OpenCL™ API 1.0 Quick Reference Card

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/opencl.

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

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_properties: CL_CONTEXT_PLATFORM,
CL_GL_CONTEXT_KHR, CL_CGL_SHAREGROUP_KHR,
CL_EGL_DISPLAY_KHR, CL_GLX_DISPLAY_KHR,
CL_WGL_HDC_KHR

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_context_properties: (same as for cl_create_context)

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 **clGetPlatformIDs** (cl_uint *num_entries*,
cl_platform_id *platforms, cl_uint *num_platforms)

cl_int **clGetPlatformInfo** (cl_platform_id *platform*,
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_PLATFORM_NAME,
CL_PLATFORM_VENDOR, CL_PLATFORM_EXTENSIONS

cl_int **clGetDeviceIDs** (cl_platform_id *platform*,
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_PLATFORM, 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 | HEIGHT},
CL_DEVICE_IMAGE3D_MAX_{WIDTH | HEIGHT | 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

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 an 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

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

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)

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_name: CL_PROGRAM_REFERENCE_COUNT,
CL_PROGRAM_CONTEXT,
CL_PROGRAM_NUM_DEVICES,
CL_PROGRAM_DEVICES,
CL_PROGRAM_SOURCE,
CL_PROGRAM_BINARY_SIZES,
CL_PROGRAM_BINARIES,
CL_PROGRAM_SOURCE

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

OpenCL™ API 1.0 Quick Reference Card

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,
CL_KERNEL_LOCAL_MEM_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 (for storage only)
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-bt 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* OPT	64-bit float, vector
half*n* OPT	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*	*n***m* matrix of 32-bit floats
double*n*x*m*	*n***m* 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, ulong 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.: v.yx, v.xx, v.lo.x

