

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/opencv.

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

Buffer Objects

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

Create Buffer Objects [5.2.1]

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

`cl_mem` **clCreateSubBuffer** (`cl_mem buffer`, `cl_mem_flags flags`, `cl_buffer_create_type buffer_create_type`, `const void *buffer_create_info`, `cl_int *errcode_ret`)

flags for `clCreateBuffer` and `clCreateSubBuffer`:
CL_MEM_READ_WRITE,
CL_MEM_{WRITE, READ}_ONLY,
CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

Read, Write, Copy Buffer Objects [5.2.2]

`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`)

Program Objects

Create Program Objects [5.6.1]

`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`)

The OpenCL Platform Layer

The OpenCL platform layer 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** (`const cl_context_properties *properties`, `cl_uint num_devices`, `const cl_device_id *devices`, `void (CL_CALLBACK *pfn_notify)` (`const char *errinfo`, `const void *private_info`, `size_t cb`, `void *user_data`), `void *user_data`, `cl_int *errcode_ret`)

properties: CL_CONTEXT_PLATFORM, CL_GL_CONTEXT_KHR, CL_CGL_SHAREGROUP_KHR, CL_{EGL, GLX}_DISPLAY_KHR, CL_WGL_HDC_KHR

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

properties: See `clCreateContext`

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

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

`cl_int` **clGetContextInfo** (`cl_context context`, `cl_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, CL_CONTEXT_NUM_DEVICES

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_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_{NATIVE, PREFERRED}_VECTOR_WIDTH_CHAR, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_SHORT, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_INT, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_LONG, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_FLOAT, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_DOUBLE, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_HALF, CL_DEVICE_MAX_CLOCK_FREQUENCY, CL_DEVICE_ADDRESS_BITS, CL_DEVICE_MAX_MEM_ALLOC_SIZE, CL_DEVICE_IMAGE_SUPPORT, CL_DEVICE_MAX_{READ, 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, SIZE}, CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE, CL_DEVICE_GLOBAL_MEM_SIZE, CL_DEVICE_MAX_CONSTANT_{BUFFER_SIZE, ARGS}, CL_DEVICE_LOCAL_MEM_{TYPE, 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, VENDOR, PROFILE, EXTENSIONS}, CL_DEVICE_HOST_UNIFIED_MEMORY, CL_DEVICE_OPENCL_C_VERSION, CL_DEVICE_VERSION, CL_DRIVER_VERSION, CL_DEVICE_PLATFORM

Map Buffer Objects [5.2.2]

`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`)

Map Buffer Objects [5.4.1-2]

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

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

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

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

Query Buffer Object [5.4.3]

`cl_int` **clGetMemObjectInfo** (`cl_mem memobj`, `cl_mem_info 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_{MAP, REFERENCE}_COUNT, CL_MEM_OFFSET, CL_MEM_CONTEXT, CL_MEM_ASSOCIATED_MEMOBJECT

Math Intrinsics:

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

Warning request/suppress:

-w -Werror

Control OpenCL C language version:

-cl-std=CL1.1 // OpenCL 1.1 specification.

Query Program Objects [5.6.5]

`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, NUM_DEVICES, DEVICES}, CL_PROGRAM_{SOURCE, BINARY_SIZES, BINARIES}

(Program Objects Continue >)

Program Objects (continued)

```
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, OPTIONS, LOG
```

Unload the OpenCL Compiler [5.6.4]

```
cl_int clUnloadCompiler (void)
```

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	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	OPTIONAL 64-bit float, vector
half _n	16-bit, 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 _n	128-bit unsigned

Kernel and Event Objects

Create Kernel Objects [5.7.1]

```
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 Args. & Object Queries [5.7.2, 5.7.3]

```
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, PRIVATE}_MEM_SIZE, CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_MULTIPLE
```

Execute Kernels [5.8]

```
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 [5.9]

```
cl_event clCreateUserEvent (cl_context context, cl_int *errcode_ret)
cl_int clSetUserEventStatus (cl_event event, cl_int execution_status)
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)
param_name: CL_EVENT_COMMAND_QUEUE, TYPE, CL_EVENT_{CONTEXT, REFERENCE_COUNT}, CL_EVENT_COMMAND_EXECUTION_STATUS
cl_int clSetEventCallback (cl_event event, cl_int command_exec_callback_type,
                           void (CL_CALLBACK *pfn_event_notify)(cl_event event, cl_int event_command_exec_status,
                                                                    void *user_data),
                           void *user_data)
cl_int clRetainEvent (cl_event event)
cl_int clReleaseEvent (cl_event event)
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)
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, START, END}
```

