i> is 0
<i>T</i> select (<i>T a</i> , <i>T b</i> , <i>S c</i>) <i>T</i> select (<i>T a</i> , <i>T b</i> , <i>U c</i>) doublen select (doublen, doublen, longn)	For each component of a vector type, result[i] = if MSB of <i>c</i> [i] is set ? <i>b</i> [i] : <i>a</i> [i] For scalar type, result = <i>c</i> ? <i>b</i> : <i>a</i>

Vector Data Load/Store Built-in Functions [6.11.7]

Q is an Address Space Qualifier listed in 6.5 unless otherwise noted. *R* 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, ushort, int, uint, long, ulong, half, or float. *Tn* refers to the vector form of type *T*.

<i>Tn</i> vloadn (size_t <i>offset</i> , const <i>Q T *p</i>)	Read vector data from memory
void vstoren (<i>Tn data</i> , size_t <i>offset</i> , <i>Q T *p</i>)	Write vector data to memory (<i>Q</i> in this function cannot be __constant)
float vload_half (size_t <i>offset</i> , const <i>Q half *p</i>)	Read a half from memory
floatn vload_halfn (size_t <i>offset</i> , const <i>Q half *p</i>)	Read multiple halves from memory
void vstore_half (float <i>data</i> , size_t <i>offset</i> , <i>Q half *p</i>) void vstore_half_R (float <i>data</i> , size_t <i>offset</i> , <i>Q half *p</i>) void vstore_half (double <i>data</i> , size_t <i>offset</i> , <i>Q half *p</i>) void vstore_half_R (double <i>data</i> , size_t <i>offset</i> , <i>Q half *p</i>)	Write a half to memory (<i>Q</i> in this function cannot be __constant)
void vstore_halfn (floatn <i>data</i> , size_t <i>offset</i> , <i>Q half *p</i>) void vstore_halfn_R (floatn <i>data</i> , size_t <i>offset</i> , <i>Q half *p</i>) void vstore_halfn (doublen <i>data</i> , size_t <i>offset</i> , <i>Q half *p</i>) void vstore_halfn_R (doublen <i>data</i> , size_t <i>offset</i> , <i>Q half *p</i>)	Write a half vector to memory (<i>Q</i> in this function cannot be __constant)
floatn vloada_halfn (size_t <i>offset</i> , const <i>Q half *p</i>)	sizeof (floatn) bytes of data read from location (<i>p</i> + (<i>offset</i> * <i>n</i>))
void vstorea_halfn (floatn <i>data</i> , size_t <i>offset</i> , <i>Q half *p</i>) void vstorea_halfn_R (floatn <i>data</i> , size_t <i>offset</i> , <i>Q half *p</i>) void vstorea_halfn (doublen <i>data</i> , size_t <i>offset</i> , <i>Q half *p</i>) void vstorea_halfn_R (doublen <i>data</i> , size_t <i>offset</i> , <i>Q half *p</i>)	Write a half vector to vector-aligned memory (<i>Q</i> in this function cannot be __constant)

Async Copies and Prefetch Built-in Functions [6.11.11]

T is type char, charn, uchar, uchar_n, short, shortn, ushort, ushortn, int, intr, uint, uintn, long, longn, ulong, ulongn, float, floatn, and optionally double, doublen.

event_t async_work_group_copy (<i>_local T *dst</i> , const <i>_global T *src</i> , size_t <i>num_elements</i> , event_t <i>event</i>) event_t async_work_group_copy (<i>_global T *dst</i> , const <i>_local T *src</i> , size_t <i>num_elements</i> , event_t <i>event</i>)	Copies <i>T</i> elements from <i>src</i> to <i>dst</i>
void wait_group_events (int <i>num_events</i> , event_t * <i>event_list</i>)	Wait for events that identify the async_work_group_copy operations to complete.
void prefetch (const <i>_global T *p</i> , size_t <i>num_elements</i>)	Prefetch <i>num_elements</i> * sizeof(<i>T</i>) bytes into the global cache.