Conversions and Type Casting Examples [6.2]

```
T a = (T)b;   // Scalar to scalar, or scalar to vector
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, global __local, local
__constant, constant __private, private

Function Qualifiers [6.7]

__kernel, 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, type*n*)	Same as: __kernel __attribute__((work_group_size_hint(X, 1, 1))) \ __attribute__((vec_type_hint(type*n*)))
__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.

uint **get_work_dim** ()	Num. of dimensions in use
size_t **get_global_size** (uint *D*)	Num. of global work-items
size_t **get_global_id** (uint *D*)	Global work-item ID value
size_t **get_local_size** (uint *D*)	Num. of local work-items
size_t **get_local_id** (uint *D*)	Local work-item ID
size_t **get_num_groups** (uint *D*)	Num. of work-groups
size_t **get_group_id** (uint *D*)	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.

Common Built-in Functions [6.11.4]

T is type float or float*n* (or optionally double, double*n*, or half*n*). **Optional extensions enable double, double*n*, and half*n* types.**

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

Integer Built-in Functions [6.11.3]

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n*. ***U*** refers to the unsigned version of *T*.

U **abs** (*T x*)	| *x* |
U **abs_diff** (*T x*, *T y*)	| *x* – *y* | without modulo overflow
T **add_sat** (*T x*, *T y*)	*x* + *y* and saturates the result
T **hadd** (*T x*, *T y*)	(*x* + *y*) >> 1 without modulo overflow
T **rhadd** (*T x*, *T y*)	(*x* + *y* + 1) >> 1
T **clz** (*T x*)	Number of leading 0-bits in *x*
T **mad_hi** (*T a*, *T b*, *T c*)	mul_hi(*a*, *b*) + *c*
T **mad24** (*T a*, *T b*, *T c*)	Multiply 24-bit integer values *a* and *b* and add the 32-bit integer result to 32-bit integer *c*
T **mad_sat** (*T a*, *T b*, *T c*)	*a* * *b* + *c* and saturates the result
T **max** (*T x*, *T y*)	*y* if *x* < *y*, otherwise it returns *x*
T **min** (*T x*, *T y*)	*y* if *y* < *x*, otherwise it returns *x*
T **mul_hi** (*T x*, *T y*)	high half of the product of *x* and *y*
T **mul24** (*T a*, *T b*)	Multiply 24-bit integer values *a* and *b*
T **rotate** (*T v*, *T i*)	result[*indx*] = *v*[*indx*] << *i*[*indx*]
T **sub_sat** (*T x*, *T y*)	*x* - *y* and saturates the result

*For **upsample**, scalar types are permitted for the vector types below.*

short*n* **upsample** (char*n hi*, uchar*n lo*)	result[*i*]= ((short)*hi*[*i*]<< 8) | *lo*[*i*]
ushort*n* **upsample** (uchar*n hi*, uchar*n lo*)	result[*i*]=((ushort)*hi*[*i*]<< 8) | *lo*[*i*]
int*n* **upsample** (short*n hi*, ushort*n lo*)	result[*i*]=((int)*hi*[*i*]<< 16) | *lo*[*i*]
uint*n* **upsample** (ushort*n hi*, ushort*n lo*)	result[*i*]=((uint)*hi*[*i*]<< 16) | *lo*[*i*]
long*n* **upsample** (int*n hi*, uint*n lo*)	result[*i*]=((long)*hi*[*i*]<< 32) | *lo*[*i*]
ulong*n* **upsample** (uint*n hi*, uint*n lo*)	result[*i*]=((ulong)*hi*[*i*]<< 32) | *lo*[*i*]

Math Built-in Functions [6.11.2]

T is type float or float*n* (or optionally double, double*n*, or half*n*). int*n*, uint*n*, and ulong*n* 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(). **Optional extensions enable double, double*n*, and half*n* types.**

T **acos** (*T*)		Arc cosine
T **acosh** (*T*)		Inverse hyperbolic cosine
T **acospi** (*T x*)		acos (*x*) / π
T **asin** (*T*)		Arc sine
T **asinh** (*T*)		Inverse hyperbolic sine
T **asinpi** (*T x*)		asin (*x*) / π
T **atan** (*T y_over_x*)		Arc tangent
T **atan2** (*T y*, *T x*)		Arc tangent of *y* / *x*
T **atanh** (*T*)		Hyperbolic arc tangent
T **atanpi** (*T x*)		atan (*x*) / π
T **atan2pi** (*T x*, *T y*)		atan2 (*x*, y) / π
T **cbrt** (*T*)		cube root
T **ceil** (*T*)		Round to integer toward + infinity
T **copysign** (*T x*, *T y*)		*x* with sign changed to sign of *y*
T **cos** (*T*)	HN	cosine
T **cosh** (*T*)		hyperbolic consine
T **cospi** (*T x*)		cos (π *x*)
T **half_divide** (*T x*, *T y*) *T* **native_divide** (*T x*, *T y*)		*x* / *y*
T **erfc** (*T*)		Complementary error function
T **erf** (*T*)		Calculates error function of *T*
T **exp** (*T x*)	HN	Exponential base e
T **exp2** (*T*)	HN	Exponential base 2
T **exp10** (*T*)	HN	Exponential base 10
T **expm1** (*T x*)		e^*x* -1.0
T **fabs** (*T*)		Absolute value