Out-of-order Execution of Kernels & Memory Object Commands [5.10]

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

Profiling Operations [5.11]

```
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, START, END}
```

Flush and Finish [5.12]

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

Vector Component Addressing [6.1.7]

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														
float3 _v ;	v.x, v.s0	v.y, v.s1	v.z, v.s2													
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

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

	v.lo	v.hi	v.odd	v.even		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	float8	v.s0123	v.s4567	v.s1357	v.s0246
float3*	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz	float16	v.s01234567	v.s89abcdef	v.s13579bdf	v.s02468ace
float4	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz					

*When using .lo or .hi with a 3-component vector, the .w component is undefined.

Conversions & 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 = as_T(b);
```

T a = convert_T_sat_R(b); //R is rounding mode

R can be one of the following rounding modes:
 _rte to nearest even _rtp toward + infinity
 _rtz toward zero _rtn toward - 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(type))) //type defaults to 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

__FILE__	Current source file
__LINE__	Integer line number
__OPENCL_VERSION__	Integer version number
__CL_VERSION_1_0__	Substitutes integer 100 for version 1.0
__CL_VERSION_1_1__	Substitutes integer 110 for version 1.1
__ENDIAN_LITTLE__	1 if device is little endian
__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

Specify Type Attributes [6.10.1]

Use to specify special attributes of enum, struct and union types.

__attribute__((aligned(<i>n</i>)))	__attribute__((endian(host)))
__attribute__((aligned))	__attribute__((endian(device)))
__attribute__((packed))	__attribute__((endian))

Math Constants [6.11.2]

The values of the following symbolic constants are type float and are accurate within the precision of a single precision floating-point number.

MAXFLOAT	Value of max. non-infinite single-precision floating-point number.
HUGE_VALF	Positive float expression, evaluates to +infinity. Used as error value.

HUGE_VAL	Positive double expression, evals. to +infinity. Used as error value. OPTIONAL	M_LN2_F	Value of loge2
INFINITY	Constant float expression, positive or unsigned infinity.	M_LN10_F	Value of loge10
NAN	Constant float expression, quiet NaN.	M_PI_F	Value of π
M_E_F	Value of e	M_PI_2_F	Value of π / 2
M_LOG2E_F	Value of log2e	M_PI_4_F	Value of π / 4
M_LOG10E_F	Value of log10e	M_1_PI_F	Value of 1 / π
		M_2_PI_F	Value of 2 / π
		M_2_SQRTPI_F	Value of 2 / √π
		M_SQRT2_F	Value of √2
		M_SQRT1_2_F	Value of 1 / √2

Work-Item Built-in Functions [6.11.1]

D is dimension index.

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

Common Built-in Functions [6.11.4]

T is type float or floatn (or optionally double, doublen, or halfn).
 Optional extensions enable double, doublen, and halfn types.

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

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, or ulongn.
U is the unsigned version of *T*. *S* is the scalar version of *T*.

<i>U</i> abs (<i>T</i> <i>x</i>)	<i>x</i>
<i>U</i> abs_diff (<i>T</i> <i>x</i> , <i>T</i> <i>y</i>)	<i>x</i> - <i>y</i> without modulo overflow
<i>T</i> add_sat (<i>T</i> <i>x</i> , <i>T</i> <i>y</i>)	<i>x</i> + <i>y</i> and saturates the result
<i>T</i> hadd (<i>T</i> <i>x</i> , <i>T</i> <i>y</i>)	(<i>x</i> + <i>y</i>) >> 1 without mod. overflow
<i>T</i> rhadd (<i>T</i> <i>x</i> , <i>T</i> <i>y</i>)	(<i>x</i> + <i>y</i> + 1) >> 1