Optional Extension: Atomic Functions [9.5]

Q is qualifier __global or __local. *T* is type int or unsigned int for 32-bit atomic functions. *T* is type long or ulong for 64-bit atomic functions.

To use the base or extended atomic functions, include this pragma in your application:

```
#pragma OPENCL EXTENSION extension-name : enable
```

For base atomic functions, *extension-name* is one of:

```
cl_khr_global_int32_base_atomics  
cl_khr_local_int32_base_atomics  
cl_khr_int64_base_atomics
```

For extended atomic functions, *extension-name* is one of:

```
cl_khr_global_int32_extended_atomics  
cl_khr_local_int32_extended_atomics  
cl_khr_int64_extended_atomics
```

Base atomic functions

<i>T</i> atom_add (<i>Q T *p</i> , <i>T val</i>)	Read, add, and store
<i>T</i> atom_sub (<i>Q T *p</i> , <i>T val</i>)	Read, sub, and store
<i>T</i> atom_xchg (<i>Q T *p</i> , <i>T val</i>)	Read, swap, and store
<i>T</i> atom_inc (<i>Q T *p</i>)	Read, increment, and store
<i>T</i> atom_dec (<i>Q T *p</i>)	Read, decrement, and store
<i>T</i> atom_cmpxchg (<i>Q T *p</i> , <i>T cmp</i> , <i>T val</i>)	Read and store (* <i>p</i> == <i>cmp</i>) ? <i>val</i> : * <i>p</i>

Extended atomic functions

<i>T</i> atom_min (<i>Q T *p</i> , <i>T val</i>)	Read, store min(* <i>p</i> , <i>val</i>)
<i>T</i> atom_max (<i>Q T *p</i> , <i>T val</i>)	Read, store max(* <i>p</i> , <i>val</i>)
<i>T</i> atom_and (<i>Q T *p</i> , <i>T val</i>)	Read, store (* <i>p</i> & <i>val</i>)
<i>T</i> atom_or (<i>Q T *p</i> , <i>T val</i>)	Read, store (* <i>p</i> <i>val</i>)
<i>T</i> atom_xor (<i>Q T *p</i> , <i>T val</i>)	Read, store (* <i>p</i> ^ <i>val</i>)

Synchronization and Explicit Memory Fence Built-in Functions [6.11.9, 6.11.10]

The *flags* argument specifies the memory address space and can be set to a combination of CLK_LOCAL_MEM_FENCE and CLK_GLOBAL_MEM_FENCE.

void barrier (<i>cl_mem_fence_flags flags</i>)	All work-items in a work-group must execute this before any can continue
void mem_fence (<i>cl_mem_fence_flags flags</i>)	Orders loads and stores of a work-item executing a kernel
void read_mem_fence (<i>cl_mem_fence_flags flags</i>)	Orders memory loads
void write_mem_fence (<i>cl_mem_fence_flags flags</i>)	Orders memory stores

More built-in functions >



The Khronos Group is an industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics and dynamic media on a wide variety of platforms and devices. See www.khronos.org to learn more about the Khronos Group.

OpenCL is a trademark of Apple Inc. and is used under license by Khronos.

OpenCL™ API 1.0 Quick Reference Card: Graphics

Following is a quick reference to the subset of the OpenCL API specification that pertains to graphics. [n.n.n] refers to the section in the full specification, which is available at www.khronos.org/opencv.

Image Objects

Create Image Objects [5.2.4]

```
cl_mem clCreateImage2D (
    cl_context context, cl_mem_flags flags,
    const cl_image_format *image_format,
    size_t image_width, size_t image_height,
    size_t image_row_pitch, void *host_ptr,
    cl_int *errcode_ret)

flags: CL_MEM_READ_WRITE,      CL_MEM_WRITE_ONLY,
        CL_MEM_READ_ONLY,      CL_MEM_USE_HOST_PTR,
        CL_MEM_ALLOC_HOST_PTR, CL_MEM_COPY_HOST_PTR
```

```
cl_mem clCreateImage3D (
    cl_context context, cl_mem_flags flags,
    const cl_image_format *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_int *errcode_ret)

flags: CL_MEM_READ_WRITE,      CL_MEM_WRITE_ONLY,
        CL_MEM_READ_ONLY,      CL_MEM_USE_HOST_PTR,
        CL_MEM_ALLOC_HOST_PTR, CL_MEM_COPY_HOST_PTR
```

Query List of Supported Image Formats [5.25]

```
cl_int clGetSupportedImageFormats (
    cl_context context, cl_mem_flags flags,
    cl_mem_object_type image_type,
    cl_uint num_entries,
    cl_image_format *image_formats,
    cl_uint *num_image_formats)

flags: CL_MEM_READ_WRITE,      CL_MEM_WRITE_ONLY,
        CL_MEM_READ_ONLY,      CL_MEM_USE_HOST_PTR,
        CL_MEM_ALLOC_HOST_PTR, CL_MEM_COPY_HOST_PTR
```

Copy Between Image and Buffer Objects [5.2.7]

```
cl_int clEnqueueCopyImageToBuffer (
    cl_command_queue command_queue,
    cl_mem src_image, cl_mem dst_buffer,
    const size_t src_origin[3], const size_t region[3],
    size_t dst_offset, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

```
cl_int clEnqueueCopyBufferToImage (
    cl_command_queue command_queue,
    cl_mem src_buffer, cl_mem dst_image,
    size_t src_offset, const size_t dst_origin[3],
    const size_t region[3],
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

Map and Unmap Image Objects [5.2.8]

```
void * clEnqueueMapImage (
    cl_command_queue command_queue,
    cl_mem image, cl_bool blocking_map,
    cl_map_flags map_flags, const size_t origin[3],
    const size_t region[3], size_t *image_row_pitch,
    size_t *image_slice_pitch,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list,
    cl_event *event, cl_int *errcode_ret)
```

Read, Write, Copy Image Objects [5.2.6]

```
cl_int clEnqueueReadImage (
    cl_command_queue command_queue,
    cl_mem image, cl_bool blocking_read,
    const size_t origin[3], const size_t region[3],
    size_t row_pitch, size_t slice_pitch, void *ptr,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

```
cl_int clEnqueueWriteImage (
    cl_command_queue command_queue,
    cl_mem image, cl_bool blocking_write,
    const size_t origin[3], const size_t region[3],
    size_t input_row_pitch, size_t input_slice_pitch,
    const void *ptr, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

```
cl_int clEnqueueCopyImage (
    cl_command_queue command_queue,
    cl_mem src_image, cl_mem dst_image,
    const size_t src_origin[3], const size_t dst_origin[3],
    const size_t region[3],
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

Query Image Objects [5.2.9]

```
cl_int clGetMemObjectInfo (cl_mem memobj,
    cl_mem_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)

param_name: CL_MEM_TYPE,
             CL_MEM_FLAGS,
             CL_MEM_HOST_PTR,
             CL_MEM_REFERENCE_COUNT,
             CL_MEM_SIZE,
             CL_MEM_MAP_COUNT,
             CL_MEM_CONTEXT
```

```
cl_int clGetImageInfo (cl_mem image,
    cl_image_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)

param_name: CL_IMAGE_FORMAT,
             CL_IMAGE_ELEMENT_SIZE,
             CL_IMAGE_ROW_PITCH,
             CL_IMAGE_SLICE_PITCH,
             CL_IMAGE_HEIGHT,
             CL_IMAGE_WIDTH,
             CL_IMAGE_DEPTH
```

Sampler Objects [5.3]

```
cl_sampler clCreateSampler (cl_context context,
    cl_bool normalized_coords,
    cl_addressing_mode addressing_mode,
    cl_filter_mode filter_mode, cl_int *errcode_ret)
```

```
cl_int clRetainSampler (cl_sampler sampler)
```

```
cl_int clReleaseSampler (cl_sampler sampler)
```

```
cl_int clGetSamplerInfo (cl_sampler sampler,
    cl_sampler_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
```

```
param_value: CL_SAMPLER_REFERENCE_COUNT,
             CL_SAMPLER_CONTEXT,
             CL_SAMPLER_FILTER_MODE,
             CL_SAMPLER_ADDRESSING_MODE,
             CL_SAMPLER_NORMALIZED_COORDS
```

Sampler Declaration Fields [6.11.8.1]

The sampler can be passed as an argument to the kernel using `clSetKernelArg`, 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,
    CLK_NORMALIZED_COORDS_FALSE
```

```
filter-mode:
    CLK_FILTER_NEAREST, CLK_FILTER_LINEAR
```

```
address-mode:
    CLK_ADDRESS_REPEAT,
    CLK_ADDRESS_CLAMP_TO_EDGE,
    CLK_ADDRESS_CLAMP, CLK_ADDRESS_NONE
```

Image Access Qualifiers [6.6]

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

```
__read_only __write_only
```

Write to 3D Image Objects [9.8]

These functions write `color` at `coord` in image.

Include this pragma to write to 3D image memory objects using the functions shown in the table below:

```
#pragma OPENCL EXTENSION
    cl_khr_3d_image_writes : enable
```

```
void write_imagef (image3d_t image, int4 coord,
    float4 color)
```

```
void write_imagei (image3d_t image, int4 coord,
    int4 color)
```

```
void write_imageui (image3d_t image, int4 coord,
    unsigned int4 color)
```

Minimum list of supported image formats:

image_channel_order	image_channel_data_type
CL_RGBA	CL_UNORM_INT8, CL_UNORM_INT16, CL_SIGNED_INT8, CL_SIGNED_INT16, CL_SIGNED_INT32, CL_UNSIGNED_INT8, CL_UNSIGNED_INT16, CL_UNSIGNED_INT32, CL_HALF_FLOAT, CL_FLOAT
CL_BGRA	CL_UNORM_INT8

Image Read and Write Built-in Functions [6.11.8]

The built-in functions defined in this section can only be used with image memory objects created with `clCreateImage2D` or `clCreateImage3D`. **OPT** = Optional function.

<code>float4 read_imagef (image2d_t image, sampler_t sampler, int2 coord)</code> <code>float4 read_imagef (image2d_t image, sampler_t sampler, float2 coord)</code>		Read an element from a 2D image. <i>sampler</i> specifies the addressing and filtering mode to use.
<code>int4 read_imagei (image2d_t image, sampler_t sampler, int2 coord)</code> <code>int4 read_imagei (image2d_t image, sampler_t sampler, float2 coord)</code>		
<code>unsigned int4 read_imageui (image2d_t image, sampler_t sampler, int2 coord)</code> <code>unsigned int4 read_imageui (image2d_t image, sampler_t sampler, float2 coord)</code>		
<code>half4 read_imageh (image2d_t image, sampler_t sampler, int2 coord)</code> OPT <code>half4 read_imageh (image2d_t image, sampler_t sampler, float2 coord)</code> OPT	OPT OPT	
<code>void write_imagef (image2d_t image, int2 coord, float4 color)</code> <code>void write_imagei (image2d_t image, int2 coord, int4 color)</code> <code>void write_imageui (image2d_t image, int2 coord, unsigned int4 color)</code>		
<code>void write_imageh (image2d_t image, int2 coord, half4 color)</code> OPT	OPT	
<code>float4 read_imagef (image3d_t image, sampler_t sampler, int4 coord)</code> <code>float4 read_imagef (image3d_t image, sampler_t sampler, float4 coord)</code>		Read an element from a 3D image. <i>sampler</i> specifies the addressing and filtering mode to use.
<code>int4 read_imagei (image3d_t image, sampler_t sampler, int4 coord)</code> <code>int4 read_imagei (image3d_t image, sampler_t sampler, float4 coord)</code>		
<code>unsigned int4 read_imageui (image3d_t image, sampler_t sampler, int4 coord)</code> <code>unsigned int4 read_imageui (image3d_t image, sampler_t sampler, float4 coord)</code>		
<code>half4 read_imageh (image3d_t image, sampler_t sampler, int4 coord)</code> OPT <code>half4 read_imageh (image3d_t image, sampler_t sampler, float4 coord)</code> OPT	OPT OPT	
<code>int get_image_width (image2d_t image)</code> <code>int get_image_width (image3d_t image)</code>		
<code>int get_image_height (image2d_t image)</code> <code>int get_image_height (image3d_t image)</code>		2D or 3D image height in pixels
<code>int get_image_depth (image3d_t image)</code>		3D image depth in pixels
<code>int get_image_channel_data_type (image2d_t image)</code> <code>int get_image_channel_data_type (image3d_t image)</code>		image channel data type
<code>int get_image_channel_order (image2d_t image)</code> <code>int get_image_channel_order (image3d_t image)</code>		image channel order
<code>int2 get_image_dim (image2d_t image)</code>		2D image width and height
<code>int4 get_image_dim (image3d_t image)</code>		3D image width, height, and depth
<code>void write_imageh (image3d_t image, int4 coord, half4 color)</code> OPT	OPT	Writes <i>color</i> value to (<i>x</i> , <i>y</i> , <i>z</i>) location specified by <i>coord</i> in the 3D image.

OpenCL/OpenGL Sharing APIs [Appendix B]

Creating OpenCL memory objects from OpenGL objects using the functions `clCreateFromGLBuffer`, `clCreateFromGLTexture2D`, `clCreateFromGLTexture3D`, or `clCreateFromGLRenderbuffer` ensures that the underlying storage of that OpenGL object will not be deleted while the corresponding OpenCL memory object still exists.

CL Buffer Objects > GL Buffer Objects [B.1.1]

`cl_mem clCreateFromGLBuffer (cl_context context, cl_mem_flags flags, GLuint bufobj, int *errcode_ret)`

flags: CL_MEM_READ_ONLY,
CL_MEM_WRITE_ONLY,
CL_MEM_READ_WRITE

Query Information [B.1.4]

`cl_int clGetGLObjectInfo (cl_mem memobj, cl_gl_object_type *gl_object_type, GLuint *gl_object_name)`

gl_object_type: CL_GL_OBJECT_BUFFER,
CL_GL_OBJECT_TEXTURE2D,
CL_GL_OBJECT_TEXTURE_RECTANGLE,
CL_GL_OBJECT_TEXTURE3D,
CL_GL_OBJECT_RENDERBUFFER

CL Image Objects > GL Textures [B.1.2]

`cl_mem clCreateFromGLTexture2D (cl_context context, cl_mem_flags flags, GLenum target, GLint miplevel, GLuint texture, int *errcode_ret)`

flags: (Same as for `clCreateFromGLBuffer`)

target: GL_TEXTURE_2D,
GL_TEXTURE_RECTANGLE_ARB,
GL_TEXTURE_CUBE_MAP_POSITIVE_X,
GL_TEXTURE_CUBE_MAP_POSITIVE_Y,
GL_TEXTURE_CUBE_MAP_POSITIVE_Z,
GL_TEXTURE_CUBE_MAP_NEGATIVE_X,
GL_TEXTURE_CUBE_MAP_NEGATIVE_Y,
GL_TEXTURE_CUBE_MAP_NEGATIVE_Z

`cl_mem clCreateFromGLTexture3D (cl_context context, cl_mem_flags flags, GLenum target, GLint miplevel, GLuint texture, int *errcode_ret)`

flags: (Same as for `clCreateFromGLBuffer`)

target: GL_TEXTURE_3D

`cl_int clGetGLTextureInfo (cl_mem memobj, cl_gl_texture_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)`

param_name: CL_GL_TEXTURE_TARGET,
CL_GL_MIPMAP_LEVEL

Share Objects [B.1.5]

`cl_int clEnqueueAcquireGLObjects (cl_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_event *event)`

`cl_int clEnqueueReleaseGLObjects (cl_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_event *event)`

CL Image Objects > GL Renderbuffers [B.1.3]

`cl_mem clCreateFromGLRenderbuffer (cl_context context, cl_mem_flags flags, GLuint renderbuffer, int *errcode_ret)`

flags: (Same as for `clCreateFromGLBuffer`)