T **fdim** (*T x*, *T y*)		"Positive difference" between *x* and *y*
T **floor** (*T*)		Round to integer toward - infinity
T **fma** (*T a*, *T b*, *T c*)		Multiply and add, then round
T **fmax** (*T x*, *T y*) half*n* **fmax** (half*n x*, half)		Return *y* if *x* < *y*, otherwise it returns *x*
T **fmin** (*T x*, *T y*) half*n* **fmin** (half*n x*, half)		Return *y* if *y* < *x*, otherwise it returns *x*
T **fmod** (*T x*, *T y*)		Modulus. Returns *x* – *y* * trunc (*x*/*y*)
T **fract** (*T x*, *T* **iptr*)		Fractional value in *x*
T **frexp** (*T x*, int*n* **exp*)		Extract mantissa and exponent
T **hypot** (*T x*, *T y*)		square root of *x*^2+ *y*^2
int*n* **ilogb** (*T x*)		Return exponent as an integer value
T **ldexp** (*T x*, int*n n*) *T* **ldexp** (*T x*, int *n*)		*x* * 2^*n*
T **lgamma** (*T x*) *T* **lgamma_r** (*T x*, int*n* **signp*)		Log gamma function
T **log** (*T*)	HN	Natural logarithm
T **log2** (*T*)	HN	Base 2 logarithm
T **log10** (*T*)	HN	Base 10 logarithm
T **log1p** (*T x*)		ln (1.0 + *x*)
T **logb** (*T x*)		exponent of *x*
T **mad** (*T a*, *T b*, *T c*)		Approximates *a* * *b* + *c*
T **modf** (*T x*, *T* **iptr*)		Decompose a floating-point number
float **nan** (uint*n nancode*)		Quiet NaN
float*n* **nan** (uint*n nancode*)		
half*n* **nan** (ushort*n nancode*)		
double*n* **nan** (ulong*n nancode*)		
double*n* **nan** (uint*n nancode*)		
T **nextafter** (*T x*, *T y*)		Next representable floating-point value following *x* in the direction of *y*
T **pow** (*T x*, *T y*)		Compute *x* to the power of *y* (*x*^*y*)
T **pown** (*T x*, int*n y*)		Compute *x*^*y*, where *y* is an integer
T **powr** (*T x*, *T y*)	HN	Compute *x*^*y*, where *x* is >= 0
T **half_recip** (*T x*) *T* **native_recip** (*T x*)		1 / *x*
T **remainder** (*T x*, *T y*)		Floating point remainder function
T **remquo** (*T x*, *T y*, int*n* **quo*)		Floating point remainder and quotient function
T **rint** (*T*)		Round integer to nearest even integer
T **rootn** (*T x*, int*n y*)		Compute *x* to the power of 1/*y*
T **round** (*T x*)		Integral value nearest to *x* rounding
T **rsqrt** (*T*)	HN	Inverse square root
T **sin** (*T*)	HN	sine
T **sincos** (*T x*, *T* **cosval*)		sine and cosine of *x*
T **sinh** (*T*)		hyperbolic sine
T **sinpi** (*T x*)		sin (π *x*)
T **sqrt** (*T*)	HN	square root
T **tan** (*T*)	HN	tangent
T **tanh** (*T*)		hyperbolic tangent
T **tanpi** (*T x*)		tan (π *x*)
T **tgamma** (*T*)		gamma function
T **trunc** (*T*)		Round to integer toward zero

Geometric Built-in Functions [6.11.5]

Vector types may have 2 or 4 components. **Optional extensions enable double, double*n*, and half*n* types.**

float4 **cross** (float4 *p0*, float4 *p1*) double4 **cross** (double4 *p0*, double4 *p1*) half4 **cross** (half4 *p0*, half4 *p1*)	Cross product
float **dot** (float *p0*, float *p1*) float **dot** (float*n p0*, float*n p1*) double **dot** (double *p0*, double *p1*) double **dot** (double*n p0*, double*n p1*) half **dot** (half *p0*, half *p1*) half **dot** (half*n p0*, half*n p1*)	Dot product
float **distance** (float *p0*, float *p1*) float **distance** (float*n p0*, float*n p1*) double **distance** (double *p0*, double *p1*) double **distance** (double*n p0*, double*n p1*) half **distance** (half *p0*, half *p1*) half **distance** (half*n p0*, half*n p1*)	Vector distance
float **length** (float *p*) float **length** (float*n p*) double **length** (double *p*) double **length** (double*n p*) half **length** (half *p*) half **length** (half*n p*)	Vector length
float **normalize** (float *p*) float*n* **normalize** (float*n p*) double **normalize** (double *p*) double*n* **normalize** (double*n p*) half **normalize** (half *p*) half*n* **normalize** (half*n p*)	Normal vector length 1
float **fast_distance** (float *p0*, float *p1*) float **fast_distance** (float*n p0*, float*n p1*)	Vector distance
float **fast_length** (float *p*) float **fast_length** (float*n p*)	Vector length
float **fast_normalize** (float *p*) float*n* **fast_normalize** (float*n p*)	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 on the reverse >

OpenCL™ API 1.0 Quick Reference Card

Relational Built-in Functions [6.11.6]