<i>T</i> clz (<i>T</i> <i>x</i>)	Number of leading 0-bits in <i>x</i>
<i>T</i> clamp (<i>T</i> <i>x</i> , <i>T</i> <i>min</i> , <i>T</i> <i>max</i>) <i>T</i> clamp (<i>T</i> <i>x</i> , <i>S</i> <i>min</i> , <i>S</i> <i>max</i>)	min(max(<i>x</i> , <i>minval</i>), <i>maxval</i>)
<i>T</i> mad_hi (<i>T</i> <i>a</i> , <i>T</i> <i>b</i> , <i>T</i> <i>c</i>)	mul_hi(<i>a</i> , <i>b</i>) + <i>c</i>
<i>T</i> mad_sat (<i>T</i> <i>a</i> , <i>T</i> <i>b</i> , <i>T</i> <i>c</i>)	<i>a</i> * <i>b</i> + <i>c</i> and saturates the result
<i>T</i> max (<i>T</i> <i>x</i> , <i>T</i> <i>y</i>) <i>T</i> max (<i>T</i> <i>x</i> , <i>S</i> <i>y</i>)	<i>y</i> if <i>x</i> < <i>y</i> , otherwise it returns <i>x</i>
<i>T</i> min (<i>T</i> <i>x</i> , <i>T</i> <i>y</i>) <i>T</i> min (<i>T</i> <i>x</i> , <i>S</i> <i>y</i>)	<i>y</i> if <i>y</i> < <i>x</i> , otherwise it returns <i>x</i>
<i>T</i> mul_hi (<i>T</i> <i>x</i> , <i>T</i> <i>y</i>)	high half of the product of <i>x</i> and <i>y</i>
<i>T</i> rotate (<i>T</i> <i>v</i> , <i>T</i> <i>i</i>)	result[<i>indx</i>] = <i>v</i> [<i>indx</i>] << <i>i</i> [<i>indx</i>]

T sub_sat (*T* *x*, *T* *y*) *x* - *y* and saturates the result

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

shortn upsample (charn <i>hi</i> , ucharn <i>lo</i>)	result[<i>i</i>] = ((short)hi[<i>i</i>] << 8) lo[<i>i</i>]
ushortn upsample (ucharn <i>hi</i> , ucharn <i>lo</i>)	result[<i>i</i>] = ((ushort)hi[<i>i</i>] << 8) lo[<i>i</i>]
intrn upsample (shortn <i>hi</i> , ushortn <i>lo</i>)	result[<i>i</i>] = ((int)hi[<i>i</i>] << 16) lo[<i>i</i>]
uintn upsample (ushortn <i>hi</i> , ushortn <i>lo</i>)	result[<i>i</i>] = ((uint)hi[<i>i</i>] << 16) lo[<i>i</i>]
longn upsample (intrn <i>hi</i> , uintn <i>lo</i>)	result[<i>i</i>] = ((long)hi[<i>i</i>] << 32) lo[<i>i</i>]
ulongn upsample (uintn <i>hi</i> , uintn <i>lo</i>)	result[<i>i</i>] = ((ulong)hi[<i>i</i>] << 32) lo[<i>i</i>]

The following fast integer functions optimize the performance of kernels. In these functions, *T* is type int, int2, int3, int4, int8, int16, uint, uint2, uint4, uint8 or uint16.

<i>T</i> mad24 (<i>T</i> <i>a</i> , <i>T</i> <i>b</i> , <i>T</i> <i>c</i>)	Multiply 24-bit int. values <i>a</i> , <i>b</i> , add 32-bit int. result to 32-bit int. <i>c</i>
<i>T</i> mul24 (<i>T</i> <i>a</i> , <i>T</i> <i>b</i>)	Multiply 24-bit int. values <i>a</i> and <i>b</i>

Math Built-in Functions [6.11.2]

T is type float or floatn (or optionally double, doublen, or halfn). intrn, uintn, and ulongn must be scalar when *T* is scalar. *Q* is qualifier __global__, __local__, or __private. **HN** indicates that Half and Native variants are available by prepending "half_" or "native_" to function name. Prototypes shown in purple are half_ and native_ only. Optional extensions enable double, doublen, half, and halfn types.

<i>T</i> acos (<i>T</i>)	Arc cosine
<i>T</i> acosh (<i>T</i>)	Inverse hyperbolic cosine
<i>T</i> acospi (<i>T</i> <i>x</i>)	acos (<i>x</i>) / π
<i>T</i> asin (<i>T</i>)	Arc sine
<i>T</i> asinh (<i>T</i>)	Inverse hyperbolic sine
<i>T</i> asinpi (<i>T</i> <i>x</i>)	asin (<i>x</i>) / π
<i>T</i> atan (<i>T</i> <i>y</i> _over_ <i>x</i>)	Arc tangent
<i>T</i> atan2 (<i>T</i> <i>y</i> , <i>T</i> <i>x</i>)	Arc tangent of <i>y</i> / <i>x</i>
<i>T</i> atanh (<i>T</i>)	Hyperbolic arc tangent
<i>T</i> atanpi (<i>T</i> <i>x</i>)	atan (<i>x</i>) / π
<i>T</i> atan2pi (<i>T</i> <i>x</i> , <i>T</i> <i>y</i>)	atan2 (<i>x</i> , <i>y</i>) / π
<i>T</i> cbqrt (<i>T</i>)	Cube root
<i>T</i> ceil (<i>T</i>)	Round to integer toward + infinity
<i>T</i> copysign (<i>T</i> <i>x</i> , <i>T</i> <i>y</i>)	<i>x</i> with sign changed to sign of <i>y</i>
<i>T</i> cos (<i>T</i>) HN	Cosine
<i>T</i> cosh (<i>T</i>)	Hyperbolic cosine
<i>T</i> cospi (<i>T</i> <i>x</i>)	cos (π <i>x</i>)
<i>T</i> half_divide (<i>T</i> <i>x</i> , <i>T</i> <i>y</i>)	<i>x</i> / <i>y</i>
<i>T</i> native_divide (<i>T</i> <i>x</i> , <i>T</i> <i>y</i>)	(<i>T</i> may be float or floatn)
<i>T</i> erfc (<i>T</i>)	Complementary error function
<i>T</i> erf (<i>T</i>)	Calculates error function of <i>T</i>
<i>T</i> exp (<i>T</i> <i>x</i>) HN	Exponential base e
<i>T</i> exp2 (<i>T</i>) HN	Exponential base 2
<i>T</i> exp10 (<i>T</i>) HN	Exponential base 10

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

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

Geometric Built-in Functions [6.11.5]

Vector types may have 2, 3, or 4 components. Optional extensions enable `double`, `doublen`, and `halfn` types.

float dot (float p0, float p1) float dot (floatn p0, floatn p1) double dot (double p0, double p1) double dot (doublen p0, doublen p1) half dot (half p0, half p1) half dot (halfn p0, halfn p1)	Dot product
float{3,4} cross (float{3,4} p0, float{3,4} p1) double{3,4} cross (double{3,4} p0, double{3,4} p1) half{3,4} cross (half{3,4} p0, half{3,4} p1)	Cross product

float distance (float p0, float p1) float distance (floatn p0, floatn p1) double distance (double p0, double p1) double distance (doublen p0, doublen p1) half distance (half p0, half p1) half distance (halfn p0, halfn p1)	Vector distance
float length (float p) float length (floatn p) double length (double p) double length (doublen p) half length (half p) half length (halfn p)	Vector length

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

Relational Built-in Functions [6.11.6]

T is type float, floatn, char, charn, uchar, uchar, short, shortn, ushort, ushortn, int, intrn, uint, uintn, long, longn, ulong, or ulongn (and optionally double, doublen). *S* is type char, charn, short, shortn, int, intrn, long, or longn. *U* is type uchar, uchar, ushort, ushortn, uint, uintn, ulong, or ulongn. Optional extensions enable `double`, `doublen`, and `halfn` types.

int izequal (float x, float y) intrn izequal (floatn x, floatn y) int izequal (double x, double y) longn izequal (doublen x, doublen y) int izequal (half x, half y) shortn izequal (halfn x, halfn y)	Compare of x == y
int isnotequal (float x, float y) intrn isnotequal (floatn x, floatn y) int isnotequal (double x, double y) longn isnotequal (doublen x, doublen y) int isnotequal (half x, half y) shortn isnotequal (halfn x, halfn y)	Compare of x != y
int isgreater (float x, float y) intrn isgreater (floatn x, floatn y) int isgreater (double x, double y) longn isgreater (doublen x, doublen y) int isgreater (half x, half y) shortn isgreater (halfn x, halfn y)	Compare of x > y
int isgreaterequal (float x, float y) intrn isgreaterequal (floatn x, floatn y) int isgreaterequal (double x, double y) longn isgreaterequal (doublen x, doublen y) int isgreaterequal (half x, half y) shortn isgreaterequal (halfn x, halfn y)	Compare of x >= y
int isless (float x, float y) intrn isless (floatn x, floatn y) int isless (double x, double y) longn isless (doublen x, doublen y) int isless (half x, half y) shortn isless (halfn x, halfn y)	Compare of x < y
int islessequal (float x, float y) intrn islessequal (floatn x, floatn y) int islessequal (double x, double y) longn islessequal (doublen x, doublen y) int islessequal (half x, half y) shortn islessequal (halfn x, halfn y)	Compare of x <= y
int islessgreater (float x, float y) intrn islessgreater (floatn x, floatn y) int islessgreater (double x, double y) longn islessgreater (doublen x, doublen y) int islessgreater (half x, half y) shortn islessgreater (halfn x, halfn y)	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 x, float y) intrn isordered (floatn x, floatn y) int isordered (double x, double y) longn isordered (doublen x, doublen y) int isordered (half x, half y) shortn isordered (halfn x, halfn y)	Test if arguments are ordered
int isunordered (float x, float y) intrn isunordered (floatn x, floatn y) int isunordered (double x, double y) longn isunordered (doublen x, doublen y) int isunordered (half x, half y) shortn isunordered (halfn x, halfn y)	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 (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 bselect (T a, T b, T c) halfn bselect (halfn a, halfn b, halfn c) doublen bselect (doublen a, doublen b, doublen 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) doublen select (doublen, doublen, longn) doublen select (doublen, doublen, ulongn) halfn select (halfn, halfn, shortn) halfn select (halfn, halfn, ushortn)	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

Vector Data Load/Store 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 extensions enable the `double`, `doublen`, `half`, and `halfn` types.

Tn vloadn (size_t offset, const Q T *p)	Read vector data from memory
void vstoren (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
floatn vload_halfn (size_t offset, const Q half *p)	Read multiple halves 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_halfn (floatn data, size_t offset, Q half *p) void vstore_halfn_R (floatn data, size_t offset, Q half *p) void vstore_halfn (doublen data, size_t offset, Q half *p) void vstore_halfn_R (doublen data, size_t offset, Q half *p)	Write a half vector to memory (Q in this function cannot be __constant)
floatn vloada_halfn (size_t offset, const Q half *p)	sizeof (floatn) bytes of data read from location (p + (offset * n))
void vstorea_halfn (floatn data, size_t offset, Q half *p) void vstorea_halfn_R (floatn data, size_t offset, Q half *p) void vstorea_halfn (doublen data, size_t offset, Q half *p) void vstorea_halfn_R (doublen 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 Functions [6.11.10]

T is type char, charn, uchar, uchar, short, shortn, ushort, ushortn, int, intrn, uint, uintn, long, longn, ulong, ulongn, float, floatn, and optionally halfn double, doublen. Optional extensions enable the `halfn`, `double`, and `doublen` types.

event_t async_work_group_copy (_local T *dst, const _global T *src, size_t num_gentypes, event_t event)	Copies num_gentypes T elements from src to dst
event_t async_work_group_copy (_global T *dst, const _local T *src, size_t num_gentypes, event_t event)	Copies num_gentypes T elements from src to dst
event_t async_work_group_strided_copy (_local T *dst, const _global T *src, size_t num_gentypes, size_t src_stride, event_t event)	Copies num_gentypes T elements from src to dst
event_t async_work_group_strided_copy (_global T *dst, const _local T *src, size_t num_gentypes, size_t dst_stride, event_t event)	Copies num_gentypes 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_gentypes)	Prefetch num_gentypes * sizeof(T) bytes into the global cache

Atomic Functions [6.11.11, 9.4]

T is type int or unsigned int. *T* may also be type float for atomic_xchg, and type long or ulong for extended 64-bit atomic functions. *Q* is volatile __global or volatile __local, except *Q* must be volatile __global for atomic_xchg when *T* is float.

The built-in atomic functions for 32-bit values begin with atomic_ while the extended 64-bit atomic functions begin with atom_ . For example:

Built-in atomic function atomic_add ()	Extended atomic function atom_add ()
---	---

Extended 64-bit atomic functions are enabled by the following pragma; extension-name is one of cl_khr_int64_{base, extended}_atomics:

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

T atomic_add (Q T *p, T val)	Read, add, and store
T atomic_sub (Q T *p, T val)	Read, subtract, and store
T atomic_xchg (Q T *p, T val)	Read, swap, and store
T atomic_inc (Q T *p)	Read, increment, and store
T atomic_dec (Q T *p)	Read, decrement, and store
T atomic_cmpxchg (Q T *p, T cmp, T val)	Read and store (*p == cmp) ? val : *p
T atomic_min (Q T *p, T val)	Read, store min(*p, val)
T atomic_max (Q T *p, T val)	Read, store max(*p, val)
T atomic_and (Q T *p, T val)	Read, store (*p & val)
T atomic_or (Q T *p, T val)	Read, store (*p val)
T atomic_xor (Q T *p, T val)	Read, store (*p ^ val)

Miscellaneous Vector Built-In Functions [6.11.12]

Tn and *Tm* mean the 2,4,8, or 16-component vectors of char, uchar, short, ushort, half, int, uint, long, ulong, float, double. *Un* means the built-in unsigned integer data types. For `vec_step()`, *Tn* also includes `char3`, `uchar3`, `short3`, `ushort3`, `half3`, `int3`, `uint3`, `long3`, `ulong3`, `float3`, and `double3`. Half and double types are enabled by `cl_khr_fp16` and `cl_khr_fp64` respectively.

<code>int vec_step (Tn a)</code> <code>int vec_step (typename)</code>	Takes a built-in scalar or vector data type argument and returns an integer value representing the number of elements in the scalar or vector.	<code>Tn shuffle (Tm x, Un mask)</code> <code>Tn shuffle2 (Tm x, Tm y, Un mask)</code>	Construct permutation of elements from one or two input vectors, return a vector with same element type as input & length that is the same as the shuffle mask.
--	--	---	---

Synchronization, Explicit Mem. Fence [6.11.9-10]

flags argument is the memory address space, set to a combination of `CLK_LOCAL_MEM_FENCE` and `CLK_GLOBAL_MEM_FENCE`.

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

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

Image Read and Write Built-in Functions [6.11.13, 9.5, 9.6.8]

The built-in functions defined in this section can only be used with image memory objects created with `clCreateImage2D` or `clCreateImage3D`. *sampler* specifies the addressing and filtering mode to use. **H** = To enable `read_imageh` and `write_imageh`, enable extension `cl_khr_fp16`. **3D** = To enable type `image3d_t` in `write_imagef(f, i, ui)`, enable extension `cl_khr_3d_image_writes`.

<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	<code>uint4 read_imageui (image3d_t image, sampler_t sampler, int4 coord)</code> <code>uint4 read_imageui (image2d_t image, sampler_t sampler, float2 coord)</code>	Read an element from a 3D image
<code>half4 read_imageh (image2d_t image, sampler_t sampler, int2 coord)</code> H <code>half4 read_imageh (image2d_t image, sampler_t sampler, float2 coord)</code> H		<code>uint4 read_imagei (image2d_t image, sampler_t sampler, int2 coord)</code> <code>uint4 read_imagei (image2d_t image, sampler_t sampler, float2 coord)</code>	Read an element from a 2D image
<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, uint4 color)</code>	Write <i>color</i> value to (<i>x</i> , <i>y</i>) location specified by <i>coord</i> in the 2D image	<code>void write_imageh (image2d_t image, int2 coord, half4 color)</code> H	
<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> <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>		Read an element from a 3D image	
<code>uint4 read_imageui (image3d_t image, sampler_t sampler, int4 coord)</code> <code>uint4 read_imageui (image3d_t image, sampler_t sampler, float4 coord)</code>			Read an element from a 3D image