T is type float, float*n*, char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n* (and optionally double, double*n*). ***S*** is type char, char*n*, short, short*n*, int, int*n*, long, or long*n*. ***U*** is type uchar, uchar*n*, ushort, ushort*n*, uint, uint*n*, ulong, or ulong*n*. **Optional extensions enable double, double*n*, and half*n* types.**

int **isequal** (float *x*, float *y*) int*n* **isequal** (float*n* *x*, float*n* *y*) int **isequal** (double *x*, double *y*) long*n* **isequal** (double*n* *x*, double*n* *y*) int **isequal** (half *x*, half *y*) short*n* **isequal** (half*n* *x*, half*n* *y*)	Compare of *x* == *y*
int **isnotequal** (float *x*, float *y*) int*n* **isnotequal** (float*n* *x*, float*n* *y*) int **isnotequal** (double *x*, double *y*) long*n* **isnotequal** (double*n* *x*, double*n* *y*) int **isnotequal** (half *x*, half *y*) short*n* **isnotequal** (half*n* *x*, half*n* *y*)	Compare of *x* != *y*
int **isgreater** (float *x*, float *y*) int*n* **isgreater** (float*n* *x*, float*n* *y*) int **isgreater** (double *x*, double *y*) long*n* **isgreater** (double*n* *x*, double*n* *y*) int **isgreater** (half *x*, half *y*) short*n* **isgreater** (half*n* *x*, half*n* *y*)	Compare of *x* > *y*
int **isgreaterequal** (float *x*, float *y*) int*n* **isgreaterequal** (float*n* *x*, float*n* *y*) int **isgreaterequal** (double *x*, double *y*) long*n* **isgreaterequal** (double*n* *x*, double*n* *y*) int **isgreaterequal** (half *x*, half *y*) short*n* **isgreaterequal** (half*n* *x*, half*n* *y*)	Compare of *x* >= *y*
int **isless** (float *x*, float *y*) int*n* **isless** (float*n* *x*, float*n* *y*) int **isless** (double *x*, double *y*) long*n* **isless** (double*n* *x*, double*n* *y*) int **isless** (half *x*, half *y*) short*n* **isless** (half*n* *x*, half*n* *y*)	Compare of *x* < *y*
int **islessequal** (float *x*, float *y*) int*n* **islessequal** (float*n* *x*, float*n* *y*) int **islessequal** (double *x*, double *y*) long*n* **islessequal** (double*n* *x*, double*n* *y*) int **islessequal** (half *x*, half *y*) short*n* **islessequal** (half*n* *x*, half*n* *y*)	Compare of *x* <= *y*
int **islessgreater** (float *x*, float *y*) int*n* **islessgreater** (float*n* *x*, float*n* *y*) int **islessgreater** (double *x*, double *y*) long*n* **islessgreater** (double*n* *x*, double*n* *y*) int **islessgreater** (half *x*, half *y*) short*n* **islessgreater** (half*n* *x*, half*n* *y*)	Compare of (*x* < *y*) || (*x* > *y*)
int **isfinite** (float) int*n* **isfinite** (float*n*) int **isfinite** (double) long*n* **isfinite** (double*n*) int **isfinite** (half) short*n* **isfinite** (half*n*)	Test for finite value
int **isinf** (float) int*n* **isinf** (float*n*) int **isinf** (double) long*n* **isinf** (double*n*) int **isinf** (half) short*n* **isinf** (half*n*)	Test for +ve or –ve infinity
int **isnan** (float) int*n* **isnan** (float*n*) int **isnan** (double) long*n* **isnan** (double*n*) int **isnan** (half) short*n* **isnan** (half*n*)	Test for a NaN
int **isnormal** (float) int*n* **isnormal** (float*n*) int **isnormal** (double) long*n* **isnormal** (double*n*) int **isnormal** (half) short*n* **isnormal** (half*n*)	Test for a normal value
int **isordered** (float *x*, float *y*) int*n* **isordered** (float*n* *x*, float*n* *y*) int **isordered** (double *x*, double *y*) long*n* **isordered** (double*n* *x*, double*n* *y*) int **isordered** (half *x*, half *y*) short*n* **isordered** (half*n* *x*, half*n* *y*)	Test if arguments are ordered
int **isunordered** (float *x*, float *y*) int*n* **isunordered** (float*n* *x*, float*n* *y*) int **isunordered** (double *x*, double *y*) long*n* **isunordered** (double*n* *x*, double*n* *y*) int **isunordered** (half *x*, half *y*) short*n* **isunordered** (half*n* *x*, half*n* *y*)	Test if arguments are unordered
int **signbit** (float) int*n* **signbit** (float*n*) int **signbit** (double) long*n* **signbit** (double*n*) int **signbit** (half) short*n* **signbit** (half*n*)	Test for sign bit
int **any** (*S x*)	1 if MSB in any component of *x* is set; else 0
int **all** (*S x*)	1 if MSB in all components of *x* are set; else 0
T **bitselect** (*T a*, *T b*, *T c*) half*n* **bitselect** (half*n a*, half*n b*, half*n c*) double*n* **bitselect** (double*n a*, double*n b*, double*n c*)	Each bit of result is corresponding bit of *a* if corresponding bit of *c* is 0
T **select** (*T a*, *T b*, *S c*) *T* **select** (*T a*, *T b*, *U c*) double*n* **select** (double*n*, double*n*, long*n*)	For each component of a vector type, result[*i*] = if MSB of *c*[*i*] is set ? *b*[*i*] : *a*[*i*] For scalar type, result = *c* ? *b* : *a*

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

Extended atomic functions	
T **atom_min** (*Q T* *p, *T val*)	Read, store min(**p*, *val*)
T **atom_max** (*Q T* *p, *T val*)	Read, store max(**p*, *val*)
T **atom_and** (*Q T* *p, *T val*)	Read, store (**p* & *val*)
T **atom_or** (*Q T* *p, *T val*)	Read, store (**p* | *val*)
T **atom_xor** (*Q T* *p, *T val*)	Read, store (**p* ^ *val*)

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 (or optionally double). ***Tn*** refers to the vector form of type *T*. **Optional extension enables the double and double*n* types.**

Tn **vload*n*** (size_t *offset*, const *Q T* *p)	Read vector data from memory
void **vstore*n*** (*Tn data*, size_t *offset*, *Q T* *p)	Write vector data to memory *(Q in this function cannot be __constant)*
float **vload_half** (size_t *offset*, const *Q* half *p)	Read a half from memory
float*n* **vload_half*n*** (size_t *offset*, const *Q* half *p)	Read multiple halfs from memory
void **vstore_half** (float *data*, size_t *offset*, *Q* half *p) void **vstore_half_***R* (float *data*, size_t *offset*, *Q* half *p) void **vstore_half** (double *data*, size_t *offset*, *Q* half *p) void **vstore_half_***R* (double *data*, size_t *offset*, *Q* half *p)	Write a half to memory *(Q in this function cannot be __constant)*
void **vstore_half*n*** (float*n* *data*, size_t *offset*, *Q* half *p) void **vstore_half*n*_***R*** (float*n* *data*, size_t *offset*, *Q* half *p) void **vstore_half*n*** (double*n* *data*, size_t *offset*, *Q* half *p) void **vstore_half*n*_***R*** (double*n* *data*, size_t *offset*, *Q* half *p)	Write a half vector to memory *(Q in this function cannot be __constant)*
float*n* **vloada_half*n*** (size_t *offset*, const *Q* half *p)	sizeof (float*n*) bytes of data read from location (*p* + (*offset* * *n*))
void **vstorea_half*n*** (float*n* *data*, size_t *offset*, *Q* half *p) void **vstorea_half*n*_***R*** (float*n* *data*, size_t *offset*, *Q* half *p) void **vstorea_half*n*** (double*n* *data*, size_t *offset*, *Q* half *p) void **vstorea_half*n*_***R*** (double*n* *data*, size_t *offset*, *Q* half *p)	Write a half vector to vector-aligned memory *(Q in this function cannot be __constant)*

Async Copies and Prefetch Built-in Functions [6.11.11]

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, ulong*n*, float, float*n*, and optionally double, double*n*. **Optional extension enables the double and double*n* types.**

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

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

More built-in functions on the reverse >

OpenCL

KHRONOS GROUP

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.

Reference card production by Miller & Mattson www.millermattson.com

See www.khronos.org/opencl for the full specification.

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/opencl.

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 [5.2.5]

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_SIZE, CL_MEM_HOST_PTR, CL_MEM_MAP_COUNT, CL_MEM_REFERENCE_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, read_only
__write_only, write_only

Image Formats [5.2.4.1, 9.8]

Supported image formats: image_channel_order with image_channel_data_type.

Built-in support:

CL_RGBA: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8|16}, CL_SIGNED_INT{8|16|32}, CL_UNSIGNED_INT{8|16|32}
CL_BGRA: CL_UNORM_INT8

Optional support:

CL_R, CL_A: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8|16}, CL_SIGNED_INT{8|16|32}, CL_UNSIGNED_INT{8|16|32}, CL_SNORM_INT{8|16}
CL_INTENSITY: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8|16}, CL_SNORM_INT{8|16}
CL_LUMINANCE: CL_UNORM_INT{8|16}, CL_HALF_FLOAT, CL_FLOAT, CL_SNORM_INT{8|16}
CL_RG, CL_RA: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8|16}, CL_SIGNED_INT{8|16|32}, CL_UNSIGNED_INT{8|16|32}, CL_SNORM_INT{8|16}