<code>int2 get_image_dim (image2d_t image)</code> <code>int2 get_image_dim (image3d_t image)</code>			Image width, height, and depth
<code>int get_image_width (image2d_t image)</code> <code>int get_image_width (image3d_t image)</code>			Image width in pixels
<code>int get_image_height (image2d_t image)</code> <code>int get_image_height (image3d_t image)</code>			Image height in pixels
<code>int get_image_depth (image3d_t image)</code>			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>			Image width, height
<code>int4 get_image_dim (image3d_t image)</code>			Image width, height, and depth
<code>void write_imagef (image3d_t image, int4 coord, float4 color)</code> 3D <code>void write_imagei (image3d_t image, int4 coord, int4 color)</code> 3D <code>void write_imageui (image3d_t image, int4 coord, uint4 color)</code> 3D			Writes <i>color</i> at <i>coord</i> in the 3D image

Image Objects

Create Image Objects [5.3.1]

`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: (also for `clCreateImage3D`, `clGetSupportedImageFormats`)
`CL_MEM_READ_WRITE`, `CL_MEM_WRITE_READ_ONLY`, `CL_MEM_USE_ALLOC_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: See `clCreateImage2D`

Query List of Supported Image Formats [5.3.2]

`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: See `clCreateImage2D`

Copy Between Image, Buffer Objects [5.3.4]

`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 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 [5.3.5]

`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 [5.3.3]

`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 [5.3.6]

`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_REFERENCE_COUNT`, `CL_MEM_CONTEXT_OFFSET`, `CL_MEM_ASSOCIATED_MEMOBJECT`

`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_ROW_SLICE_PITCH`, `CL_IMAGE_HEIGHT`, `CL_IMAGE_WIDTH`, `CL_IMAGE_DEPTH`, `CL_IMAGE_D3D10_SUBRESOURCE_KHR`, `CL_MEM_D3D10_RESOURCE_KHR`

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. The default qualifier is `__read_only`.

`__read_only`, `read_only`
`__write_only`, `write_only`

Image Formats [5.3.1.1, 9.5]

Supported image formats: `image_channel_order` with `image_channel_data_type`.

Built-in support: [Table 5.7]

CL_RGBA: `CL_HALF_FLOAT`, `CL_FLOAT`, `CL_UNORM_INT8`, `CL_SIGNED_INT8`, `CL_UNSIGNED_INT8`, `CL_UNSIGNED_INT16`, `CL_UNSIGNED_INT32`

CL_BGRA: `CL_UNORM_INT8`

Optional support: [Table 5.5]

CL_R, **CL_A**: `CL_HALF_FLOAT`, `CL_FLOAT`, `CL_UNORM_INT8`, `CL_SIGNED_INT8`, `CL_UNSIGNED_INT8`, `CL_UNSIGNED_INT16`, `CL_UNSIGNED_INT32`, `CL_SNORM_INT8`, `CL_SNORM_INT16`

CL_INTENSITY: `CL_HALF_FLOAT`, `CL_FLOAT`, `CL_UNORM_INT8`, `CL_SIGNED_INT8`, `CL_UNSIGNED_INT8`, `CL_UNSIGNED_INT16`

CL_LUMINANCE: `CL_UNORM_INT8`, `CL_HALF_FLOAT`, `CL_FLOAT`, `CL_UNORM_INT16`, `CL_SIGNED_INT16`, `CL_UNSIGNED_INT16`

CL_RG, **CL_RA**: `CL_HALF_FLOAT`, `CL_FLOAT`, `CL_UNORM_INT8`, `CL_SIGNED_INT8`, `CL_UNSIGNED_INT8`, `CL_UNSIGNED_INT16`

CL_RGB: `CL_UNORM_SHORT`, `CL_UNSIGNED_SHORT`, `CL_UNORM_INT101010`

CL_ARGB: `CL_UNORM_INT8`, `CL_SIGNED_INT8`, `CL_UNSIGNED_INT8`, `CL_UNSIGNED_INT16`

CL_BGRA: `CL_SIGNED_INT8`, `CL_UNSIGNED_INT8`, `CL_UNSIGNED_INT16`

Sampler Objects [5.5]

`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_name: `CL_SAMPLER_REFERENCE_COUNT`, `CL_SAMPLER_CONTEXT_FILTER_MODE`, `CL_SAMPLER_ADDRESSING_MODE`, `CL_SAMPLER_NORMALIZED_COORDS`

Sampler Declaration Fields [6.11.13.1]

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, FALSE}`

address-mode:
`CLK_ADDRESS_{REPEAT, CLAMP, NONE},`
`CLK_ADDRESS_{CLAMP_TO_EDGE, MIRRORED_REPEAT}`

filter-mode:
`CLK_FILTER_NEAREST, CLK_FILTER_LINEAR`

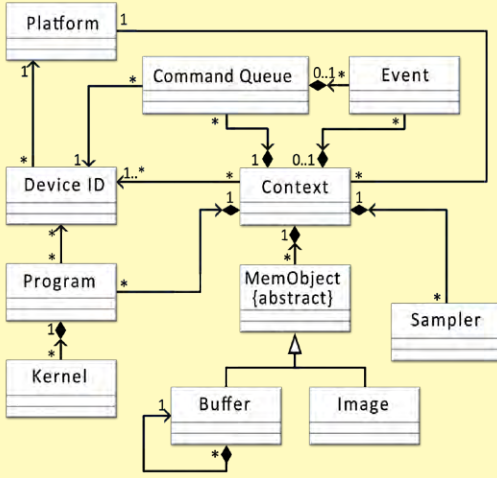
OpenCL Class Diagram [5.13]

The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language¹ (UML) notation. The diagram shows both nodes and edges which are classes and their relationships. As a simplification it shows only classes, and no attributes or operations.

Annotations

Relationships	
abstract classes	{abstract}
aggregations	◆
inheritance	△
relationship navigability	^

Cardinality	
many	*
one and only one	1
optionally one	0..1
one or more	1..*



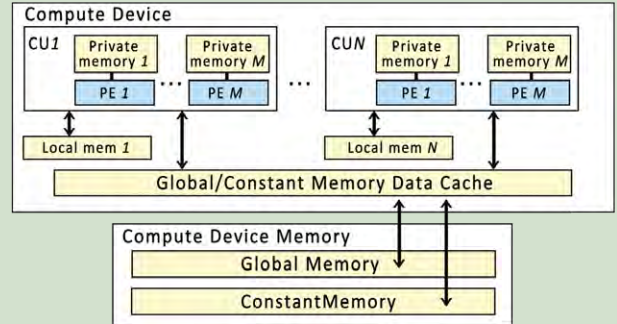
¹ Unified Modeling Language (<http://www.uml.org/>) is a trademark of Object Management Group (OMG).

OpenCL Device Architecture Diagram [3.3]

The table below shows memory regions with allocation and memory access capabilities.

	Global	Constant	Local	Private
Host	Dynamic allocation Read/Write access	Dynamic allocation Read/Write access	Dynamic allocation No access	No allocation No access
Kernel	No allocation Read/Write access	Static allocation Read-only access	Static allocation Read/Write access	Static allocation Read/Write access

This conceptual OpenCL device architecture diagram shows processing elements (PE), compute units (CU), and devices. The host is not shown.



OpenCL/OpenGL Sharing APIs

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

CL Buffer Objects > GL Buffer Objects [9.8.2]
`cl_mem clCreateFromGLBuffer (cl_context context, cl_mem_flags flags, GLuint bufobj, int *errcode_ret)`
flags: `CL_MEM_{READ, WRITE}_ONLY, CL_MEM_READ_WRITE`

CL Image Objects > GL Textures [9.8.3]
`cl_mem clCreateFromGLTexture2D (cl_context context, cl_mem_flags flags, GLenum texture_target, GLint miplevel, GLuint texture, cl_int *errcode_ret)`
flags: See `clCreateFromGLBuffer`
texture_target: `GL_TEXTURE_{2D, 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 texture_target, GLint miplevel, GLuint texture, cl_int *errcode_ret)`
flags: See `clCreateFromGLBuffer`
texture_target: `GL_TEXTURE_3D`

CL Image Objects > GL Renderbuffers [9.8.4]

`cl_mem clCreateFromGLRenderbuffer (cl_context context, cl_mem_flags flags, GLuint renderbuffer, cl_int *errcode_ret)`
flags: `clCreateFromGLBuffer`

Query Information [9.8.5]
`cl_int clGetGLObjectInfo (cl_mem memobj, cl_gl_object_type *gl_object_type, GLuint *gl_object_name)`
**gl_object_type returns:* `CL_GL_OBJECT_BUFFER, CL_GL_OBJECT_{TEXTURE2D, 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 [9.8.6]
`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 Event Objects > GL Sync Objects [9.9]
`cl_event clCreateEventFromGLsyncKHR (cl_context context, GLsync sync, cl_int *errcode_ret)`

CL Context > GL Context, Sharegroup [9.7]
`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`

OpenCL/Direct3D 10 Sharing APIs [9.10]

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

`cl_int clGetDeviceIDsFromD3D10KHR (cl_platform_id platform, cl_d3d10_device_source_khr d3d_device_source, void *d3d_object, cl_d3d10_device_set_khr d3d_device_set, cl_uint num_entries, cl_device_id *devices, cl_uint *num_devices)`
d3d_device_source: `CL_D3D10_DEVICE_KHR, CL_D3D10_DXGI_ADAPTER_KHR`
d3d_object: `ID3D10Device, IDXGIAdapter`
d3d_device_set: `CL_ALL_DEVICES_FOR_D3D10_KHR, CL_PREFERRED_DEVICES_FOR_D3D10_KHR`

`cl_mem clCreateFromD3D10BufferKHR (cl_context context, cl_mem_flags flags, ID3D10Buffer *resource, cl_int *errcode_ret)`
flags: `CL_MEM_{READ, WRITE}_ONLY, CL_MEM_READ_WRITE`

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

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

`cl_int clEnqueueAcquireD3D10ObjectsKHR (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 clEnqueueReleaseD3D10ObjectsKHR (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)`



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.