CL_RGB: CL_UNORM_SHORT_{555|565}, CL_UNORM_INT_101010
CL_ARGB: CL_UNORM_INT8, CL_SIGNED_INT8, CL_UNSIGNED_INT8, CL_SNORM_INT8
CL_BGRA: CL_SIGNED_INT8, CL_UNSIGNED_INT8, CL_SNORM_INT8

OpenCL™ API 1.0 Quick Reference Card: Graphics

Image Read and Write Built-in Functions [6.11.8, 9.8]

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

float4 **read_imagef** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) float4 **read_imagef** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) int4 **read_imagei** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) int4 **read_imagei** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) unsigned int4 **read_imageui** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) unsigned int4 **read_imageui** (image2d_t *image*, sampler_t *sampler*, float2 *coord*)	Read an element from a 2D image. *sampler* specifies the addressing and filtering mode to use.
half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) OPT half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) OPT	
void **write_imagef** (image2d_t *image*, int2 *coord*, float4 *color*) void **write_imagei** (image2d_t *image*, int2 *coord*, int4 *color*) void **write_imageui** (image2d_t *image*, int2 *coord*, unsigned int4 *color*)	Write *color* value to (*x*, *y*) location specified by *coord* in the 2D image
void **write_imageh** (image2d_t *image*, int2 *coord*, half4 *color*) OPT	
float4 **read_imagef** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) float4 **read_imagef** (image3d_t *image*, sampler_t *sampler*, float4 *coord*) int4 **read_imagei** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) int4 **read_imagei** (image3d_t *image*, sampler_t *sampler*, float4 *coord*) unsigned int4 **read_imageui** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) unsigned int4 **read_imageui** (image3d_t *image*, sampler_t *sampler*, float4 *coord*)	Read an element from a 3D image. *sampler* specifies the addressing and filtering mode to use.
half4 **read_imageh** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) OPT half4 **read_imageh** (image3d_t *image*, sampler_t *sampler*, float4 *coord*) OPT	
int **get_image_width** (image2d_t *image*) int **get_image_width** (image3d_t *image*)	2D or 3D image width in pixels
int **get_image_height** (image2d_t *image*) int **get_image_height** (image3d_t *image*)	2D or 3D image height in pixels
int **get_image_depth** (image3d_t *image*)	3D image depth in pixels
int **get_image_channel_data_type** (image2d_t *image*) int **get_image_channel_data_type** (image3d_t *image*)	image channel data type
int **get_image_channel_order** (image2d_t *image*) int **get_image_channel_order** (image3d_t *image*)	image channel order
int2 **get_image_dim** (image2d_t *image*)	2D image width and height
int4 **get_image_dim** (image3d_t *image*)	3D image width, height, and depth
void **write_imageh** (image3d_t *image*, int4 *coord*, half4 *color*) OPT	Writes *color* value to (*x*, *y*, *z*) location specified by *coord* in the 3D image.
void **write_imagef** (image3d_t *image*, int4 *coord*, float4 *color*) OPT void **write_imagei** (image3d_t *image*, int4 *coord*, int4 *color*) OPT void **write_imageui** (image3d_t *image*, int4 *coord*, unsigned int4 *color*) OPT	Writes *color* at *coord* in the 3D image. Include this pragma to enable these functions: #pragma OPENCL EXTENSION cl_khr_3d_image_writes : enable

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. (Items shown in red are optional)

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

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, GL_TEXTURE_CUBE_MAP_POSITIVE_{X | Y | Z}, GL_TEXTURE_CUBE_MAP_NEGATIVE_{X | Y | 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 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)

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_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*)

Querying CL Devices in GL Context [9.11]

cl_int **clGetGLContextInfoKHR** (
const cl_context_properties *properties*,
cl_gl_context_info *param_name*,
size_t *param_value_size*, void *param_value*,
size_t *param_value_size_ret)

param_name: CL_DEVICES_FOR_GL_CONTEXT_KHR, CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR