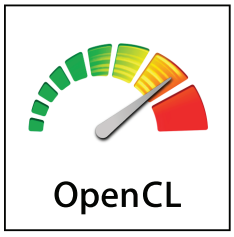

KHRONOS GROUP

The **OpenCL Extension** Specification

Version: 1.2

Document Revision: 15

Khronos OpenCL Working Group

Editor: Aaftab Munshi

9. Optional Extensions[1]

This document describes the list of optional features supported by OpenCL 1.2. Optional extensions may be supported by some OpenCL devices. Optional extensions are not required to be supported by a conformant OpenCL implementation, but are expected to be widely available; they define functionality that is likely to move into the required feature set in a future revision of the OpenCL specification. A brief description of how OpenCL extensions are defined is provided below.

For OpenCL extensions approved by the OpenCL working group, the following naming conventions are used:

- A unique *name string* of the form "**cl_khr_<*name*>**" is associated with each extension. If the extension is supported by an implementation, this string will be present in the CL_PLATFORM_EXTENSIONS or CL_DEVICE_EXTENSIONS string described in *table 4.3*.

- All API functions defined by the extension will have names of the form **cl<*FunctionName*>KHR**.

- All enumerants defined by the extension will have names of the form **CL_<*enum_name*>_KHR.**

OpenCL extensions approved by the OpenCL working group can be *promoted* to required core features in later revisions of OpenCL. When this occurs, the extension specifications are merged into the core specification. Functions and enumerants that are part of such promoted extensions will have the **KHR** affix removed. OpenCL implementations of such later revisions must also export the name strings of promoted extensions in the CL_PLATFORM_EXTENSIONS or CL_DEVICE_EXTENSIONS string, and support the **KHR**-affixed versions of functions and enumerants as a transition aid.

For vendor extensions, the following naming conventions are used:

- A unique *name string* of the form "**cl_<*vendor_name*>_<*name*>**" is associated with each extension. If the extension is supported by an implementation, this string will be present in the CL_PLATFORM_EXTENSIONS or CL_DEVICE_EXTENSIONS string described in *table 4.3*.

- All API functions defined by the vendor extension will have names of the form **cl<*FunctionName*><*vendor_name*>**.

---

[1] This document describes *section 9* of the OpenCL 1.2 specification. Any reference to *section 1.x – 8.x* or *tables 1.x – 8.x* in this document refer to sections and tables described in the OpenCL 1.2 specification.

- All enumerants defined by the vendor extension will have names of the form **CL_*<enum_name>*_*<vendor_name>*.**

9.1 Compiler Directives for Optional Extensions

The **`#pragma OPENCL EXTENSION`** directive controls the behavior of the OpenCL compiler with respect to extensions. The **`#pragma OPENCL EXTENSION`** directive is defined as:

```
#pragma OPENCL EXTENSION extension_name : behavior
#pragma OPENCL EXTENSION all : behavior
```

where *extension_name* is the name of the extension. The *extension_name* will have names of the form **cl_khr_***<name>* for an extension approved by the OpenCL working group and will have names of the form **cl_***<vendor_name>*_*<name>* for vendor extensions. The token **all** means that the behavior applies to all extensions supported by the compiler. The *behavior* can be set to one of the following values given by the table below.

behavior	Description
enable	Behave as specified by the extension *extension_name*. Report an error on the **`#pragma OPENCL EXTENSION`** if the *extension_name* is not supported, or if **all** is specified.
disable	Behave (including issuing errors and warnings) as if the extension *extension_name* is not part of the language definition. If **all** is specified, then behavior must revert back to that of the non-extended core version of the language being compiled to. Warn on the **`#pragma OPENCL EXTENSION`** if the extension *extension_name* is not supported.

The **`#pragma OPENCL EXTENSION`** directive is a simple, low-level mechanism to set the behavior for each extension. It does not define policies such as which combinations are appropriate; those must be defined elsewhere. The order of directives matter in setting the behavior for each extension. Directives that occur later override those seen earlier. The **all** variant sets the behavior for all extensions, overriding all previously issued extension directives, but only if the *behavior* is set to **disable**.

The initial state of the compiler is as if the directive

```
#pragma OPENCL EXTENSION all : disable
```

was issued, telling the compiler that all error and warning reporting must be done according to this specification, ignoring any extensions.

Every extension which affects the OpenCL language semantics, syntax or adds built-in functions to the language must create a preprocessor `#define` that matches the extension name string. This `#define` would be available in the language if and only if the extension is supported on a given implementation.

Example:

An extension which adds the extension string "**`cl_khr_3d_image_writes`**" should also add a preprocessor **`#define`** called **`cl_khr_3d_image_writes`**. A kernel can now use this preprocessor `#define` to do something like:

```
#ifdef cl_khr_3d_image_writes
     // do something using the extension
#else
     // do something else or #error!
#endif
```

9.2 Getting OpenCL API Extension Function Pointers

The function

void* **clGetExtensionFunctionAddressForPlatform**[2] (
cl_platform_id *platform*,
const char **funcname*)

returns the address of the extension function named by *funcname* for a given *platform* The pointer returned should be cast to a function pointer type matching the extension function's definition defined in the appropriate extension specification and header file. A return value of NULL indicates that the specified function does not exist for the implementation or *platform* is not a valid platform. A non-NULL return value for **clGetExtensionFunctionAddressForPlatform** does not guarantee that an extension function is actually supported by the platform. The application must also make a corresponding query using **clGetPlatformInfo**(platform, CL_PLATFORM_EXTENSIONS, …) or **clGetDeviceInfo**(device, CL_DEVICE_EXTENSIONS, …) to determine if an extension is supported by the OpenCL implementation.

clGetExtensionFunctionAddressForPlatform may not be queried for core (non-extension) functions in OpenCL. For functions that are queryable with

[2] Since there is no way to qualify the query with a device, the function pointer returned must work for all implementations of that extension on different devices for a platform. The behavior of calling a device extension function on a device not supporting that extension is undefined.

clGetExtensionFunctionAddressForPlatform, implementations may choose to also export those functions statically from the object libraries implementing those functions. However, portable applications cannot rely on this behavior.

Function pointer typedefs must be declared for all extensions that add API entrypoints. These typedefs are a required part of the extension interface, to be provided in an appropriate header (such as `cl_ext.h` if the extension is an OpenCL extension, or `cl_gl_ext.h` if the extension is an OpenCL / OpenGL sharing extension).

The following convention must be followed for all extensions affecting the host API:

```
#ifndef extension_name
#define extension_name     1

// all data typedefs, token #defines, prototypes, and
// function pointer typedefs for this extension

// function pointer typedefs must use the
// following naming convention
typedef CL_API_ENTRY return type
                (CL_API_CALL *clextension_func_nameTAG_fn)(…);

#endif // extension_name
```

where `TAG` can be `KHR`, `EXT` or `vendor-specific`.

Consider, for example, the **cl_khr_gl_sharing** extension. This extension would add the following to cl_gl_ext.h:

```
#ifndef cl_khr_gl_sharing
#define cl_khr_gl_sharing  1

// all data typedefs, token #defines, prototypes, and
// function pointer typedefs for this extension
#define CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR   -1000
#define CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR     0x2006
#define CL_DEVICES_FOR_GL_CONTEXT_KHR            0x2007
#define CL_GL_CONTEXT_KHR                        0x2008
#define CL_EGL_DISPLAY_KHR                       0x2009
#define CL_GLX_DISPLAY_KHR                       0x200A
#define CL_WGL_HDC_KHR                           0x200B
#define CL_CGL_SHAREGROUP_KHR                    0x200C

// function pointer typedefs must use the
// following naming convention
typedef CL_API_ENTRY cl_int
      (CL_API_CALL *clGetGLContextInfoKHR_fn)(
                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*/);

#endif // cl_khr_gl_sharing
```

9.3 64-bit Atomics

The optional extensions **cl_khr_int64_base_atomics** and **cl_khr_int64_extended_atomics** implement atomic operations on 64-bit signed and unsigned integers to locations in `__global` and `__local` memory. .

An application that wants to use any of these extensions will need to include the **`#pragma OPENCL EXTENSION cl_khr_int64_base_atomics : enable`** or **`#pragma OPENCL EXTENSION cl_khr_int64_extended_atomics : enable`** directive in the OpenCL program source. The atomic functions supported by the **cl_khr_int64_base_atomics** extension are described in *table 9.1*. All of the functions listed in *table 9.1* are performed in one atomic transaction. The atomic functions supported by the **cl_khr_int64_extended_atomics** extension are described in *table 9.2*. All of the functions listed in *table 9.2* are performed in one atomic transaction.

These transactions are atomic for the device executing these atomic functions. There is no guarantee of atomicity if the atomic operations to the same memory location are being performed by kernels executing on multiple devices.

Function	Description
long **atom_add** (volatile __global long *p, long val) long **atom_add** (volatile __local long *p, long val) ulong **atom_add** (volatile __global ulong *p, ulong val) ulong **atom_add** (volatile __local ulong *p, ulong val)	Read the 64-bit value (referred to as *old*) stored at location pointed by p. Compute (old + val) and store result at location pointed by p. The function returns *old*.
long **atom_sub** (volatile __global long *p, long val) long **atom_sub** (volatile __local long *p, long val) ulong **atom_sub** (volatile __global ulong *p, ulong val) ulong **atom_sub** (volatile __local ulong *p, ulong val)	Read the 64-bit value (referred to as *old*) stored at location pointed by p. Compute (old - val) and store result at location pointed by p. The function returns *old*.
long **atom_xchg** (volatile __global long *p, long val) long **atom_xchg** (volatile __local long *p, long val) ulong **atom_xchg** (volatile __global ulong *p, ulong val) ulong **atom_xchg** (volatile __local ulong *p, ulong val)	Swaps the *old* value stored at location p with new value given by val. Returns *old* value.
long **atom_inc** (volatile __global long *p) long **atom_inc** (volatile __local long *p) ulong **atom_inc** (volatile __global ulong *p) ulong **atom_inc** (volatile __local ulong *p)	Read the 64-bit value (referred to as *old*) stored at location pointed by p. Compute (old + *1*) and store result at location pointed by p. The function returns *old*.
long **atom_dec** (volatile __global long *p) long **atom_dec** (volatile __local long *p) ulong **atom_dec** (volatile __global ulong *p)	Read the 64-bit value (referred to as *old*) stored at location pointed by p. Compute (old - *1*) and store result at location pointed by p.

ulong **atom_dec** (volatile __local ulong *p)	The function returns *old*.
long **atom_cmpxchg** (volatile __global long *p, long *cmp*, long *val*) long **atom_cmpxchg** (volatile __local long *p, long *cmp*, long *val*) ulong **atom_cmpxchg** (volatile __global ulong *p, ulong *cmp*, ulong *val*) ulong **atom_cmpxchg** (volatile __local ulong *p, ulong *cmp*, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* == *cmp*) ? *val* : *old* and store result at location pointed by *p*. The function returns *old*.

Table 9.1 *Built-in Atomic Functions for* **cl_khr_int64_base_atomics** *extension*

Function	Description
long **atom_min** (volatile __global long *p, long *val*) long **atom_min** (volatile __local long *p, long *val*) ulong **atom_min** (volatile __global ulong *p, ulong *val*) ulong **atom_min** (volatile __local ulong *p, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute **min**(*old*, *val*) and store minimum value at location pointed by *p*. The function returns *old*.
long **atom_max** (volatile __global long *p, long *val*) long **atom_max** (volatile __local long *p, long *val*) ulong **atom_max** (volatile __global ulong *p, ulong *val*) ulong **atom_max** (volatile __local ulong *p, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute **max**(*old*, *val*) and store maximum value at location pointed by *p*. The function returns *old*.
long **atom_and** (volatile __global long *p, long *val*) long **atom_and** (volatile __local long *p, long *val*) ulong **atom_and** (volatile __global ulong *p, ulong *val*) ulong **atom_and** (volatile __local ulong *p, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* & val) and store result at location pointed by *p*. The function returns *old*.
long **atom_or** (volatile __global long *p, long *val*) long **atom_or** (volatile __local long *p, long *val*) ulong **atom_or** (volatile __global ulong *p, ulong *val*) ulong **atom_or** (volatile __local ulong *p, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* | val) and store result at location pointed by *p*. The function returns *old*.
long **atom_xor** (volatile __global long *p, long *val*) long **atom_xor** (volatile __local long *p, long *val*) ulong **atom_xor** (volatile __global ulong *p, ulong *val*) ulong **atom_xor** (volatile __local ulong *p, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* ^ val) and store result at location pointed by *p*. The function returns *old*.

Table 9.2 *Built-in Atomic Functions for* **cl_khr_int64_extended_atomics** *extension*

Note: Atomic operations on 64-bit integers and 32-bit integers (and float) are also atomic w.r.t. each other.

9.4 Writing to 3D image memory objects

OpenCL supports 2D image memory objects that can be read or written by kernels. Reads and writes to the same 2D image memory object are not supported in a kernel. OpenCL also supports reads to 3D image memory objects in kernels. Writes to a 3D image memory object are not supported unless the **`cl_khr_3d_image_writes`** extension is implemented. Reads and writes to the same 3D image memory object are not allowed in a kernel.

An application that wants to use this extension to write to 3D image memory objects will need to include the **`#pragma OPENCL EXTENSION cl_khr_3d_image_writes : enable`** directive in the OpenCL program source.

The built-in functions implemented by the **`cl_khr_3d_image_writes`** extension are described in the table below.

Function	Description
void **write_imagef** (image3d_t *image*, int4 *coord*, float4 *color*) void **write_imagei** (image3d_t *image*, int4 *coord*, int4 *color*) void **write_imageui** (image3d_t *image*, int4 *coord*, uint4 *color*)	Write *color* value to location specified by coordinate *(x, y, z)* in the 3D image object specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *coord.x, coord.y and coord.z* are considered to be unnormalized coordinates and must be in the range 0 ... image width – 1, 0 … image height – 1 and 0 … image depth – 1. **write_imagef** can only be used with image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or set to CL_SNORM_INT8, CL_UNORM_INT8, CL_SNORM_INT16, CL_UNORM_INT16, CL_HALF_FLOAT or CL_FLOAT. Appropriate data format conversion will be done to convert channel data from a floating-point value to actual data format in which the channels are stored. **write_imagei** can only be used with image objects created with *image_channel_data_type* set to one of the following values: CL_SIGNED_INT8, CL_SIGNED_INT16 and CL_SIGNED_INT32. **write_imageui** can only be used with image

	objects created with *image_channel_data_type* set to one of the following values: CL_UNSIGNED_INT8, CL_UNSIGNED_INT16 and CL_UNSIGNED_INT32. The behavior of **write_imagef**, **write_imagei** and **write_imageui** for image objects with *image_channel_data_type* values not specified in the description above or with (*x*, *y*, *z*) coordinate values that are not in the range (0 … image width – 1, 0 … image height – 1, 0 … image depth – 1) respectively is undefined.

9.5 Half Precision Floating-Point

This extension adds support for half scalar and vector types as built-in types that can be used for arithmetic operations, conversions etc. An application that wants to use `half` and `halfn` types will need to include the **`#pragma OPENCL EXTENSION cl_khr_fp16 : enable`** directive.

The list of built-in scalar, and vector data types defined in *tables 6.1*, and *6.2* are extended to include the following:

Type	Description
half2	A 2-component half-precision floating-point vector.
half3	A 3-component half-precision floating-point vector.
half4	A 4-component half-precision floating-point vector.
half8	A 8-component half-precision floating-point vector.
half16	A 16-component half-precision floating-point vector.

The built-in vector data types for `halfn` are also declared as appropriate types in the OpenCL API (and header files) that can be used by an application. The following table describes the built-in vector data types for `halfn` as defined in the OpenCL C programming language and the corresponding data type available to the application:

Type in OpenCL Language	API type for application
half2	**cl_half2**
half 3	**cl_half3**
half 4	**cl_half4**
half 8	**cl_half8**
half16	**cl_half16**

The relational, equality, logical and logical unary operators described in *section 6.3* can be used with `half` scalar and `half`*`n`* vector types and shall produce a scalar `int` and vector `short`*`n`* result respectively.

The OpenCL compiler accepts an h and H suffix on floating point literals, indicating the literal is typed as a half .

9.5.1 Conversions

The implicit conversion rules specified in *section 6.2.1* now include the `half` scalar and `half`*`n`* vector data types.

The explicit casts described in *section 6.2.2* are extended to take a `half` scalar data type and a

`halfn` vector data type.

The explicit conversion functions described in *section 6.2.3* are extended to take a `half` scalar data type and a `halfn` vector data type.

The `as_typen()` function for re-interpreting types as described in *section 6.2.4.2* is extended to allow conversion-free casts between `shortn`, `ushortn` and `halfn` scalar and vector data types.

9.5.2 Math Functions

The built-in math functions defined in *table 6.8* (also listed below) are extended to include appropriate versions of functions that take `half,` and `half{2|3|4|8|16}` as arguments and return values. `gentype` now also includes `half, half2, half3, half4, half8` and `half16`.

For any specific use of a function, the actual type has to be the same for all arguments and the return type.

Function	Description
gentype **acos** (gentype)	Arc cosine function.
gentype **acosh** (gentype)	Inverse hyperbolic cosine.
gentype **acospi** (gentype *x*)	Compute **acos** (*x*) / π.
gentype **asin** (gentype)	Arc sine function.
gentype **asinh** (gentype)	Inverse hyperbolic sine.
gentype **asinpi** (gentype *x*)	Compute **asin** (*x*) / π.
gentype **atan** (gentype *y_over_x*)	Arc tangent function.
gentype **atan2** (gentype *y*, gentype *x*)	Arc tangent of *y* / *x*.
gentype **atanh** (gentype)	Hyperbolic arc tangent.
gentype **atanpi** (gentype *x*)	Compute **atan** (*x*) / π.
gentype **atan2pi** (gentype *y*, gentype *x*)	Compute **atan2** (*y*, *x*) / π.
gentype **cbrt** (gentype)	Compute cube-root.
gentype **ceil** (gentype)	Round to integral value using the round to positive infinity rounding mode.
gentype **copysign** (gentype *x*, gentype *y*)	Returns *x* with its sign changed to match the sign of *y*.
gentype **cos** (gentype)	Compute cosine.
gentype **cosh** (gentype)	Compute hyperbolic consine.
gentype **cospi** (gentype *x*)	Compute **cos** (π *x*).
gentype **erfc** (gentype)	Complementary error function.
gentype **erf** (gentype)	Error function encountered in integrating the normal distribution.

gentype **exp** (gentype *x*)	Compute the base- e exponential of *x*.
gentype **exp2** (gentype)	Exponential base 2 function.
gentype **exp10** (gentype)	Exponential base 10 function.
gentype **expm1** (gentype *x*)	Compute e^x- 1.0.
gentype **fabs** (gentype)	Compute absolute value of a floating-point number.
gentype **fdim** (gentype *x*, gentype *y*)	*x* - *y* if *x* > *y*, +0 if x is less than or equal to y.
gentype **floor** (gentype)	Round to integral value using the round to negative infinity rounding mode.
gentype **fma** (gentype *a*, gentype *b*, gentype *c*)	Returns the correctly rounded floating-point representation of the sum of *c* with the infinitely precise product of *a* and *b*. Rounding of intermediate products shall not occur. Edge case behavior is per the IEEE 754-2008 standard.
gentype **fmax** (gentype x, gentype y) gentype **fmax** (gentype *x*, half *y*)	Returns *y* if *x* < *y*, otherwise it returns *x*. If one argument is a NaN, **fmax()** returns the other argument. If both arguments are NaNs, **fmax()** returns a NaN.
gentype **fmin** (gentype *x*, gentype *y*) gentype **fmin** (gentype *x*, half *y*)	Returns *y* if *y* < *x*, otherwise it returns *x*. If one argument is a NaN, **fmin()** returns the other argument. If both arguments are NaNs, **fmin()** returns a NaN.
gentype **fmod** (gentype *x*, gentype *y*)	Modulus. Returns *x* – *y* * **trunc** (*x*/*y*) .
gentype **fract** (gentype *x*, __global gentype **iptr*) gentype **fract** (gentype *x*, __local gentype **iptr*) gentype **fract** (gentype *x*, __private gentype **iptr*)	Returns **fmin**(*x* – **floor** (*x*), 0x1.ffcp-1f). **floor**(x) is returned in *iptr*.
half*n* **frexp** (half*n* *x*, __global int*n* *exp) half*n* **frexp** (half*n* *x*, __local int*n* *exp) half*n* **frexp** (half*n* *x*, __private int*n* *exp) half **frexp** (half *x*, __global int *exp) half **frexp** (half *x*, __local int *exp) half **frexp** (half *x*, __private int *exp)	Extract mantissa and exponent from *x*. For each component the mantissa returned is a float with magnitude in the interval [1/2, 1) or 0. Each component of *x* equals mantissa returned * 2^{exp}.
gentype **hypot** (gentype *x*, gentype *y*)	Compute the value of the square root of $x^2 + y^2$ without undue overflow or underflow.
int*n* **ilogb** (half*n* *x*) int **ilogb** (half *x*)	Return the exponent as an integer value.
half*n* **ldexp** (half*n* *x*, int*n* *k*) half*n* **ldexp** (half*n* *x*, int *k*)	Multiply *x* by 2 to the power *k*.

half **ldexp** (half *x*, int *k*)	
gentype **lgamma** (gentype *x*) half*n* **lgamma_r** (half*n* *x*, __global int*n* **signp*) half*n* **lgamma_r** (half*n* *x*, __local int*n* **signp*) half*n* **lgamma_r** (half*n* *x*, __private int*n* **signp*) half **lgamma_r** (half *x*, __global int **signp*) half **lgamma_r** (half *x*, __local int **signp*) half **lgamma_r** (half *x*, __private int **signp*)	Log gamma function. Returns the natural logarithm of the absolute value of the gamma function. The sign of the gamma function is returned in the *signp* argument of **lgamma_r**.
gentype **log** (gentype)	Compute natural logarithm.
gentype **log2** (gentype)	Compute a base 2 logarithm.
gentype **log10** (gentype)	Compute a base 10 logarithm.
gentype **log1p** (gentype *x*)	Compute $\log_e(1.0 + x)$.
gentype **logb** (gentype *x*)	Compute the exponent of *x*, which is the integral part of $\log_r \mid x \mid$.
gentype **mad** (gentype *a*, gentype *b*, gentype *c*)	**mad** approximates *a* * *b* + *c*. Whether or how the product of *a* * *b* is rounded and how supernormal or subnormal intermediate products are handled is not defined. **mad** is intended to be used where speed is preferred over accuracy[3].
gentype **maxmag** (gentype *x*, gentype *y*)	Returns *x* if $\mid x \mid > \mid y \mid$, *y* if $\mid y \mid > \mid x \mid$, otherwise **fmax**(*x*, *y*).
gentype **minmag** (gentype *x*, gentype *y*)	Returns *x* if $\mid x \mid < \mid y \mid$, *y* if $\mid y \mid < \mid x \mid$, otherwise **fmin**(*x*, *y*).
gentype **modf** (gentype *x*, __global gentype **iptr*) gentype **modf** (gentype *x*, __local gentype **iptr*) gentype **modf** (gentype *x*, __private gentype **iptr*)	Decompose a floating-point number. The **modf** function breaks the argument *x* into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part in the object pointed to by *iptr*.
half*n* **nan** (ushort*n* *nancode*) half **nan** (ushort *nancode*)	Returns a quiet NaN. The *nancode* may be placed in the significand of the resulting NaN.
gentype **nextafter** (gentype *x*, gentype *y*)	Computes the next representable half-precision floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, **nextafter**() returns the largest representable floating-point number less

[3] The user is cautioned that for some usages, e.g. **mad**(a, b, -a*b), the definition of **mad**() is loose enough that almost any result is allowed from **mad**() for some values of a and b.

	than *x*.
gentype **pow** (gentype *x*, gentype *y*)	Compute *x* to the power *y*.
half*n* **pown** (half*n* *x*, int*n* *y*) half **pown** (half *x*, int *y*)	Compute *x* to the power *y*, where *y* is an integer.
gentype **powr** (gentype *x*, gentype *y*)	Compute *x* to the power *y*, where *x* is >= 0.
gentype **remainder** (gentype *x*, gentype *y*)	Compute the value *r* such that *r* = *x* - *n***y*, where *n* is the integer nearest the exact value of *x*/*y*. If there are two integers closest to *x*/*y*, *n* shall be the even one. If *r* is zero, it is given the same sign as *x*.
half*n* **remquo** (half*n* *x*, half*n* *y*, __global int*n* **quo*) half*n* **remquo** (half*n* *x*, half*n* *y*, __local int*n* **quo*) half*n* **remquo** (half*n* *x*, half*n* *y*, __private int*n* **quo*) half **remquo** (half *x*, half *y*, __global int **quo*) half **remquo** (half *x*, half *y*, __local int **quo*) half **remquo** (half *x*, half *y*, __private int **quo*)	The **remquo** function computes the value r such that *r* = *x* - *k***y*, where *k* is the integer nearest the exact value of *x*/*y*. If there are two integers closest to *x*/*y*, *k* shall be the even one. If *r* is zero, it is given the same sign as *x*. This is the same value that is returned by the **remainder** function. **remquo** also calculates the lower seven bits of the integral quotient *x*/*y*, and gives that value the same sign as *x*/*y*. It stores this signed value in the object pointed to by *quo*.
gentype **rint** (gentype)	Round to integral value (using round to nearest even rounding mode) in floating-point format. Refer to section 7.1 for description of rounding modes.
half*n* **rootn** (half*n* *x*, int*n* *y*) half **rootn** (half *x*, int *y*)	Compute *x* to the power 1/*y*.
gentype **round** (gentype *x*)	Return the integral value nearest to *x* rounding halfway cases away from zero, regardless of the current rounding direction.
gentype **rsqrt** (gentype)	Compute inverse square root.
gentype **sin** (gentype)	Compute sine.
gentype **sincos** (gentype *x*, __global gentype **cosval*) gentype **sincos** (gentype *x*, __local gentype **cosval*) gentype **sincos** (gentype *x*, __private gentype **cosval*)	Compute sine and cosine of x. The computed sine is the return value and computed cosine is returned in *cosval*.
gentype **sinh** (gentype)	Compute hyperbolic sine.
gentype **sinpi** (gentype *x*)	Compute **sin** (π *x*).

gentype **sqrt** (gentype)	Compute square root.
gentype **tan** (gentype)	Compute tangent.
gentype **tanh** (gentype)	Compute hyperbolic tangent.
gentype **tanpi** (gentype *x*)	Compute **tan** (πx).
gentype **tgamma** (gentype)	Compute the gamma function.
gentype **trunc** (gentype)	Round to integral value using the round to zero rounding mode.

Table 6.8 *Scalar and Vector Argument Built-in Math Function Table*

The **FP_FAST_FMA_HALF** macro indicates whether the **fma()** family of functions are fast compared with direct code for half precision floating-point. If defined, the **FP_FAST_FMA_HALF** macro shall indicate that the **fma()** function generally executes about as fast as, or faster than, a multiply and an add of **half** operands

The macro names given in the following list must use the values specified. These constant expressions are suitable for use in #if preprocessing directives.

```
#define HALF_DIG          3
#define HALF_MANT_DIG     11
#define HALF_MAX_10_EXP   +4
#define HALF_MAX_EXP      +16
#define HALF_MIN_10_EXP   -4
#define HALF_MIN_EXP      -13
#define HALF_RADIX        2
#define HALF_MAX          0x1.ffcp15h
#define HALF_MIN          0x1.0p-14h
#define HALF_EPSILON      0x1.0p-10h
```

The following table describes the built-in macro names given above in the OpenCL C programming language and the corresponding macro names available to the application.

Macro in OpenCL Language	Macro for application
HALF_DIG	**CL_HALF_DIG**
HALF_MANT_DIG	**CL_HALF_MANT_DIG**
HALF_MAX_10_EXP	**CL_HALF_MAX_10_EXP**
HALF_MAX_EXP	**CL_HALF_MAX_EXP**
HALF_MIN_10_EXP	**CL_HALF_MIN_10_EXP**
HALF_MIN_EXP	**CL_HALF_MIN_EXP**
HALF_RADIX	**CL_HALF_RADIX**
HALF_MAX	**CL_HALF_MAX**
HALF_MIN	**CL_HALF_MIN**
HALF_EPSILSON	**CL_HALF_EPSILON**

The following constants are also available. They are of type `half` and are accurate within the precision of the `half` type.

Constant	Description
M_E_H	Value of e
M_LOG2E_H	Value of $\log_2 e$
M_LOG10E_H	Value of $\log_{10} e$
M_LN2_H	Value of $\log_e 2$
M_LN10_H	Value of $\log_e 10$
M_PI_H	Value of π
M_PI_2_H	Value of $\pi / 2$
M_PI_4_H	Value of $\pi / 4$
M_1_PI_H	Value of $1 / \pi$
M_2_PI_H	Value of $2 / \pi$
M_2_SQRTPI_H	Value of $2 / \sqrt{\pi}$
M_SQRT2_H	Value of $\sqrt{2}$
M_SQRT1_2_H	Value of $1 / \sqrt{2}$

9.5.3 Common Functions[4]

The built-in common functions defined in *table 6.12* (also listed below) are extended to include appropriate versions of functions that take `half`, and `half{2|3|4|8|16}` as arguments and return values. `gentype` now also includes `half, half2, half3, half4, half8` and `half16`. These are described below.

Function	Description
gentype **clamp** (gentype *x*, gentype *minval*, gentype *maxval*) gentype **clamp** (gentype *x*, half *minval*, half *maxval*)	Returns **min**(**max**(*x*, *minval*), *maxval*) . Results are undefined if *minval* > *maxval*.
gentype **degrees** (gentype *radians*)	Converts *radians* to degrees, i.e. (180 / π) * *radians*.
gentype **max** (gentype *x*, gentype *y*) gentype **max** (gentype *x*, half *y*)	Returns *y* if *x* < *y*, otherwise it returns *x*. If *x* and *y* are infinite or NaN, the return values are undefined.

[4] The **mix** and **smoothstep** functions can be implemented using contractions such as **mad** or **fma**.

gentype **min** (gentype *x*, gentype *y*) gentype **min** (gentype *x*, half *y*)	Returns *y* if *y* < *x*, otherwise it returns *x*. If *x* and *y* are infinite or NaN, the return values are undefined.
gentype **mix** (gentype *x*, gentype *y*, gentype *a*) gentype **mix** (gentype *x*, gentype *y*, half *a*)	Returns the linear blend of *x* & *y* implemented as: *x* + (*y* – *x*) * *a* *a* must be a value in the range 0.0 … 1.0. If *a* is not in the range 0.0 … 1.0, the return values are undefined.
gentype **radians** (gentype *degrees*)	Converts *degrees* to radians, i.e. (π / 180) * *degrees*.
gentype **step** (gentype *edge*, gentype *x*) gentype **step** (half *edge*, gentype *x*)	Returns 0.0 if *x* < *edge*, otherwise it returns 1.0.
gentype **smoothstep** (gentype *edge0*, gentype *edge1*, gentype *x*) gentype **smoothstep** (half *edge0*, half *edge1*, gentype *x*)	Returns 0.0 if *x* <= *edge0* and 1.0 if *x* >= *edge1* and performs smooth Hermite interpolation between 0 and 1when *edge0* < *x* < *edge1*. This is useful in cases where you would want a threshold function with a smooth transition. This is equivalent to: gentype t; t = clamp ((x – edge0) / (edge1 – edge0), 0, 1); return t * t * (3 – 2 * t); Results are undefined if *edge0* >= *edge1*.
gentype **sign** (gentype *x*)	Returns 1.0 if *x* > 0, -0.0 if *x* = -0.0, +0.0 if *x* = +0.0, or –1.0 if *x* < 0. Returns 0.0 if *x* is a NaN.

Table 6.12 *Scalar and Vector Argument Built-in Common Function Table*

9.5.4 Geometric Functions[5]

The built-in geometric functions defined in *table 6.13* (also listed below) are extended to include appropriate versions of functions that take `half`, and `half{2|3|4}` as arguments and return values. `gentype` now also includes `half, half2, half3` and `half4`. These are described below.

[5] The geometric functions can be implemented using contractions such as **mad** or **fma**.

Function	Description
half4 **cross** (half4 *p0*, half4 *p1*) half3 **cross** (half3 *p0*, half3 *p1*)	Returns the cross product of *p0.xyz* and *p1.xyz*. The *w* component of double result will be 0.0.
half **dot** (gentype *p0*, gentype *p1*)	Compute dot product.
half **distance** (gentype *p0*, gentype *p1*)	Returns the distance between *p0* and *p1*. This is calculated as **length**(*p0 – p1*).
half **length** (gentype *p*)	Return the length of vector x, i.e., $\sqrt{p.x^2 + p.y^2 + \ldots}$
gentype **normalize** (gentype *p*)	Returns a vector in the same direction as *p* but with a length of 1.

Table 6.13 *Scalar and Vector Argument Built-in Geometric Function Table*

9.5.5 Relational Functions

The scalar and vector relational functions described in *table 6.14* are extended to include versions that take `half`, `half2`, `half3`, `half4`, `half8` and `half16` as arguments.

The relational and equality operators (<, <=, >, >=, !=, ==) can be used with `half`*n* vector types and shall produce a vector `short`*n* result as described in *section 6.3*.

The functions **isequal**, **isnotequal**, **isgreater**, **isgreaterequal**, **isless**, **islessequal**, **islessgreater**, **isfinite**, **isinf**, **isnan**, **isnormal**, **isordered**, **isunordered** and **signbit** shall return a 0 if the specified relation is *false* and a 1 if the specified relation is true for scalar argument types. These functions shall return a 0 if the specified relation is *false* and a –1 (i.e. all bits set) if the specified relation is *true* for vector argument types.

The relational functions **isequal**, **isgreater**, **isgreaterequal**, **isless**, **islessequal**, and **islessgreater** always return 0 if either argument is not a number (NaN). **isnotequal** returns 1 if one or both arguments are not a number (NaN) and the argument type is a scalar and returns -1 if one or both arguments are not a number (NaN) and the argument type is a vector.

The functions described in *table 6.14* are extended to include the `half`*n* vector types.

Function	Description
int **isequal** (half *x*, half *y*) short*n* **isequal** (half*n* *x*, half*n* *y*)	Returns the component-wise compare of *x* == *y*.
int **isnotequal** (half *x*, half *y*) short*n* **isnotequal** (half*n* *x*, half*n* *y*)	Returns the component-wise compare of *x* != *y*.
int **isgreater** (half *x*, half *y*) short*n* **isgreater** (half*n* *x*, half*n* *y*)	Returns the component-wise compare of *x* > *y*.
int **isgreaterequal** (half *x*,	Returns the component-wise compare of *x* >= *y*.

half *y*) short*n* **isgreaterequal** (half*n x*, half*n y*)	
int **isless** (half *x*, half *y*) short*n* **isless** (half*n x*, half*n y*)	Returns the component-wise compare of $x < y$.
int **islessequal** (half *x*, half *y*) short*n* **islessequal** (half*n x*, half*n y*)	Returns the component-wise compare of $x <= y$.
int **islessgreater** (half *x*, half *y*) short*n* **islessgreater** (half*n x*, half*n y*)	Returns the component-wise compare of $(x < y) \mid\mid (x > y)$.
int **isfinite** (half) short*n* **isfinite** (half*n*)	Test for finite value.
int **isinf** (half) short*n* **isinf** (half*n*)	Test for infinity value (positive or negative) .
int **isnan** (half) short*n* **isnan** (half*n*)	Test for a NaN.
int **isnormal** (half) short*n* **isnormal** (half*n*)	Test for a normal value.
int **isordered** (half *x*, half *y*) short*n* **isordered** (half*n x*, half*n y*)	Test if arguments are ordered. **isordered**() takes arguments *x* and *y*, and returns the result **isequal**(*x*, *x*) && **isequal**(*y*, *y*).
int **isunordered** (half *x*, half *y*) short*n* **isunordered** (half*n x*, half*n y*)	Test if arguments are unordered. **isunordered**() takes arguments *x* and *y*, returning non-zero if *x* or *y* is a NaN, and zero otherwise.
int **signbit** (half) short*n* **signbit** (half*n*)	Test for sign bit. The scalar version of the function returns a 1 if the sign bit in the half is set else returns 0. The vector version of the function returns the following for each component in half*n*: -1 (i.e all bits set) if the sign bit in the half is set else returns 0.
half*n* **bitselect** (half*n a*, half*n b*, half*n c*)	Each bit of the result is the corresponding bit of *a* if the corresponding bit of *c* is 0. Otherwise it is the corresponding bit of *b*.
half*n* **select** (half*n a*, half*n b*, short*n c*) half*n* **select** (half*n a*, half*n b*, ushort*n c*)	For each component, *result[i]* = if MSB of *c[i]* is set ? *b[i]* : *a[i]*. igentype and ugentype must have the same number of elements and bits as gentype.

Table 6.14 *Vector Relational Functions*

9.5.6 Vector Data Load and Store Functions

The vector data load (**vload*n***) and store (**vstore*n***) functions described in *table 6.14* (also listed below) are extended to include versions that read from or write to half scalar or vector values. The generic type `gentype` is extended to include `half`. The generic type `gentype`*n* is extended to include `half`,`half2`, `half3`, `half4`, `half8` and `half16`.

Function	Description
gentype*n* **vload*n*** (size_t *offset*, const __global gentype *p) gentype*n* **vload*n*** (size_t *offset*, const __local gentype *p) gentype*n* **vload*n*** (size_t *offset*, const __constant gentype *p) gentype*n* **vload*n*** (size_t *offset*, const __private gentype *p)	Return sizeof (gentype*n*) bytes of data read from address (*p* + (*offset* * *n*)). The read address computed as (*p* + (*offset* * *n*)) must be 16-bit aligned.
void **vstore*n*** (gentype*n* *data*, size_t *offset*, __global gentype *p) void **vstore*n*** (gentype*n* *data*, size_t *offset*, __local gentype *p) void **vstore*n*** (gentype*n* *data*, size_t *offset*, __private gentype *p)	Write sizeof (gentype*n*) bytes given by *data* to address (*p* + (*offset* * *n*)). The write address computed as (*p* + (*offset* * *n*)) must be 16-bit aligned.

Table 6.15 *Vector Data Load and Store Functions*[6]

9.5.7 Async Copies from Global to Local Memory, Local to Global Memory, and Prefetch

The OpenCL C programming language implements the following functions that provide asynchronous copies between global and local memory and a prefetch from global memory.

The generic type `gentype` is extended to include `half`, `half2`, `half3`, `half4`, `half8` and `half16`.

[6] **vload*3*** reads *x*, *y*, *z* components from address (*p* + (*offset* * 3)) into a 3-component vector and **vstore*3*** writes *x*, *y*, *z* components from a 3-component vector to address (*p* + (*offset* * 3)).

Function	Description
event_t **async_work_group_copy** (__local gentype **dst*, const __global gentype **src*, size_t *num_gentypes*, event_t *event*) event_t **async_work_group_copy** (__global gentype **dst*, const __local gentype **src*, size_t *num_gentypes*, event_t *event*)	Perform an async copy of *num_gentypes* `gentype` elements from *src* to *dst*. The async copy is performed by all work-items in a work-group and this built-in function must therefore be encountered by all work-items in a work-group executing the kernel with the same argument values; otherwise the results are undefined. Returns an event object that can be used by **wait_group_events** to wait for the async copy to finish. The *event* argument can also be used to associate the **async_work_group_copy** with a previous async copy allowing an event to be shared by multiple async copies; otherwise *event* should be zero. If *event* argument is not zero, the event object supplied in *event* argument will be returned. This function does not perform any implicit synchronization of source data such as using a **barrier** before performing the copy.
event_t **async_work_group_strided_copy** (__local gentype **dst*, const __global gentype **src*, size_t *num_gentypes*, size_t *src_stride*, event_t *event*) event_t **async_work_group_strided_copy** (__global gentype **dst*, const __local gentype **src*, size_t *num_gentypes*, size_t *dst_stride*, event_t *event*)	Perform an async gather of *num_gentypes* `gentype` elements from *src* to *dst*. The *src_stride* is the stride in elements for each gentype element read from *src*. The async gather is performed by all work-items in a work-group and this built-in function must therefore be encountered by all work-items in a work-group executing the kernel with the same argument values; otherwise the results are undefined. Returns an event object that can be used by **wait_group_events** to wait for the async copy to finish. The *event* argument can also be used to associate the **async_work_group_strided_copy** with a

	previous async copy allowing an event to be shared by multiple async copies; otherwise *event* should be zero. If *event* argument is not zero, the event object supplied in *event* argument will be returned. This function does not perform any implicit synchronization of source data such as using a **barrier** before performing the copy. The behavior of **async_work_group_strided_copy** is undefined if *src_stride* or *dst_stride* is 0, or if the *src_stride* or *dst_stride* values cause the *src* or *dst* pointers to exceed the upper bounds of the address space during the copy.
void **wait_group_events** (int *num_events*, event_t **event_list*)	Wait for events that identify the **async_work_group_copy** operations to complete. The event objects specified in *event_list* will be released after the wait is performed. This function must be encountered by all work-items in a work-group executing the kernel with the same *num_events* and event objects specified in *event_list*; otherwise the results are undefined.
void **prefetch** (const __global gentype **p*, size_t *num_gentypes*)	Prefetch *num_gentypes* * sizeof(`gentype`) bytes into the global cache. The prefetch instruction is applied to a work-item in a work-group and does not affect the functional behavior of the kernel.

Table 6.18 *Built-in Async Copy and Prefetch functions*

9.5.8 Image Read and Write Functions

The image read and write functions defined in *tables 6.23*, *6.24* and *6.25* are extended to support image color values that are a `half` type.

Function	Description
half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, float2 *coord*)	Use the coordinate *(coord.x, coord.y)* to do an element lookup in the 2D image object specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats, CL_UNORM_INT8, or CL_UNORM_INT16. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to CL_SNORM_INT8, or CL_SNORM_INT16. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to CL_HALF_FLOAT. The **read_imageh** calls that take integer coordinates must use a sampler with filter mode set to CLK_FILTER_NEAREST, normalized coordinates set to CLK_NORMALIZED_COORDS_FALSE and addressing mode set to CLK_ADDRESS_CLAMP_TO_EDGE, CLK_ADDRESS_CLAMP or CLK_ADDRESS_NONE; otherwise the values returned are undefined. Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description above are undefined.
half4 **read_imageh** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) half4 **read_imageh** (image3d_t *image*, sampler_t *sampler*, float4 *coord*)	Use the coordinate *(coord.x*, *coord.y*, *coord.z)* to do an element lookup in the 3D image object specified by *image*. *coord.w* is ignored. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16.

	read_imageh returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to CL_SNORM_INT8, or CL_SNORM_INT16. **read_imageh**returns half precision floating-point values for image objects created with *image_channel_data_type* set to CL_HALF_FLOAT. The **read_imageh** calls that take integer coordinates must use a sampler with filter mode set to CLK_FILTER_NEAREST, normalized coordinates set to CLK_NORMALIZED_COORDS_FALSE and addressing mode set to CLK_ADDRESS_CLAMP_TO_EDGE, CLK_ADDRESS_CLAMP or CLK_ADDRESS_NONE; otherwise the values returned are undefined. Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description are undefined.
half4 **read_imageh** (image2d_array_t *image*, sampler_t *sampler*, int4 *coord*) half4 **read_imageh** (image2d_array_t *image*, sampler_t *sampler*, float4 *coord*)	Use *coord.xy* to do an element lookup in the 2D image identified by *coord.z* in the 2D image array specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with image_channel_data_type set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with image_channel_data_type set to CL_SNORM_INT8, or CL_SNORM_INT16. **read_imageh** returns half precision floating-point values for image objects created with image_channel_data_type set to CL_HALF_FLOAT. The **read_imageh** calls that take integer coordinates must use a sampler with filter mode set to CLK_FILTER_NEAREST, normalized coordinates set to CLK_NORMALIZED_COORDS_FALSE and addressing mode set to CLK_ADDRESS_CLAMP_TO_EDGE,

	CLK_ADDRESS_CLAMP or CLK_ADDRESS_NONE; otherwise the values returned are undefined. Values returned by **read_imageh** for image objects with image_channel_data_type values not specified in the description above are undefined.
half4 **read_imageh** (image1d_t *image*, sampler_t *sampler*, int *coord*) half4 **read_imageh** (image1d_t *image*, sampler_t *sampler*, float *coord*)	Use *coord* to do an element lookup in the 1D image object specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to CL_SNORM_INT8, or CL_SNORM_INT16. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to CL_HALF_FLOAT. The **read_imageh** calls that take integer coordinates must use a sampler with filter mode set to CLK_FILTER_NEAREST, normalized coordinates set to CLK_NORMALIZED_COORDS_FALSE and addressing mode set to CLK_ADDRESS_CLAMP_TO_EDGE, CLK_ADDRESS_CLAMP or CLK_ADDRESS_NONE; otherwise the values returned are undefined. Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description above are undefined.
half4 **read_imageh** (image1d_array_t *image*, sampler_t *sampler*, int2 *coord*) half4 **read_imageh** (image1d_array_t *image*, sampler_t *sampler*, float4 *coord*)	Use *coord.x* to do an element lookup in the 1D image identified by *coord.y* in the 1D image array specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with image_channel_data_type set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16.

	read_imageh returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with image_channel_data_type set to CL_SNORM_INT8, or CL_SNORM_INT16. **read_imageh** returns half precision floating-point values for image objects created with image_channel_data_type set to CL_HALF_FLOAT. The **read_imageh** calls that take integer coordinates must use a sampler with filter mode set to CLK_FILTER_NEAREST, normalized coordinates set to CLK_NORMALIZED_COORDS_FALSE and addressing mode set to CLK_ADDRESS_CLAMP_TO_EDGE, CLK_ADDRESS_CLAMP or CLK_ADDRESS_NONE; otherwise the values returned are undefined. Values returned by **read_imageh** for image objects with image_channel_data_type values not specified in the description above are undefined.

Table 6.23 *Built-in Image Read Functions*

Function	Description
half4 **read_imageh** (image2d_t *image*, int2 *coord*)	Use the coordinate *(coord.x, coord.y)* to do an element lookup in the 2D image object specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to CL_SNORM_INT8, or CL_SNORM_INT16. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to CL_HALF_FLOAT.

	Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description above are undefined.
half4 **read_imageh** (image3d_t *image*, int4 *coord*)	Use the coordinate *(coord.x*, *coord.y*, *coord.z)* to do an element lookup in the 3D image object specified by *image*. *coord.w* is ignored. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to CL_SNORM_INT8, or CL_SNORM_INT16. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to CL_HALF_FLOAT. Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description are undefined.
half4 **read_imageh** (image2d_array_t *image*, int4 *coord*)	Use *coord.xy* to do an element lookup in the 2D image identified by *coord.z* in the 2D image array specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to CL_SNORM_INT8, or CL_SNORM_INT16. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to CL_HALF_FLOAT.

	Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description above are undefined.
half4 **read_imageh** (image1d_t *image*, int *coord*) half4 **read_imageh** (image1d_buffer_t *image*, int *coord*)	Use *coord* to do an element lookup in the 1D image or 1D image buffer object specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to CL_SNORM_INT8, or CL_SNORM_INT16. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to CL_HALF_FLOAT. Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description above are undefined.
half4 **read_imageh** (image1d_array_t *image*, int2 *coord*)	Use *coord.x* to do an element lookup in the 2D image identified by *coord.y* in the 2D image array specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to CL_SNORM_INT8, or CL_SNORM_INT16. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to CL_HALF_FLOAT. Values returned by **read_imageh** for image objects with

	image_channel_data_type values not specified in the description above are undefined.

Table 6.24 *Built-in Image Sampler-less Read Functions*

Function	Description
void **write_imageh** (image2d_t *image*, int2 *coord*, half4 *color*)	Write *color* value to location specified by *coord.xy* in the 2D image specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *x* & *y* are considered to be unnormalized coordinates and must be in the range 0 ... width – 1, and 0 … height – 1. **write_imageh** can only be used with image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or set to CL_SNORM_INT8, CL_UNORM_INT8, CL_SNORM_INT16, CL_UNORM_INT16 or CL_HALF_FLOAT. The behavior of **write_imageh** for image objects created with *image_channel_data_type* values not specified in the description above or with (*x*, *y*) coordinate values that are not in the range (0 … width – 1, 0 … height – 1) respectively, is undefined.
void **write_imageh** (image2d_array_t *image*, int4 *coord*, half4 *color*)	Write *color* value to location specified by *coord.xy* in the 2D image identified by *coord.z* in the 2D image array specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *coord.x*, *coord.y* and *coord.z* are considered to be unnormalized coordinates and must be in the range 0 ... image width – 1, 0 … image height – 1 and 0 … image number of layers – 1. **write_imageh** can only be used with image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or set to CL_SNORM_INT8, CL_UNORM_INT8, CL_SNORM_INT16, CL_UNORM_INT16 or CL_HALF_FLOAT. The behavior of **write_imageh** for image objects created with *image_channel_data_type* values not specified in the

	description above or with (*x*, *y, z*) coordinate values that are not in the range (0 … image width – 1, 0 … image height – 1, 0 … image number of layers – 1), respectively, is undefined.
void **write_imageh** (image1d_t *image*, int *coord*, half4 *color*) void **write_imageh** (image1d_buffer_t *image*, int *coord*, half4 *color*)	Write *color* value to location specified by *coord* in the 1D image or 1D image buffer object specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *coord* is considered to be unnormalized coordinates and must be in the range 0 ... image width – 1. **write_imageh** can only be used with image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or set to CL_SNORM_INT8, CL_UNORM_INT8, CL_SNORM_INT16, CL_UNORM_INT16 or CL_HALF_FLOAT. Appropriate data format conversion will be done to convert channel data from a floating-point value to actual data format in which the channels are stored. The behavior of **write_imageh** for image objects created with *image_channel_data_type* values not specified in the description above or with coordinate values that is not in the range (0 … image width – 1), is undefined.
void **write_imageh** (image1d_array_t *image*, int2 *coord*, half4 *color*)	Write *color* value to location specified by *coord.x* in the 1D image identified by *coord.y* in the 1D image array specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *coord.x* and *coord.y* are considered to be unnormalized coordinates and must be in the range 0 ... image width – 1 and 0 … image number of layers – 1. **write_imageh** can only be used with image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or set to CL_SNORM_INT8, CL_UNORM_INT8, CL_SNORM_INT16, CL_UNORM_INT16 or CL_HALF_FLOAT. Appropriate data format conversion will be done to convert channel data from a floating-point value to actual data format in which the channels are stored. The behavior of **write_imageh** for image objects created with *image_channel_data_type* values not specified in the description above or with (*x*, *y*) coordinate values that are

	not in the range (0 … image width – 1, 0 … image number of layers – 1), respectively, is undefined.
void **write_imageh** (image3d_t *image*, int4 *coord*, half4 *color*)	Write color value to location specified by coord.xyz in the 3D image object specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. coord.x, coord.y and coord.z are considered to be unnormalized coordinates and must be in the range 0 ... image width – 1, 0 … image height – 1 and 0 … image depth – 1. **write_imageh** can only be used with image objects created with image_channel_data_type set to one of the pre-defined packed formats or set to CL_SNORM_INT8, CL_UNORM_INT8, CL_SNORM_INT16, CL_UNORM_INT16 or CL_HALF_FLOAT. The behavior of **write_imageh** for image objects created with image_channel_data_type values not specified in the description above or with (x, y, z) coordinate values that are not in the range (0 … image width – 1, 0 … image height – 1, 0 … image depth – 1), respectively, is undefined. NOTE: This built-in function is only available in the cl_khr_3d_image_writes extension is also supported by the device.

Table 6.25 *Built-in Image Write Functions*

9.5.9 IEEE754 Compliance

The following table entry describes the additions to *table 4.3,* which allows applications to query the configuration information using **clGetDeviceInfo** for an OpenCL device that supports half precision floating-point.

Op-code	Return Type	Description
CL_DEVICE_HALF_FP_CONFIG	cl_device_ fp_config	Describes half precision floating-point capability of the OpenCL device. This is a bit-field that describes one or more of the following values: CL_FP_DENORM – denorms are

		supported CL_FP_INF_NAN – INF and NaNs are supported CL_FP_ROUND_TO_NEAREST – round to nearest even rounding mode supported CL_FP_ROUND_TO_ZERO – round to zero rounding mode supported CL_FP_ROUND_TO_INF – round to positive and negative infinity rounding modes supported CP_FP_FMA – IEEE754-2008 fused multiply-add is supported. CL_FP_SOFT_FLOAT – Basic floating-point operations (such as addition, subtraction, multiplication) are implemented in software. The required minimum half precision floating-point capability as implemented by this extension is CL_FP_ROUND_TO_ZERO or CL_FP_ROUND_TO_NEAREST | CL_FP_INF_NAN.

9.5.10 Relative Error as ULPs

In this section we discuss the maximum relative error defined as *ulp* (units in the last place). If CL_FP_ROUND_TO_NEAREST is supported, the default rounding mode for half-precision floating-point operations will be round to nearest even; otherwise the default rounding mode will be round to zero. Addition, subtraction, multiplication, fused multiply-add operations on half types are required to be correctly rounded using the default rounding mode for half-precision floating-point operations. Conversions to half floating point format must be correctly rounded using the indicated `convert_` operator rounding mode or the default rounding mode for half-precision floating-point operations if no rounding mode is specified by the operator, or a C-style cast is used. Conversions from half to integer format shall correctly round using the indicated `convert_` operator rounding mode, or towards zero if no rounding mode is specified by the operator or a C-style cast is used. All conversions from half to floating point formats are exact.

The following table describes the minimum accuracy of half precision floating-point arithmetic operations given as ULP values. The reference value used to compute the ULP value of an arithmetic operation is the infinitely precise result.

Function	Min Accuracy - ULP values[7]
x + y	Correctly rounded
x – y	Correctly rounded
x * y	Correctly rounded
1.0 / *x*	Correctly rounded
x / y	Correctly rounded
acos	<= 2 ulp
acospi	<= 2 ulp
asin	<= 2 ulp
asinpi	<= 2 ulp
atan	<= 2 ulp
atan2	<= 2 ulp
atanpi	<= 2 ulp
atan2pi	<= 2 ulp
acosh	<= 2 ulp
asinh	<= 2 ulp
atanh	<= 2 ulp
cbrt	<= 2 ulp
ceil	Correctly rounded
copysign	0 ulp
cos	<= 2 ulp
cosh	<= 2 ulp
cospi	<= 2 ulp
erfc	<= 4 ulp
erf	<= 4 ulp
exp	<= 2 ulp
exp2	<= 2 ulp
exp10	<= 2 ulp
expm1	<= 2 ulp
fabs	0 ulp
fdim	Correctly rounded
floor	Correctly rounded
fma	Correctly rounded
fmax	0 ulp
fmin	0 ulp
fmod	0 ulp
fract	Correctly rounded
frexp	0 ulp

[7] 0 ulp is used for math functions that do not require rounding.

hypot	<= 2 ulp
ilogb	0 ulp
ldexp	Correctly rounded
log	<= 2 ulp
log2	<= 2 ulp
log10	<= 2 ulp
log1p	<= 2 ulp
logb	0 ulp
mad	Any value allowed (infinite ulp)
maxmag	0 ulp
minmag	0 ulp
modf	0 ulp
nan	0 ulp
nextafter	0 ulp
pow(x, y)	<= 4 ulp
pown(x, y)	<= 4 ulp
powr(x, y)	<= 4 ulp
remainder	0 ulp
remquo	0 ulp
rint	Correctly rounded
rootn	<= 4 ulp
round	Correctly rounded
rsqrt	<=1 ulp
sin	<= 2 ulp
sincos	<= 2 ulp for sine and cosine values
sinh	<= 2 ulp
sinpi	<= 2 ulp
sqrt	Correctly rounded
tan	<= 2 ulp
tanh	<= 2 ulp
tanpi	<= 2 ulp
tgamma	<= 4 ulp
trunc	Correctly rounded

NOTE: *Implementations may perform floating-point operations on* `half` *scalar or vector data types by converting the* `half` *values to single precision floating-point values and performing the operation in single precision floating-point. In this case, the implementation will use the* `half` *scalar or vector data type as a storage only format*.

9.6 Creating CL context from a GL context or share group

9.6.1 Overview

The OpenCL specification in *section 9.7* defines how to share data with texture and buffer objects in a parallel OpenGL implementation, but does not define how the association between an OpenCL context and an OpenGL context or share group is established. This extension defines optional attributes to OpenCL context creation routines which associate a GL context or share group object with a newly created OpenCL context. If this extension is supported by an implementation, the string **cl_khr_gl_sharing** will be present in the CL_PLATFORM_EXTENSIONS or CL_DEVICE_EXTENSIONS string described in *table 4.3*.

An OpenGL implementation supporting buffer objects and sharing of texture and buffer object images with OpenCL is required by this extension.

9.6.2 New Procedures and Functions

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*);

9.6.3 New Tokens

Returned by **clCreateContext**, **clCreateContextFromType**, and **clGetGLContextInfoKHR** when an invalid OpenGL context or share group object handle is specified in *properties*:

CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR	-1000

Accepted as the *param_name* argument of **clGetGLContextInfoKHR**:

CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR	0x2006
CL_DEVICES_FOR_GL_CONTEXT_KHR	0x2007

Accepted as an attribute name in the *properties* argument of **clCreateContext** and **clCreateContextFromType**:

CL_GL_CONTEXT_KHR	0x2008

CL_EGL_DISPLAY_KHR	0x2009
CL_GLX_DISPLAY_KHR	0x200A
CL_WGL_HDC_KHR	0x200B
CL_CGL_SHAREGROUP_KHR	0x200C

9.6.4 Additions to Chapter 4 of the OpenCL 1.2 Specification

In *section 4.4*, replace the description of *properties* under **clCreateContext** with:

"*properties* points to an attribute list, which is a array of ordered <attribute name, value> pairs terminated with zero. If an attribute is not specified in *properties*, then its default value (listed in *table 4.5*) is used (it is said to be specified implicitly). If *properties* is NULL or empty (points to a list whose first value is zero), all attributes take on their default values.

Attributes control sharing of OpenCL memory objects with OpenGL buffer, texture, and renderbuffer objects as described in *section 9.7*. Depending on the platform-specific API used to bind OpenGL contexts to the window system, the following attributes may be set to identify an OpenGL context:

- When the CGL binding API is supported, the attribute CL_CGL_SHAREGROUP_KHR should be set to a CGLShareGroup handle to a CGL share group object.
- When the EGL binding API is supported, the attribute CL_GL_CONTEXT_KHR should be set to an EGLContext handle to an OpenGL ES or OpenGL context, and the attribute CL_EGL_DISPLAY_KHR should be set to the EGLDisplay handle of the display used to create the OpenGL ES or OpenGL context.
- When the GLX binding API is supported, the attribute CL_GL_CONTEXT_KHR should be set to a GLXContext handle to an OpenGL context, and the attribute CL_GLX_DISPLAY_KHR should be set to the Display handle of the X Window System display used to create the OpenGL context.
- When the WGL binding API is supported, the attribute CL_GL_CONTEXT_KHR should be set to an HGLRC handle to an OpenGL context, and the attribute CL_WGL_HDC_KHR should be set to the HDC handle of the display used to create the OpenGL context.

Memory objects created in the context so specified may be shared with the specified OpenGL or OpenGL ES context (as well as with any other OpenGL contexts on the share list of that context, according to the description of sharing in the GLX 1.4 and EGL 1.4 specifications, and the WGL documentation for OpenGL implementations on Microsoft Windows), or with the explicitly identified OpenGL share group for CGL. If no OpenGL or OpenGL ES context or share group is specified in the attribute list, then memory objects may not be shared, and calling any of the commands in *section 9.7* will result in a CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR error."

OpenCL / OpenGL sharing does not support the CL_CONTEXT_INTEROP_USER_SYNC property defined in *table 4.5*. Specifying this property when creating a context with OpenCL / OpenGL sharing will return an appropriate error.

Add to *table 4.5*:

Attribute Name	Allowed Values (Default value is in bold)	Description
CL_GL_CONTEXT_KHR	**0**, OpenGL context handle	OpenGL context to associated the OpenCL context with
CL_CGL_SHAREGROUP_KHR	**0**, CGL share group handle	CGL share group to associate the OpenCL context with
CL_EGL_DISPLAY_KHR	**EGL_NO_DISPLAY**, EGLDisplay handle	EGLDisplay an OpenGL context was created with respect to
CL_GLX_DISPLAY_KHR	**None**, X handle	X Display an OpenGL context was created with respect to
CL_WGL_HDC_KHR	**0**, HDC handle	HDC an OpenGL context was created with respect to

Table 4.5: *Context creation attributes*

Replace the first error in the list for **clCreateContext** with:

"*errcode_ret* returns CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR if a context was specified by any of the following means:

- A context was specified for an EGL-based OpenGL ES or OpenGL implementation by setting the attributes CL_GL_CONTEXT_KHR and CL_EGL_DISPLAY_KHR.
- A context was specified for a GLX-based OpenGL implementation by setting the attributes CL_GL_CONTEXT_KHR and CL_GLX_DISPLAY_KHR.
- A context was specified for a WGL-based OpenGL implementation by setting the attributes CL_GL_CONTEXT_KHR and CL_WGL_HDC_KHR

and any of the following conditions hold:

- The specified display and context attributes do not identify a valid OpenGL or OpenGL ES context.
- The specified context does not support buffer and renderbuffer objects.

- The specified context is not compatible with the OpenCL context being created (for example, it exists in a physically distinct address space, such as another hardware device; or it does not support sharing data with OpenCL due to implementation restrictions).

errcode_ret returns CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR if a share group was specified for a CGL-based OpenGL implementation by setting the attribute CL_CGL_SHAREGROUP_KHR, and the specified share group does not identify a valid CGL share group object.

errcode_ret returns CL_INVALID_OPERATION if a context was specified as described above and any of the following conditions hold:

- A context or share group object was specified for one of CGL, EGL, GLX, or WGL and the OpenGL implementation does not support that window-system binding API.
- More than one of the attributes CL_CGL_SHAREGROUP_KHR, CL_EGL_DISPLAY_KHR, CL_GLX_DISPLAY_KHR, and CL_WGL_HDC_KHR is set to a non-default value.
- Both of the attributes CL_CGL_SHAREGROUP_KHR and CL_GL_CONTEXT_KHR are set to non-default values.
- Any of the devices specified in the *devices* argument cannot support OpenCL objects which share the data store of an OpenGL object, as described in *section 9.7*.

errcode_ret returns CL_INVALID_PROPERTY if an attribute name other than those specified in *table 4.5* or if CL_CONTEXT_INTEROP_USER_SYNC is specified in *properties*."

Replace the description of *properties* under **clCreateContextFromType** with:

"*properties* points to an attribute list whose format and valid contents are identical to the **properties** argument of **clCreateContext**."

Replace the first error in the list for **clCreateContextFromType** with the same two new errors described above for **clCreateContext**.

9.6.5 Additions to section 9.7 of the OpenCL 1.2 Extension Specification

Add new *section 9.7.7* :

"OpenCL device(s) corresponding to an OpenGL context may be queried. Such a device may not always exist (for example, if an OpenGL context is specified on a GPU not supporting OpenCL command queues, but which does support shared CL/GL objects), and if it does exist, may change over time. When such a device does exist, acquiring and releasing shared CL/GL objects may be faster on a command queue corresponding to this device than on command

queues corresponding to other devices available to an OpenCL context. To query the currently corresponding device, use the function

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*)

properties points to an attribute list whose format and valid contents are identical to the *properties* argument of **clCreateContext**. *properties* must identify a single valid GL context or GL share group object.

param_name is a constant that specifies the GL context information to query, and must be one of the values shown in *table 9.ctxprop*.

param_value is a pointer to memory where the result of the query is returned as described in *table 9.ctxprop*. If *param_value* is NULL, it is ignored.

param_value_size specifies the size in bytes of memory pointed to by *param_value*. This size must be greater than or equal to the size of the return type described in *table 9.ctxprop*.

param_value_size_ret returns the actual size in bytes of data being queried by *param_value*. If *param_value_size_ret* is NULL, it is ignored.

param_name	Return Type	Information returned in param_value
CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR	cl_device_id	Return the CL device currently associated with the specified OpenGL context.
CL_DEVICES_FOR_GL_CONTEXT_KHR	cl_device_id[]	List of all CL devices which may be associated with the specified OpenGL context.

Table 9.ctxprop: *GL context information that can be queried with **clGetGLContextInfoKHR***

clGetGLContextInfoKHR returns CL_SUCCESS if the function is executed successfully. If no device(s) exist corresponding to *param_name*, the call will not fail, but the value of *param_value_size_ret* will be zero.

clGetGLContextInfoKHR returns CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR if a context was specified by any of the following means:

- A context was specified for an EGL-based OpenGL ES or OpenGL implementation by setting the attributes CL_GL_CONTEXT_KHR and CL_EGL_DISPLAY_KHR.

- A context was specified for a GLX-based OpenGL implementation by setting the attributes CL_GL_CONTEXT_KHR and CL_GLX_DISPLAY_KHR.

- A context was specified for a WGL-based OpenGL implementation by setting the attributes CL_GL_CONTEXT_KHR and CL_WGL_HDC_KHR.

and any of the following conditions hold:

- The specified display and context attributes do not identify a valid OpenGL or OpenGL ES context.

- The specified context does not support buffer and renderbuffer objects.

- The specified context is not compatible with the OpenCL context being created (for example, it exists in a physically distinct address space, such as another hardware device; or it does not support sharing data with OpenCL due to implementation restrictions).

clGetGLContextInfoKHR returns CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR if a share group was specified for a CGL-based OpenGL implementation by setting the attribute CL_CGL_SHAREGROUP_KHR, and the specified share group does not identify a valid CGL share group object.

clGetGLContextInfoKHR returns CL_INVALID_OPERATION if a context was specified as described above and any of the following conditions hold:

- A context or share group object was specified for one of CGL, EGL, GLX, or WGL and the OpenGL implementation does not support that window-system binding API.

- More than one of the attributes CL_CGL_SHAREGROUP_KHR, CL_EGL_DISPLAY_KHR, CL_GLX_DISPLAY_KHR, and CL_WGL_HDC_KHR is set to a non-default value.

- Both of the attributes CL_CGL_SHAREGROUP_KHR and CL_GL_CONTEXT_KHR are set to non-default values.

- Any of the devices specified in the <devices> argument cannot support OpenCL objects which share the data store of an OpenGL object, as described in *section 9.7*.

clGetGLContextInfoKHR returns CL_INVALID_VALUE if an attribute name other than those specified in *table 4.5* is specified in *properties*.

Additionally, **clGetGLContextInfoKHR** returns CL_INVALID_VALUE if *param_name* is not one of the values listed in *table 9.ctxprop*, or if the size in bytes specified by *param_value_size* is less than the size of the return type shown in *table 9.ctxprop*, and *param_value* is not a NULL value, CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device, or CL_OUT_OF_HOST_MEMORY if there is a failure to

allocate resources required by the OpenCL implementation on the host."

9.6.6 Issues

1. How should the OpenGL context be identified when creating an associated OpenCL context?

RESOLVED: by using a (display,context handle) attribute pair to identify an arbitrary OpenGL or OpenGL ES context with respect to one of the window-system binding layers EGL, GLX, or WGL, or a share group handle to identify a CGL share group. If a context is specified, it need not be current to the thread calling clCreateContext*.

A previously suggested approach would use a single boolean attribute CL_USE_GL_CONTEXT_KHR to allow creating a context associated with the currently bound OpenGL context. This may still be implemented as a separate extension, and might allow more efficient acquire/release behavior in the special case where they are being executed in the same thread as the bound GL context used to create the CL context.

2. What should the format of an attribute list be?

After considerable discussion, we think we can live with a list of <attribute name,value> pairs terminated by zero. The list is passed as 'cl_context_properties **properties'*, where cl_context_properties is typedefed to be 'intptr_t' in cl.h.

This effectively allows encoding all scalar integer, pointer, and handle values in the host API into the argument list and is analogous to the structure and type of EGL attribute lists. NULL attribute lists are also allowed. Again as for EGL, any attributes not explicitly passed in the list will take on a defined default value that does something reasonable.

Experience with EGL, GLX, and WGL has shown attribute lists to be a sufficiently flexible and general mechanism to serve the needs of management calls such as context creation. It is not completely general (encoding floating-point and non-scalar attribute values is not straightforward), and other approaches were suggested such as opaque attribute lists with getter/setter methods, or arrays of varadic structures.

3. What's the behavior of an associated OpenGL or OpenCL context when using resources defined by the other associated context, and that context is destroyed?

RESOLVED: As described in *section 9.7*, OpenCL objects place a reference on the data store underlying the corresponding GL object when they're created. The GL name corresponding to that data store may be deleted, but the data store itself remains so long as any CL object has a reference to it. However, destroying all GL contexts in the share group corresponding to a CL context results in implementation-dependent behavior when using a corresponding CL object, up to and including program termination.

4. How about sharing with D3D?

Sharing between D3D and OpenCL should use the same attribute list mechanism, though obviously with different parameters, and be exposed as a similar parallel OpenCL extension. There may be an interaction between that extension and this one since it's not yet clear if it will be possible to create a CL context simultaneously sharing GL and D3D objects.

5. Under what conditions will context creation fail due to sharing?

RESOLVED: Several cross-platform failure conditions are described (GL context or CGL share group doesn't exist, GL context doesn't support types of GL objects required by the *section 9.7* interfaces, GL context implementation doesn't allow sharing), but additional failures may result due to implementation-dependent reasons and should be added to this extension as such failures are discovered. Sharing between OpenCL and OpenGL requires integration at the driver internals level.

6. What command queues can **clEnqueueAcquire/ReleaseGLObjects** be placed on?

RESOLVED: All command queues. This restriction is enforced at context creation time. If any device passed to context creation cannot support shared CL/GL objects, context creation will fail with a CL_INVALID_OPERATION error.

7. How can applications determine which command queue to place an Acquire/Release on?

RESOLVED: The **clGetGLContextInfoKHR** returns either the CL device currently corresponding to a specified GL context (typically the display it's running on), or a list of all the CL devices the specified context might run on (potentially useful in multiheaded / "virtual screen" environments). This command is not simply placed in *section 9.7* because it relies on the same property-list method of specifying a GL context introduced by this extension.

If no devices are returned, it means that the GL context exists on an older GPU not capable of running OpenCL, but still capable of sharing objects between GL running on that GPU and CL running elsewhere.

8. What is the meaning of the CL_DEVICES_FOR_GL_CONTEXT_KHR query?

RESOLVED: The list of all CL devices that may ever be associated with a specific GL context. On platforms such as MacOS X, the "virtual screen" concept allows multiple GPUs to back a single virtual display. Similar functionality might be implemented on other windowing systems, such as a transparent heterogenous multiheaded X server. Therefore the exact meaning of this query is interpreted relative to the binding layer API in use.

9) Miscellaneous issues during syncing of version 12 with the OpenCL 1.0 revision 47 spec language and the minor changes made including this extension as section 9.11 of that spec:

- Rev47 spec numbers table 9.ctxprop as "9.7" but this depends on the core spec revision.
- Rev47 spec uses 'cl_context' as the return type for **clGetGLContextInfoKHR** param

names, but cl_device_id / cl_device_id[] are the proper types.

- Rev47 spec omits the paragraph describing CL_SUCCESS return from **clGetGLContextInfoKHR**.

9.7 Sharing Memory Objects with OpenGL / OpenGL ES Buffer, Texture and Renderbuffer Objects

This section discusses OpenCL functions that allow applications to use OpenGL buffer, texture and renderbuffer objects as OpenCL memory objects. This allows efficient sharing of data between OpenCL and OpenGL. The OpenCL API may be used to execute kernels that read and/or write memory objects that are also OpenGL objects.

An OpenCL image object may be created from an OpenGL texture or renderbuffer object. An OpenCL buffer object may be created from an OpenGL buffer object.

OpenCL memory objects may be created from OpenGL objects if and only if the OpenCL context has been created from an OpenGL share group object or context. OpenGL share groups and contexts are created using platform specific APIs such as EGL, CGL, WGL, and GLX. On MacOS X, an OpenCL context may be created from an OpenGL share group object using the OpenCL platform extension **cl_apple_gl_sharing**. On other platforms including Microsoft Windows, Linux/Unix and others, an OpenCL context may be created from an OpenGL context using the Khronos platform extension **cl_khr_gl_sharing**. Refer to the platform documentation for your OpenCL implementation, or visit the Khronos Registry at http://www.khronos.org/registry/cl/ for more information.

Any supported OpenGL object defined within the GL share group object, or the share group associated with the GL context from which the CL context is created, may be shared, with the exception of the default OpenGL objects (i.e. objects named zero), which may not be shared.

9.7.1 Lifetime of Shared Objects

An OpenCL memory object created from an OpenGL object (hereinafter refered to as a “shared CL/GL object”) remains valid as long as the corresponding GL object has not been deleted. If the GL object is deleted through the GL API (e.g. **glDeleteBuffers**, **glDeleteTextures,** or **glDeleteRenderbuffers**), subsequent use of the CL buffer or image object will result in undefined behavior, including but not limited to possible CL errors and data corruption, but may not result in program termination.

The CL context and corresponding command-queues are dependent on the existence of the GL share group object, or the share group associated with the GL context from which the CL context is created. If the GL share group object or all GL contexts in the share group are destroyed, any use of the CL context or command-queue(s) will result in undefined behavior, which may include program termination. Applications should destroy the CL command-queue(s) and CL context before destroying the corresponding GL share group or contexts

9.7.2 CL Buffer Objects → GL Buffer Objects

The function

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

creates an OpenCL buffer object from an OpenGL buffer object.

context is a valid OpenCL context created from an OpenGL context.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of *flags*. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* can be used.

bufobj is the name of a GL buffer object. The data store of the GL buffer object must have have been previously created by calling **glBufferData**, although its contents need not be initialized. The size of the data store will be used to determine the size of the CL buffer object.

errcode_ret will return an appropriate error code as described below. If *errcode_ret* is NULL, no error code is returned.

clCreateFromGLBuffer returns a valid non-zero OpenCL buffer object and *errcode_ret* is set to CL_SUCCESS if the buffer object is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context or was not created from a GL context.
- CL_INVALID_VALUE if values specified in *flags* are not valid.
- CL_INVALID_GL_OBJECT if *bufobj* is not a GL buffer object or is a GL buffer object but does not have an existing data store or the size of the buffer is 0.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The size of the GL buffer object data store at the time **clCreateFromGLBuffer** is called will be used as the size of buffer object returned by **clCreateFromGLBuffer**. If the state of a GL buffer object is modified through the GL API (e.g. **glBufferData**) while there exists a corresponding CL buffer object, subsequent use of the CL buffer object will result in undefined behavior.

The **clRetainMemObject** and **clReleaseMemObject** functions can be used to retain and release the buffer object.

The CL buffer object created using clCreateFromGLBuffer can also be used to create a CL 1D image buffer object.

9.7.3 CL Image Objects → GL Textures

The function

cl_mem **clCreateFromGLTexture** (cl_context *context*,
cl_mem_flags *flags*,
GLenum *texture_target*,
GLint *miplevel*,
GLuint *texture*,
cl_int **errcode_ret*)

creates the following:

- an OpenCL 2D image object from an OpenGL 2D texture object or a single face of an OpenGL cubemap texture object,
- an OpenCL 2D image array object from an OpenGL 2D texture array object,
- an OpenCL 1D image object from an OpenGL 1D texture object,
- an OpenCL 1D image buffer object from an OpenGL texture buffer object,
- an OpenCL 1D image array object from an OpenGL 1D texture array object,
- an OpenCL 3D image object from an OpenGL 3D texture object.

context is a valid OpenCL context created from an OpenGL context.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of *flags*. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* may be used.

texture_target must be one of GL_TEXTURE_1D, GL_TEXTURE_1D_ARRAY, GL_TEXTURE_BUFFER, GL_TEXTURE_2D, GL_TEXTURE_2D_ARRAY, GL_TEXTURE_3D, GL_TEXTURE_CUBE_MAP_POSITIVE_X, GL_TEXTURE_CUBE_MAP_POSITIVE_Y, GL_TEXTURE_CUBE_MAP_POSITIVE_Z, GL_TEXTURE_CUBE_MAP_NEGATIVE_X, GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, or

GL_TEXTURE_RECTANGLE[8]. *texture_target* is used only to define the image type of *texture*. No reference to a bound GL texture object is made or implied by this parameter.

miplevel is the mipmap level to be used[9]. If *texture_target* is GL_TEXTURE_BUFFER, *miplevel* must be 0.

texture is the name of a GL 1D, 2D, 3D, 1D array, 2D array, cubemap, rectangle or buffer texture object. The texture object must be a complete texture as per OpenGL rules on texture completeness. The *texture* format and dimensions defined by OpenGL for the specified *miplevel* of the texture will be used to create the OpenCL image memory object. Only GL texture objects with an internal format that maps to appropriate image channel order and data type specified in *tables 5.5* and *5.6* may be used to create the OpenCL image memory object.

errcode_ret will return an appropriate error code as described below. If *errcode_ret* is NULL, no error code is returned.

clCreateFromGLTexture returns a valid non-zero OpenCL image object and *errcode_ret* is set to CL_SUCCESS if the image object is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context or was not created from a GL context.
- CL_INVALID_VALUE if values specified in *flags* are not valid or if value specified in *texture_target* is not one of the values specified in the description of *texture_target*.
- CL_INVALID_MIP_LEVEL if *miplevel* is less than the value of $level_{base}$ (for OpenGL implementations) or zero (for OpenGL ES implementations); or greater than the value of *q* (for both OpenGL and OpenGL ES). $level_{base}$ and *q* are defined for the texture in *section 3.8.10* (Texture Completeness) of the OpenGL 2.1 specification and *section 3.7.10* of the OpenGL ES 2.0.
- CL_INVALID_MIP_LEVEL if *miplevel* is greather than zero and the OpenGL implementation does not support creating from non-zero mipmap levels.
- CL_INVALID_GL_OBJECT if *texture* is not a GL texture object whose type matches *texture_target*, if the specified *miplevel* of *texture* is not defined, or if the width or height of the specified *miplevel* is zero.
- CL_INVALID_IMAGE_FORMAT_DESCRIPTOR if the OpenGL texture internal format does not map to a supported OpenCL image format.

[8] Requires OpenGL 3.1. Alternatively, GL_TEXTURE_RECTANGLE_ARB may be specified if the OpenGL extension **GL_ARB_texture_rectangle** is supported.

[9] Implementations may return CL_INVALID_OPERATION for miplevel values > 0.

- CL_INVALID_OPERATION if *texture* is a GL texture object created with a border width value greater than zero.

- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.

- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

If the state of a GL texture object is modified through the GL API (e.g. **glTexImage2D**, **glTexImage3D** or the values of the texture parameters GL_TEXTURE_BASE_LEVEL or GL_TEXTURE_MAX_LEVEL are modified) while there exists a corresponding CL image object, subsequent use of the CL image object will result in undefined behavior.

The **clRetainMemObject** and **clReleaseMemObject** functions can be used to retain and release the image objects.

9.7.3.1 List of OpenGL and corresponding OpenCL Image Formats

Table 9.4 describes the list of GL texture internal formats and the corresponding CL image formats. If a GL texture object with an internal format from *table 9.4* is successfully created by OpenGL, then there is guaranteed to be a mapping to one of the corresponding CL image format(s) in that table. Texture objects created with other OpenGL internal formats may (but are not guaranteed to) have a mapping to a CL image format; if such mappings exist, they are guaranteed to preserve all color components, data types, and at least the number of bits/component actually allocated by OpenGL for that format.

GL internal format	CL image format (channel order, channel data type)
GL_RGBA8	CL_RGBA, CL_UNORM_INT8 or CL_BGRA, CL_UNORM_INT8
GL_RGBA, GL_UNSIGNED_INT_8_8_8_8_REV	CL_RGBA, CL_UNORM_INT8
GL_BGRA, GL_UNSIGNED_INT_8_8_8_8_REV	CL_BGRA, CL_UNORM_INT8
GL_RGBA16	CL_RGBA, CL_UNORM_INT16
GL_RGBA8I, GL_RGBA8I_EXT	CL_RGBA, CL_SIGNED_INT8
GL_RGBA16I, GL_RGBA16I_EXT	CL_RGBA, CL_SIGNED_INT16
GL_RGBA32I, GL_RGBA32I_EXT	CL_RGBA, CL_SIGNED_INT32
GL_RGBA8UI, GL_RGBA8UI_EXT	CL_RGBA, CL_UNSIGNED_INT8
GL_RGBA16UI, GL_RGBA16UI_EXT	CL_RGBA, CL_UNSIGNED_INT16
GL_RGBA32UI, GL_RGBA32UI_EXT	CL_RGBA, CL_UNSIGNED_INT32

GL_RGBA16F, GL_RGBA16F_ARB	CL_RGBA, CL_HALF_FLOAT
GL_RGBA32F, GL_RGBA32F_ARB	CL_RGBA, CL_FLOAT

Table 9.4 *Mapping of GL internal format to CL image format*

9.7.4 CL Image Objects → GL Renderbuffers

The function

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

creates an OpenCL 2D image object from an OpenGL renderbuffer object.

context is a valid OpenCL context created from an OpenGL context.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of *flags*. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* can be used.

renderbuffer is the name of a GL renderbuffer object. The renderbuffer storage must be specified before the image object can be created. The *renderbuffer* format and dimensions defined by OpenGL will be used to create the 2D image object. Only GL renderbuffers with internal formats that maps to appropriate image channel order and data type specified in *tables 5.5* and *5.6* can be used to create the 2D image object.

errcode_ret will return an appropriate error code as described below. If *errcode_ret* is NULL, no error code is returned.

clCreateFromGLRenderbuffer returns a valid non-zero OpenCL image object and *errcode_ret* is set to CL_SUCCESS if the image object is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context or was not created from a GL context.
- CL_INVALID_VALUE if values specified in *flags* are not valid.
- CL_INVALID_GL_OBJECT if *renderbuffer* is not a GL renderbuffer object or if the width or height of *renderbuffer* is zero.

- CL_INVALID_IMAGE_FORMAT_DESCRIPTOR if the OpenGL renderbuffer internal format does not map to a supported OpenCL image format.
- CL_INVALID_OPERATION if *renderbuffer* is a multi-sample GL renderbuffer object.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

If the state of a GL renderbuffer object is modified through the GL API (i.e. changes to the dimensions or format used to represent pixels of the GL renderbuffer using appropriate GL API calls such as **glRenderbufferStorage**) while there exists a corresponding CL image object, subsequent use of the CL image object will result in undefined behavior.

The **clRetainMemObject** and **clReleaseMemObject** functions can be used to retain and release the image objects.

Table 9.4 describes the list of GL renderbuffer internal formats and the corresponding CL image formats. If a GL renderbuffer object with an internal format from *table 9.4* is successfully created by OpenGL, then there is guaranteed to be a mapping to one of the corresponding CL image format(s) in that table. Renderbuffer objects created with other OpenGL internal formats may (but are not guaranteed to) have a mapping to a CL image format; if such mappings exist, they are guaranteed to preserve all color components, data types, and at least the number of bits/component actually allocated by OpenGL for that format.

9.7.5 Querying GL object information from a CL memory object

The OpenGL object used to create the OpenCL memory object and information about the object type i.e. whether it is a texture, renderbuffer or buffer object can be queried using the following function.

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

gl_object_type returns the type of GL object attached to *memobj* and can be CL_GL_OBJECT_BUFFER, CL_GL_OBJECT_TEXTURE2D, CL_GL_OBJECT_TEXTURE3D, CL_GL_OBJECT_TEXTURE2D_ARRAY, CL_GL_OBJECT_TEXTURE1D, CL_GL_OBJECT_TEXTURE1D_ARRAY, CL_GL_OBJECT_TEXTURE_BUFFER, or CL_GL_OBJECT_RENDERBUFFER. If *gl_object_type* is NULL, it is ignored

gl_object_name returns the GL object name used to create *memobj*. If *gl_object_name* is NULL, it is ignored.

clGetGLObjectInfo returns CL_SUCCESS if the call was executed successfully. Otherwise, it returns one of the following errors:

- CL_INVALID_MEM_OBJECT if *memobj* is not a valid OpenCL memory object.
- CL_INVALID_GL_OBJECT if there is no GL object associated with *memobj*.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

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*)

returns additional information about the GL texture object associated with *memobj*.

param_name specifies what additional information about the GL texture object associated with *memobj* to query. The list of supported *param_name* types and the information returned in *param_value* by **clGetGLTextureInfo** is described in *table 9.5* below.

param_value is a pointer to memory where the result being queried is returned. If *param_value* is NULL, it is ignored.

param_value_size is used to specify the size in bytes of memory pointed to by *param_value*. This size must be >= size of return type as described in *table 9.5* below.

param_value_size_ret returns the actual size in bytes of data copied to *param_value*. If *param_value_size_ret* is NULL, it is ignored.

cl_gl_texture_info	Return Type	Info. returned in *param_value*
CL_GL_TEXTURE_TARGET	GLenum	The *texture_target* argument specified in **clCreateFromGLTexture**.
CL_GL_MIPMAP_LEVEL	GLint	The *miplevel* argument specified in

		clCreateFromGLTexture.

Table *9.5* *List of supported param_names by clGetGLTextureInfo*

clGetGLTextureInfo returns CL_SUCCESS if the function is executed successfully. Otherwise, it returns one of the following errors:

- CL_INVALID_MEM_OBJECT if *memobj* is not a valid OpenCL memory object.
- CL_INVALID_GL_OBJECT if there is no GL texture object associated with *memobj*.
- CL_INVALID_VALUE if *param_name* is not valid, or if size in bytes specified by *param_value_size* is < size of return type as described in *table 9.5* and *param_value* is not NULL, or if *param_value* and *param_value_size_ret* are NULL.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

9.7.6 Sharing memory objects that map to GL objects between GL and CL contexts

The function

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*)

is used to acquire OpenCL memory objects that have been created from OpenGL objects. These objects need to be acquired before they can be used by any OpenCL commands queued to a command-queue. The OpenGL objects are acquired by the OpenCL context associated with *command_queue* and can therefore be used by all command-queues associated with the OpenCL context.

command_queue is a valid command-queue. All devices used to create the OpenCL context associated with *command_queue* must support acquiring shared CL/GL objects. This constraint is enforced at context creation time.

num_objects is the number of memory objects to be acquired in *mem_objects*.

mem_objects is a pointer to a list of CL memory objects that correspond to GL objects.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is NULL, then this particular command does not wait on any event to complete. If *event_wait_list* is NULL, *num_events_in_wait_list* must be 0. If *event_wait_list* is not NULL, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points.

event returns an event object that identifies this command and can be used to query or queue a wait for the command to complete. *event* can be NULL in which case it will not be possible for the application to query the status of this command or queue a wait for this command to complete. If the *event_wait_list* and the *event* arguments are not NULL, the *event* argument should not refer to an element of the *event_wait_list* array.

clEnqueueAcquireGLObjects returns CL_SUCCESS if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is NULL the function does nothing and returns CL_SUCCESS. Otherwise, it returns one of the following errors:

- CL_INVALID_VALUE if *num_objects* is zero and *mem_objects* is not a NULL value or if *num_objects* > 0 and *mem_objects* is NULL.
- CL_INVALID_MEM_OBJECT if memory objects in *mem_objects* are not valid OpenCL memory objects.
- CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue.
- CL_INVALID_CONTEXT if context associated with *command_queue* was not created from an OpenGL context
- CL_INVALID_GL_OBJECT if memory objects in *mem_objects* have not been created from a GL object(s).
- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is NULL and *num_events_in_wait_list* > 0, or *event_wait_list* is not NULL and *num_events_in_wait_list* is 0, or if event objects in *event_wait_list* are not valid events.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

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*)

is used to release OpenCL memory objects that have been created from OpenGL objects. These objects need to be released before they can be used by OpenGL. The OpenGL objects are released by the OpenCL context associated with *command_queue*.

num_objects is the number of memory objects to be released in *mem_objects*.

mem_objects is a pointer to a list of CL memory objects that correpond to GL objects.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this command can be executed. If *event_wait_list* is NULL, then this particular command does not wait on any event to complete. If *event_wait_list* is NULL, *num_events_in_wait_list* must be 0. If *event_wait_list* is not NULL, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points.

event returns an event object that identifies this particular read / write command and can be used to query or queue a wait for the command to complete. *event* can be NULL in which case it will not be possible for the application to query the status of this command or queue a wait for this command to complete. If the *event_wait_list* and the *event* arguments are not NULL, the *event* argument should not refer to an element of the *event_wait_list* array.

clEnqueueReleaseGLObjects returns CL_SUCCESS if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is NULL the function does nothing and returns CL_SUCCESS. Otherwise, it returns one of the following errors:

- CL_INVALID_VALUE if *num_objects* is zero and *mem_objects* is not a NULL value or if *num_objects* > 0 and *mem_objects* is NULL.
- CL_INVALID_MEM_OBJECT if memory objects in *mem_objects* are not valid OpenCL memory objects.
- CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue.
- CL_INVALID_CONTEXT if context associated with *command_queue* was not created from an OpenGL context

- CL_INVALID_GL_OBJECT if memory objects in *mem_objects* have not been created from a GL object(s).

- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is NULL and *num_events_in_wait_list* > 0, or *event_wait_list* is not NULL and *num_events_in_wait_list* is 0, or if event objects in *event_wait_list* are not valid events.

- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.

- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

9.7.6.1 Synchronizing OpenCL and OpenGL Access to Shared Objects

In order to ensure data integrity, the application is responsible for synchronizing access to shared CL/GL objects by their respective APIs. Failure to provide such synchronization may result in race conditions and other undefined behavior including non-portability between implementations.

Prior to calling **clEnqueueAcquireGLObjects**, the application must ensure that any pending GL operations which access the objects specified in *mem_objects* have completed. This may be accomplished portably by issuing and waiting for completion of a **glFinish** command on all GL contexts with pending references to these objects. Implementations may offer more efficient synchronization methods; for example on some platforms calling **glFlush** may be sufficient, or synchronization may be implicit within a thread, or there may be vendor-specific extensions that enable placing a fence in the GL command stream and waiting for completion of that fence in the CL command queue. Note that no synchronization methods other than **glFinish** are portable between OpenGL implementations at this time.

Similarly, after calling **clEnqueueReleaseGLObjects**, the application is responsible for ensuring that any pending OpenCL operations which access the objects specified in *mem_objects* have completed prior to executing subsequent GL commands which reference these objects. This may be accomplished portably by calling **clWaitForEvents** with the event object returned by **clEnqueueReleaseGLObjects,** or by calling **clFinish**. As above, some implementations may offer more efficient methods.

The application is responsible for maintaining the proper order of operations if the CL and GL contexts are in separate threads.

If a GL context is bound to a thread other than the one in which **clEnqueueReleaseGLObjects** is called, changes to any of the objects in *mem_objects* may not be visible to that context without additional steps being taken by the application. For an OpenGL 3.1 (or later) context, the requirements are described in Appendix D ("Shared Objects and Multiple Contexts") of the

OpenGL 3.1 Specification. For prior versions of OpenGL, the requirements are implementation-dependent.

Attempting to access the data store of an OpenGL object after it has been acquired by OpenCL and before it has been released will result in undefined behavior. Similarly, attempting to access a shared CL/GL object from OpenCL before it has been acquired by the OpenCL command queue, or after it has been released, will result in undefined behavior.

9.8 Creating CL event objects from GL sync objects

9.8.1 Overview

This extension allows creating OpenCL event objects linked to OpenGL fence sync objects, potentially improving efficiency of sharing images and buffers between the two APIs. The companion **GL_ARB_cl_event** extension provides the complementary functionality of creating an OpenGL sync object from an OpenCL event object.

In addition, this extension modifies the behavior of **clEnqueueAcquireGLObjects** and **clEnqueueReleaseGLObjects** to implicitly guarantee synchronization with an OpenGL context bound in the same thread as the OpenCL context.

If this extension is supported by an implementation, the string **cl_khr_gl_event** will be present in the CL_PLATFORM_EXTENSIONS or CL_DEVICE_EXTENSIONS string described in *table 4.3*.

9.8.2 New Procedures and Functions

cl_event **clCreateEventFromGLsyncKHR** (cl_context *context*,
GLsync *sync*,
cl_int **errcode_ret*);

9.8.3 New Tokens

Returned by **clGetEventInfo** when *param_name* is CL_EVENT_COMMAND_TYPE:

CL_COMMAND_GL_FENCE_SYNC_OBJECT_KHR 0x200D

9.8.4 Additions to Chapter 5 of the OpenCL 1.2 Specification

Add following to the fourth paragraph of *section 5.9* (prior to the description of **clWaitForEvents**):

"Event objects can also be used to reflect the status of an OpenGL sync object. The sync object in turn refers to a fence command executing in an OpenGL command stream. This provides another method of coordinating sharing of buffers and images between OpenGL and OpenCL (see *section 9.7.6.1*)."

Add CL_COMMAND_GL_FENCE_SYNC_OBJECT_KHR to the valid *param_value* values returned by **clGetEventInfo** for *param_name* CL_EVENT_COMMAND_TYPE (in the second row and third column of *table 5.18*).

Add new *subsection 5.9.1*:

"**5.9.1 Linking Event Objects to OpenGL Synchronization Objects**

An event object may be created by linking to an OpenGL **sync object**. Completion of such an event object is equivalent to waiting for completion of the fence command associated with the linked GL sync object.

The function

cl_event **clCreateEventFromGLsyncKHR** (cl_context *context*,
GLsync *sync*,
cl_int **errcode_ret*)

creates a linked event object.

context is a valid OpenCL context created from an OpenGL context or share group, using the **cl_khr_gl_sharing** extension.

sync is the name of a sync object in the GL share group associated with *context*.

clCreateEventFromGLsyncKHR returns a valid OpenCL event object and *errcode_ret* is set to CL_SUCCESS if the event object is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context, or was not created from a GL context.
- CL_INVALID_GL_OBJECT if *sync* is not the name of a sync object in the GL share group associated with *context*.

The parameters of an event object linked to a GL sync object will return the following values when queried with **clGetEventInfo**:

- The CL_EVENT_COMMAND_QUEUE of a linked event is NULL, because the event is not associated with any OpenCL command queue.
- The CL_EVENT_COMMAND_TYPE of a linked event is CL_COMMAND_GL_FENCE_SYNC_OBJECT_KHR, indicating that the event is associated with a GL sync object, rather than an OpenCL command.
- The CL_EVENT_COMMAND_EXECUTION_STATUS of a linked event is either

CL_SUBMITTED, indicating that the fence command associated with the sync object has not yet completed, or CL_COMPLETE, indicating that the fence command has completed.

clCreateEventFromGLsyncKHR performs an implicit **clRetainEvent** on the returned event object. Creating a linked event object also places a reference on the linked GL sync object. When the event object is deleted, the reference will be removed from the GL sync object.

Events returned from **clCreateEventFromGLsyncKHR** may only be consumed by **clEnqueueAcquireGLObjects**. Passing such events to any other CL API will generate a CL_INVALID_EVENT error."

9.8.5 Additions to Chapter 9 of the OpenCL 1.2 Specification

Add following the paragraph describing parameter *event* to **clEnqueueAcquireGLObjects**:

"If an OpenGL context is bound to the current thread, then any OpenGL commands which

1. affect or access the contents of a memory object listed in the *mem_objects* list, and
2. were issued on that OpenGL context prior to the call to **clEnqueueAcquireGLObjects**

will complete before execution of any OpenCL commands following the **clEnqueueAcquireGLObjects** which affect or access any of those memory objects. If a non-NULL *event* object is returned, it will report completion only after completion of such OpenGL commands."

Add following the paragraph describing parameter *event* to **clEnqueueReleaseGLObjects**:

"If an OpenGL context is bound to the current thread, then then any OpenGL commands which

1. affect or access the contents of the memory objects listed in the *mem_objects* list, and
2. are issued on that context after the call to **clEnqueueReleaseGLObjects**

will not execute until after execution of any OpenCL commands preceding the **clEnqueueReleaseGLObjects** which affect or access any of those memory objects. If a non-NULL *event* object is returned, it will report completion before execution of such OpenGL commands."

Replace the second paragraph of *section 9.7.6.1* (Synchronizing OpenCL and OpenGL Access to Shared Objects) with:

"Prior to calling **clEnqueueAcquireGLObjects**, the application must ensure that any pending OpenGL operations which access the objects specified in *mem_objects* have completed.

If the **cl_khr_gl_event** extension is supported, then the OpenCL implementation will ensure that any such pending OpenGL operations are complete for an OpenGL context bound to the same

thread as the OpenCL context. This is referred to as *implicit synchronization*.

If the **cl_khr_gl_event** extension is supported and the OpenGL context in question supports fence sync objects, completion of OpenGL commands may also be determined by placing a GL fence command after those commands using **glFenceSync**, creating an event from the resulting GL sync object using **clCreateEventFromGLsyncKHR**, and determining completion of that event object via **clEnqueueAcquireGLObjects**. This method may be considerably more efficient than calling **glFinish**, and is referred to as *explicit synchronization*. Explicit synchronization is most useful when an OpenGL context bound to another thread is accessing the memory objects.

If the **cl_khr_gl_event** extension is not supported, completion of OpenGL commands may be determined by issuing and waiting for completion of a **glFinish** command on all OpenGL contexts with pending references to these objects. Some implementations may offer other efficient synchronization methods. If such methods exist they will be described in platform-specific documentation.

Note that no synchronization method other than **glFinish** is portable between all OpenGL implementations and all OpenCL implementations. While this is the only way to ensure completion that is portable to all platforms, **glFinish** is an expensive operation and its use should be avoided if the **cl_khr_gl_event** extension is supported on a platform."

9.8.6 Issues

1) How are references between CL events and GL syncs handled?

PROPOSED: The linked CL event places a single reference on the GL sync object. That reference is removed when the CL event is deleted. A more expensive alternative would be to reflect changes in the CL event reference count through to the GL sync.

2) How are linkages to synchronization primitives in other APIs handled?

UNRESOLVED. We will at least want to have a way to link events to EGL sync objects. There is probably no analogous DX concept. There would be an entry point for each type of synchronization primitive to be linked to, such as clCreateEventFromEGLSyncKHR.

An alternative is a generic clCreateEventFromExternalEvent taking an attribute list. The attribute list would include information defining the type of the external primitive and additional information (GL sync object handle, EGL display and sync object handle, etc.) specific to that type. This allows a single entry point to be reused.

These will probably be separate extensions following the API proposed here.

3) Should the CL_EVENT_COMMAND_TYPE correspond to the type of command (fence) or the type of the linked sync object?

PROPOSED: To the type of the linked sync object.

4) Should we support both explicit and implicit synchronization?

PROPOSED: Yes. Implicit synchronization is suitable when GL and CL are executing in the same application thread. Explicit synchronization is suitable when they are executing in different threads but the expense of glFinish is too high.

5) Should this be a platform or device extension?

PROPOSED: Platform extension. This may result in considerable under-the-hood work to implement the sync->event semantics using only the public GL API, however, when multiple drivers and devices with different GL support levels coexist in the same runtime.

6) Where can events generated from GL syncs be usable?

PROPOSED: Only with clEnqueueAcquireGLObjects, and attempting to use such an event elsewhere will generate an error. There is no apparent use case for using such events elsewhere, and possibly some cost to supporting it, balanced by the cost of checking the source of events in all other commands accepting them as parameters.

9.9 Sharing Memory Objects with Direct3D 10

9.9.1 Overview

The goal of this extension is to provide interoperability between OpenCL and Direct3D 10. This is designed to function analogously to the OpenGL interoperability as defined in *sections 9.7* and *9.8*. If this extension is supported by an implementation, the string **cl_khr_d3d10_sharing** will be present in the CL_PLATFORM_EXTENSIONS or CL_DEVICE_EXTENSIONS string described in *table 4.3*.

9.9.2 Header File

As currently proposed the interfaces for this extension would be provided in `cl_d3d10.h`.

9.9.3 New Procedures and Functions

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*)

cl_mem **clCreateFromD3D10BufferKHR** (cl_context *context*,
cl_mem_flags *flags*,
ID3D10Buffer **resource*,
cl_int **errcode_ret*)

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

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

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*)

9.9.4 New Tokens

Accepted as a Direct3D 10 device source in the *d3d_device_source* parameter of **clGetDeviceIDsFromD3D10KHR**:

CL_D3D10_DEVICE_KHR	0x4010
CL_D3D10_DXGI_ADAPTER_KHR	0x4011

Accepted as a set of Direct3D 10 devices in the *d3d_device_set* parameter of **clGetDeviceIDsFromD3D10KHR**:

CL_PREFERRED_DEVICES_FOR_D3D10_KHR	0x4012
CL_ALL_DEVICES_FOR_D3D10_KHR	0x4013

Accepted as a property name in the *properties* parameter of **clCreateContext** and **clCreateContextFromType**:

CL_CONTEXT_D3D10_DEVICE_KHR	0x4014

Accepted as a property name in the *param_name* parameter of **clGetContextInfo**:

CL_CONTEXT_D3D10_PREFER_SHARED_RESOURCES_KHR	0x402C

Accepted as the property being queried in the *param_name* parameter of **clGetMemObjectInfo**:

CL_MEM_D3D10_RESOURCE_KHR	0x4015

Accepted as the property being queried in the *param_name* parameter of **clGetImageInfo**:

CL_IMAGE_D3D10_SUBRESOURCE_KHR 0x4016

Returned in the *param_value* parameter of **clGetEventInfo** when *param_name* is CL_EVENT_COMMAND_TYPE:

CL_COMMAND_ACQUIRE_D3D10_OBJECTS_KHR 0x4017
CL_COMMAND_RELEASE_D3D10_OBJECTS_KHR 0x4018

Returned by **clCreateContext** and **clCreateContextFromType** if the Direct3D 10 device specified for interoperability is not compatible with the devices against which the context is to be created:

CL_INVALID_D3D10_DEVICE_KHR -1002

Returned by **clCreateFromD3D10BufferKHR** when *resource* is not a Direct3D 10 buffer object, and by **clCreateFromD3D10Texture2DKHR** and **clCreateFromD3D10Texture3DKHR** when *resource* is not a Direct3D 10 texture object.

CL_INVALID_D3D10_RESOURCE_KHR -1003

Returned by **clEnqueueAcquireD3D10ObjectsKHR** when any of *mem_objects* are currently acquired by OpenCL

CL_D3D10_RESOURCE_ALREADY_ACQUIRED_KHR -1004

Returned by **clEnqueueReleaseD3D10ObjectsKHR** when any of *mem_objects* are not currently acquired by OpenCL

CL_D3D10_RESOURCE_NOT_ACQUIRED_KHR -1005

9.9.5 Additions to Chapter 4 of the OpenCL 1.2 Specification

In *section 4.4*, replace the description of *properties* under **clCreateContext** with:

"*properties* specifies a list of context property names and their corresponding values. Each property is followed immediately by the corresponding desired value. The list is terminated with zero. If a property is not specified in *properties*, then its default value (listed in *table 4.5*) is used (it is said to be specified implicitly). If *properties* is NULL or empty (points to a list whose first value is zero), all attributes take on their default values."

Add the following to *table 4.5*:

cl_context_properties enum	Property value	Description
CL_CONTEXT_D3D10_DEVICE_KHR	ID3D10Device *	Specifies the ID3D10Device *

		to use for Direct3D 10 interoperability. The default value is NULL.

Add to the list of errors for **clCreateContext**:

- CL_INVALID_D3D10_DEVICE_KHR if the value of the property CL_CONTEXT_D3D10_DEVICE_KHR is non-NULL and does not specify a valid Direct3D 10 device with which the *cl_device_ids* against which this context is to be created may interoperate.

- CL_INVALID_OPERATION if Direct3D 10 interoperability is specified by setting CL_INVALID_D3D10_DEVICE_KHR to a non-NULL value, and interoperability with another graphics API is also specified."

Add to the list of errors for **clCreateContextFromType** the same new errors described above for **clCreateContext**.

Add the following row to *table 4.7*:

cl_context_info	Return Type	Information returned in param_value
CL_CONTEXT_D3D10_PREFER_SHARED_RESOURCES_KHR	cl_bool	Returns CL_TRUE if Direct3D 10 resources created as shared by setting *MiscFlags* to include D3D10_RESOURCE_MISC_SHARED will perform faster when shared with OpenCL, compared with resources which have not set this flag. Otherwise returns CL_FALSE.

9.9.6 Additions to Chapter 5 of the OpenCL 1.2 Specification

Add to the list of errors for **clGetMemObjectInfo**:

- CL_INVALID_D3D10_RESOURCE_KHR if *param_name* is CL_MEM_D3D10_RESOURCE_KHR and *memobj* was not created by the function **clCreateFromD3D10BufferKHR**, **clCreateFromD3D10Texture2DKHR**, or **clCreateFromD3D10Texture3DKHR**."

Extend *table 5.11* to include the following entry.

cl_mem_info	Return type	Info. returned in *param_value*
CL_MEM_D3D10_ RESOURCE_KHR	ID3D10Resource *	If *memobj* was created using **clCreateFromD3D10BufferKHR**, **clCreateFromD3D10Texture2DKHR**, or **clCreateFromD3D10Texture3DKHR**, returns the *resource* argument specified when *memobj* was created.

Add to the list of errors for **clGetImageInfo**:

- CL_INVALID_D3D10_RESOURCE_KHR if *param_name* is CL_MEM_D3D10_SUBRESOURCE_KHR and *image* was not created by the function **clCreateFromD3D10Texture2DKHR**, or **clCreateFromD3D10Texture3DKHR**."

Extend *table 5.9* to include the following entry.

cl_image_info	Return type	Info. returned in *param_value*
CL_MEM_D3D10_ SUBRESOURCE_KHR	ID3D10Resource *	If *image* was created using **clCreateFromD3D10Texture2DKHR**, or **clCreateFromD3D10Texture3DKHR**, returns the *subresource* argument specified when *image* was created.

Add to *table 5.18* in the **Info returned in <param_value>** column for *cl_event_info* = CL_EVENT_COMMAND_TYPE:

CL_COMMAND_ACQUIRE_D3D10_OBJECTS_KHR
CL_COMMAND_RELEASE_D3D10_OBJECTS_KHR

9.9.7 Sharing Memory Objects with Direct3D 10 Resources

This section discusses OpenCL functions that allow applications to use Direct3D 10 resources as OpenCL memory objects. This allows efficient sharing of data between OpenCL and Direct3D 10. The OpenCL API may be used to execute kernels that read and/or write memory objects that are also Direct3D 10 resources. An OpenCL image object may be created from a Direct3D 10 texture resource. An OpenCL buffer object may be created from a Direct3D 10 buffer resource. OpenCL memory objects may be created from Direct3D 10 objects if and only if the OpenCL context has been created from a Direct3D 10 device.

9.9.7.1 Querying OpenCL Devices Corresponding to Direct3D 10 Devices

The OpenCL devices corresponding to a Direct3D 10 device may be queried. The OpenCL devices corresponding to a DXGI adapter may also be queried. The OpenCL devices corresponding to a Direct3D 10 device will be a subset of the OpenCL devices corresponding to the DXGI adapter against which the Direct3D 10 device was created.

The OpenCL devices corresponding to a Direct3D 10 device or a DXGI device may be queried using the function

```
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)
```

platform refers to the platform ID returned by **clGetPlatformIDs**.

d3d_device_source specifies the type of *d3d_object*, and must be one of the values shown in *table 9.9.1*.

d3d_object specifies the object whose corresponding OpenCL devices are being queried. The type of *d3d_object* must be as specified in *table 9.9.1*.

d3d_device_set specifies the set of devices to return, and must be one of the values shown in *table 9.9.2*.

num_entries is the number of cl_device_id entries that can be added to *devices*. If *devices* is not NULL then *num_entries* must be greater than zero.

devices returns a list of OpenCL devices found. The cl_device_id values returned in *devices* can be used to identify a specific OpenCL device. If *devices* is NULL, this argument is ignored. The number of OpenCL devices returned is the minimum of the value specified by *num_entries* and the number of OpenCL devices corresponding to *d3d_object*.

num_devices returns the number of OpenCL devices available that correspond to *d3d_object*. If *num_devices* is NULL, this argument is ignored.

clGetDeviceIDsFromD3D10KHR returns CL_SUCCESS if the function is executed successfully. Otherwise it may return

- CL_INVALID_PLATFORM if *platform* is not a valid platform.
- CL_INVALID_VALUE if *d3d_device_source* is not a valid value, *d3d_device_set* is not a

valid value, *num_entries* is equal to zero and *devices* is not NULL, or if both *num_devices* and *devices* are NULL.

- CL_DEVICE_NOT_FOUND if no OpenCL devices that correspond to *d3d_object* were found.

cl_d3d_device_source_khr	Type of *d3d_object*
CL_D3D10_DEVICE_KHR	ID3D10Device *
CL_D3D10_DXGI_ADAPTER_KHR	IDXGIAdapter *

Table 9.9.1 *Types used to specify the object whose corresponding OpenCL devices are being queried by* ***clGetDeviceIDsFromD3D10KHR***

cl_d3d_device_set_khr	Devices returned in *devices*
CL_PREFERRED_DEVICES_FOR_D3D10_KHR	The OpenCL devices associated with the specified Direct3D object.
CL_ALL_DEVICES_FOR_D3D10_KHR	All OpenCL devices which may interoperate with the specified Direct3D object. Performance of sharing data on these devices may be considerably less than on the preferred devices.

Table 9.9.2 *Sets of devices queriable using* ***clGetDeviceIDsFromD3D10KHR***

9.9.7.2 Lifetime of Shared Objects

An OpenCL memory object created from a Direct3D 10 resource remains valid as long as the corresponding Direct3D 10 resource has not been deleted. If the Direct3D 10 resource is deleted through the Direct3D 10 API, subsequent use of the OpenCL memory object will result in undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

The successful creation of a cl_context against a Direct3D 10 device specified via the context create parameter CL_CONTEXT_D3D10_DEVICE_KHR will increment the internal Direct3D reference count on the specified Direct3D 10 device. The internal Direct3D reference count on that Direct3D 10 device will be decremented when the OpenCL reference count on the returned OpenCL context drops to zero.

The OpenCL context and corresponding command-queues are dependent on the existence of the Direct3D 10 device from which the OpenCL context was created. If the Direct3D 10 device is deleted through the Direct3D 10 API, subsequent use of the OpenCL context will result in

undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

9.9.7.3 Sharing Direct3D 10 Buffer Resources as OpenCL Buffer Objects

The function

cl_mem **clCreateFromD3D10BufferKHR** (cl_context *context*,
cl_mem_flags *flags*,
ID3D10Buffer **resource*,
cl_int **errcode_ret*)

creates an OpenCL buffer object from a Direct3D 10 buffer.

context is a valid OpenCL context created from a Direct3D 10 device.

flags is a bit-field that is used to specify usage information. Refer to table 5.3 for a description of *flags*. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* can be used.

resource is a pointer to the Direct3D 10 buffer to share.

errcode_ret will return an appropriate error code. If *errcode_ret* is NULL, no error code is returned.

clCreateFromD3D10BufferKHR returns a valid non-zero OpenCL buffer object and *errcode_ret* is set to CL_SUCCESS if the buffer object is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context.
- CL_INVALID_VALUE if values specified in *flags* are not valid.
- CL_INVALID_D3D10_RESOURCE_KHR if *resource* is not a Direct3D 10 buffer resource, if *resource* was created with the D3D10_USAGE flag D3D10_USAGE_IMMUTABLE, if a cl_mem from *resource* has already been created using **clCreateFromD3D10BufferKHR**, or if *context* was not created against the same Direct3D 10 device from which *resource* was created.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The size of the returned OpenCL buffer object is the same as the size of *resource*. This call will increment the internal Direct3D reference count on *resource*. The internal Direct3D reference count on *resource* will be decremented when the OpenCL reference count on the returned

OpenCL memory object drops to zero.

9.9.7.4 Sharing Direct3D 10 Texture and Resources as OpenCL Image Objects

The function

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

creates an OpenCL 2D image object from a subresource of a Direct3D 10 2D texture.

context is a valid OpenCL context created from a Direct3D 10 device.

flags is a bit-field that is used to specify usage information. Refer to table 5.3 for a description of *flags*. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* can be used.

resource is a pointer to the Direct3D 10 2D texture to share.

subresource is the subresource of *resource* to share.

errcode_ret will return an appropriate error code. If *errcode_ret* is NULL, no error code is returned.

clCreateFromD3D10Texture2DKHR returns a valid non-zero OpenCL image object and *errcode_ret* is set to CL_SUCCESS if the image object is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context.
- CL_INVALID_VALUE if values specified in *flags* are not valid or if *subresource* is not a valid subresource index for *resource*.
- CL_INVALID_D3D10_RESOURCE_KHR if *resource* is not a Direct3D 10 texture resource, if *resource* was created with the D3D10_USAGE flag D3D10_USAGE_IMMUTABLE, if *resource* is a multisampled texture, if a cl_mem from subresource *subresource* of *resource* has already been created using **clCreateFromD3D10Texture2DKHR**, or if *context* was not created against the same Direct3D 10 device from which *resource* was created.
- CL_INVALID_IMAGE_FORMAT_DESCRIPTOR if the Direct3D 10 texture format of

resource is not listed in *table 9.9.3* or if the Direct3D 10 texture format of *resource* does not map to a supported OpenCL image format.

- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The width and height of the returned OpenCL 2D image object are determined by the width and height of subresource *subresource* of *resource*. The channel type and order of the returned OpenCL 2D image object is determined by the format of *resource* by *table 9.9.3*.

This call will increment the internal Direct3D reference count on *resource*. The internal Direct3D reference count on *resource* will be decremented when the OpenCL reference count on the returned OpenCL memory object drops to zero.

The function

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

creates an OpenCL 3D image object from a subresource of a Direct3D 10 3D texture.

context is a valid OpenCL context created from a Direct3D 10 device.

flags is a bit-field that is used to specify usage information. Refer to table 5.3 for a description of *flags*. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* can be used.

resource is a pointer to the Direct3D 10 3D texture to share.

subresource is the subresource of *resource* to share.

errcode_ret will return an appropriate error code. If *errcode_ret* is NULL, no error code is returned.

clCreateFromD3D10Texture3DKHR returns a valid non-zero OpenCL image object and *errcode_ret* is set to CL_SUCCESS if the image object is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context.

- CL_INVALID_VALUE if values specified in *flags* are not valid or if *subresource* is not a valid subresource index for *resource*.

- CL_INVALID_D3D10_RESOURCE_KHR if *resource* is not a Direct3D 10 texture resource, if *resource* was created with the D3D10_USAGE flag D3D10_USAGE_IMMUTABLE, if *resource* is a multisampled texture, if a cl_mem from subresource *subresource* of *resource* has already been created using **clCreateFromD3D10Texture3DKHR**, or if *context* was not created against the same Direct3D 10 device from which *resource* was created.

- CL_INVALID_IMAGE_FORMAT_DESCRIPTOR if the Direct3D 10 texture format of *resource* is not listed in *table 9.9.3* or if the Direct3D 10 texture format of *resource* does not map to a supported OpenCL image format.

- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The width, height and depth of the returned OpenCL 3D image object are determined by the width, height and depth of subresource *subresource* of *resource*. The channel type and order of the returned OpenCL 3D image object is determined by the format of *resource* by *table 9.9.3*.

This call will increment the internal Direct3D reference count on *resource*. The internal Direct3D reference count on *resource* will be decremented when the OpenCL reference count on the returned OpenCL memory object drops to zero.

DXGI format	CL image format (channel order, channel data type)
DXGI_FORMAT_R32G32B32A32_FLOAT	CL_RGBA, CL_FLOAT
DXGI_FORMAT_R32G32B32A32_UINT	CL_RGBA, CL_UNSIGNED_INT32
DXGI_FORMAT_R32G32B32A32_SINT	CL_RGBA, CL_SIGNED_INT32
DXGI_FORMAT_R16G16B16A16_FLOAT	CL_RGBA, CL_HALF_FLOAT
DXGI_FORMAT_R16G16B16A16_UNORM	CL_RGBA, CL_UNORM_INT16
DXGI_FORMAT_R16G16B16A16_UINT	CL_RGBA, CL_UNSIGNED_INT16
DXGI_FORMAT_R16G16B16A16_SNORM	CL_RGBA, CL_SNORM_INT16
DXGI_FORMAT_R16G16B16A16_SINT	CL_RGBA, CL_SIGNED_INT16
DXGI_FORMAT_R8G8B8A8_UNORM	CL_RGBA, CL_UNORM_INT8
DXGI_FORMAT_R8G8B8A8_UINT	CL_RGBA, CL_UNSIGNED_INT8
DXGI_FORMAT_R8G8B8A8_SNORM	CL_RGBA, CL_SNORM_INT8
DXGI_FORMAT_R8G8B8A8_SINT	CL_RGBA, CL_SIGNED_INT8
DXGI_FORMAT_R32G32_FLOAT	CL_RG, CL_FLOAT
DXGI_FORMAT_R32G32_UINT	CL_RG, CL_UNSIGNED_INT32
DXGI_FORMAT_R32G32_SINT	CL_RG, CL_SIGNED_INT32
DXGI_FORMAT_R16G16_FLOAT	CL_RG, CL_HALF_FLOAT
DXGI_FORMAT_R16G16_UNORM	CL_RG, CL_UNORM_INT16

DXGI_FORMAT_R16G16_UINT	CL_RG, CL_UNSIGNED_INT16
DXGI_FORMAT_R16G16_SNORM	CL_RG, CL_SNORM_INT16
DXGI_FORMAT_R16G16_SINT	CL_RG, CL_SIGNED_INT16
DXGI_FORMAT_R8G8_UNORM	CL_RG, CL_UNORM_INT8
DXGI_FORMAT_R8G8_UINT	CL_RG, CL_UNSIGNED_INT8
DXGI_FORMAT_R8G8_SNORM	CL_RG, CL_SNORM_INT8
DXGI_FORMAT_R8G8_SINT	CL_RG, CL_SIGNED_INT8
DXGI_FORMAT_R32_FLOAT	CL_R, CL_FLOAT
DXGI_FORMAT_R32_UINT	CL_R, CL_UNSIGNED_INT32
DXGI_FORMAT_R32_SINT	CL_R, CL_SIGNED_INT32
DXGI_FORMAT_R16_FLOAT	CL_R, CL_HALF_FLOAT
DXGI_FORMAT_R16_UNORM	CL_R, CL_UNORM_INT16
DXGI_FORMAT_R16_UINT	CL_R, CL_UNSIGNED_INT16
DXGI_FORMAT_R16_SNORM	CL_R, CL_SNORM_INT16
DXGI_FORMAT_R16_SINT	CL_R, CL_SIGNED_INT16
DXGI_FORMAT_R8_UNORM	CL_R, CL_UNORM_INT8
DXGI_FORMAT_R8_UINT	CL_R, CL_UNSIGNED_INT8
DXGI_FORMAT_R8_SNORM	CL_R, CL_SNORM_INT8
DXGI_FORMAT_R8_SINT	CL_R, CL_SIGNED_INT8

Table 9.9.3 *List of Direct3D 10 and corresponding OpenCL image formats*

9.9.7.5 Querying Direct3D properties of memory objects created from Direct3D 10 resources

Properties of Direct3D 10 objects may be queried using **clGetMemObjectInfo** and **clGetImageInfo** with *param_name* CL_MEM_D3D10_RESOURCE_KHR and CL_IMAGE_D3D10_SUBRESOURCE_KHR respectively as described in *sections 5.4.3* and *5.3.6*.

9.9.7.6 Sharing memory objects created from Direct3D 10 resources between Direct3D 10 and OpenCL contexts

The function

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*)

is used to acquire OpenCL memory objects that have been created from Direct3D 10 resources. The Direct3D 10 objects are acquired by the OpenCL context associated with *command_queue* and can therefore be used by all command-queues associated with the OpenCL context.

OpenCL memory objects created from Direct3D 10 resources must be acquired before they can be used by any OpenCL commands queued to a command-queue. If an OpenCL memory object created from a Direct3D 10 resource is used while it is not currently acquired by OpenCL, the call attempting to use that OpenCL memory object will return CL_D3D10_RESOURCE_NOT_ACQUIRED_KHR.

If CL_CONTEXT_INTEROP_USER_SYNC is not specified as CL_TRUE during context creation, **clEnqueueAcquireD3D10ObjectsKHR** provides the synchronization guarantee that any Direct3D 10 calls involving the interop device(s) used in the OpenCL context made before **clEnqueueAcquireD3D10ObjectsKHR** is called will complete executing before *event* reports completion and before the execution of any subsequent OpenCL work issued in *command_queue* begins. If the context was created with properties specifying CL_CONTEXT_INTEROP_USER_SYNC as CL_TRUE, the user is responsible for guaranteeing that any Direct3D 10 calls involving the interop device(s) used in the OpenCL context made before **clEnqueueAcquireD3D10ObjectsKHR** is called have completed before calling **clEnqueueAcquireD3D10ObjectsKHR.**

command_queue is a valid command-queue.

num_objects is the number of memory objects to be acquired in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from Direct3D 10 resources.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is NULL, then this particular command does not wait on any event to complete. If *event_wait_list* is NULL, *num_events_in_wait_list* must be 0. If *event_wait_list* is not NULL, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points.

event returns an event object that identifies this particular command and can be used to query or queue a wait for this particular command to complete. *event* can be NULL in which case it will not be possible for the application to query the status of this command or queue a wait for this command to complete. If the *event_wait_list* and the *event* arguments are not NULL, the *event* argument should not refer to an element of the *event_wait_list* array.

clEnqueueAcquireD3D10ObjectsKHR returns CL_SUCCESS if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is NULL then the function does nothing and returns CL_SUCCESS. Otherwise it returns one of the following errors:

- CL_INVALID_VALUE if *num_objects* is zero and *mem_objects* is not a NULL value or if

num_objects > 0 and *mem_objects* is NULL.

- CL_INVALID_MEM_OBJECT if memory objects in *mem_objects* are not valid OpenCL memory objects or if memory objects in *mem_objects* have not been created from Direct3D 10 resources.

- CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue.

- CL_INVALID_CONTEXT if context associated with *command_queue* was not created from an Direct3D 10 context.

- CL_D3D10_RESOURCE_ALREADY_ACQUIRED_KHR if memory objects in *mem_objects* have previously been acquired using **clEnqueueAcquireD3D10ObjectsKHR** but have not been released using **clEnqueueReleaseD3D10ObjectsKHR**.

- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is NULL and *num_events_in_wait_list* > 0, or *event_wait_list* is not NULL and *num_events_in_wait_list* is 0, or if event objects in *event_wait_list* are not valid events.

- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

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*)

is used to release OpenCL memory objects that have been created from Direct3D 10 resources. The Direct3D 10 objects are released by the OpenCL context associated with *command_queue*.

OpenCL memory objects created from Direct3D 10 resources which have been acquired by OpenCL must be released by OpenCL before they may be accessed by Direct3D 10. Accessing a Direct3D 10 resource while its corresponding OpenCL memory object is acquired is in error and will result in undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

If CL_CONTEXT_INTEROP_USER_SYNC is not specified as CL_TRUE during context creation, **clEnqueueReleaseD3D10ObjectsKHR** provides the synchronization guarantee that any calls to Direct3D 10 calls involving the interop device(s) used in the OpenCL context made after the call to **clEnqueueReleaseD3D10ObjectsKHR** will not start executing until after all events in

event_wait_list are complete and all work already submitted to *command_queue* completes execution. If the context was created with properties specifying CL_CONTEXT_INTEROP_USER_SYNC as CL_TRUE, the user is responsible for guaranteeing that any Direct3D 10 calls involving the interop device(s) used in the OpenCL context made after **clEnqueueReleaseD3D10ObjectsKHR** will not start executing until after event returned by **clEnqueueReleaseD3D10ObjectsKHR** reports completion.

num_objects is the number of memory objects to be released in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from Direct3D 10 resources.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is NULL, then this particular command does not wait on any event to complete. If *event_wait_list* is NULL, *num_events_in_wait_list* must be 0. If *event_wait_list* is not NULL, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event* returns an event object that identifies this particular command and can be used to query or queue a wait for this particular command to complete. *event* can be NULL in which case it will not be possible for the application to query the status of this command or queue a wait for this command to complete. If the *event_wait_list* and the *event* arguments are not NULL, the *event* argument should not refer to an element of the *event_wait_list* array.

clEnqueueReleaseD3D10ObjectsKHR returns CL_SUCCESS if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is NULL the function does nothing and returns CL_SUCCESS. Otherwise it returns one of the following errors:

- CL_INVALID_VALUE if *num_objects* is zero and *mem_objects* is not a NULL value or if *num_objects* > 0 and *mem_objects* is NULL.
- CL_INVALID_MEM_OBJECT if memory objects in *mem_objects* are not valid OpenCL memory objects or if memory objects in *mem_objects* have not been created from Direct3D 10 resources.
- CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue.
- CL_INVALID_CONTEXT if context associated with *command_queue* was not created from a Direct3D 10 device.
- CL_D3D10_RESOURCE_NOT_ACQUIRED_KHR if memory objects in *mem_objects* have not previously been acquired using **clEnqueueAcquireD3D10ObjectsKHR**, or have been released using **clEnqueueReleaseD3D10ObjectsKHR** since the last time that they were acquired.
- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is NULL and *num_events_in_wait_list* > 0, or *event_wait_list* is not NULL and

num_events_in_wait_list> is 0, or if event objects in *event_wait_list* are not valid events.

- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

9.9.8 Issues

1) Should this extension be KHR or EXT?

PROPOSED: KHR. If this extension is to be approved by Khronos then it should be KHR, otherwise EXT. Not all platforms can support this extension, but that is also true of OpenGL interop.

RESOLVED: KHR.

2) Requiring SharedHandle on ID3D10Resource

Requiring this can largely simplify things at the DDI level and make some implementations faster. However, the DirectX spec only defines the shared handle for a subset of the resources we would like to support:

> D3D10_RESOURCE_MISC_SHARED - Enables the sharing of resource data between two or more Direct3D devices. The only resources that can be shared are 2D non-mipmapped textures.

PROPOSED A: Add wording to the spec about some implementations needing the resource setup as shared:

"Some implementations may require the resource to be shared on the D3D10 side of the API"

If we do that, do we need another enum to describe this failure case?

PROPOSED B: Require that all implementations support both shared and non-shared resources. The restrictions prohibiting multisample textures and the flag D3D10_USAGE_IMMUTABLE guarantee software access to all shareable resources.

RESOLVED: Require that implementations support both D3D10_RESOURCE_MISC_SHARED being set and not set. Add the query for CL_CONTEXT_D3D10_PREFER_SHARED_RESOURCES_KHR to determine on a per-context basis which method will be faster.

3) Texture1D support

There is not a matching CL type, so do we want to support this and map to buffer or Texture2D? If so the command might correspond to the 2D / 3D versions:

cl_mem **clCreateFromD3D10Texture1D** (cl_context *context*,
cl_mem_flags *flags*,
ID3D10Texture2D **resource*,
UINT *subresource*,
cl_int **errcode_ret*)

RESOLVED: We will not add support for ID3D10Texture1D objects unless a corresponding OpenCL 1D Image type is created.

4) CL/D3D10 queries

The GL interop has clGetGLObjectInfo and clGetGLTextureInfo. It is unclear if these are needed on the D3D10 interop side since the D3D10 spec makes these queries trivial on the D3D10 object itself. Also, not all of the sematics of the GL call map across.

PROPOSED: Add the **clGetMemObjectInfo** and **clGetImageInfo** parameter names CL_MEM_D3D10_RESOURCE_KHR and CL_IMAGE_D3D10_SUBRESOURCE_KHR to query the D3D10 resource from which a cl_mem was created. From this data, any D3D10 side information may be queried using the D3D10 API.

RESOLVED: We will use **clGetMemObjectInfo** and **clGetImageInfo** to access this information.

9.10 DX9 Media Surface Sharing

9.10.1 Overview

The goal of this extension is to allow applications to use media surfaces as OpenCL memory objects. This allows efficient sharing of data between OpenCL and selected adapter APIs (only DX9 for now). If this extension is supported, an OpenCL image object can be created from a media surface and the OpenCL API can be used to execute kernels that read and/or write memory objects that are media surfaces. Note that OpenCL memory objects may be created from the adapter media surface if and only if the OpenCL context has been created from that adapter.

If this extension is supported by an implementation, the string **cl_khr_dx9_media_sharing** will be present in the CL_PLATFORM_EXTENSIONS or CL_DEVICE_EXTENSIONS string described in *table 4.3*.

9.10.2 Header File

As currently proposed the interfaces for this extension would be provided in `cl_dx9_media_sharing.h`.

9.10.3 New Procedures and Functions

cl_int **clGetDeviceIDsFromDX9MediaAdapterKHR** (cl_platform_id *platform*,
cl_uint *num_media_adapters*,
cl_dx9_media_adapter_type_khr **media_adapters_type*,
void **media_adapters*,
cl_dx9_media_adapter_set_khr *media_adapter_set*,
cl_uint *num_entries*,
cl_device_id **devices*,
cl_int **num_devices*)

cl_mem **clCreateFromDX9MediaSurfaceKHR** (cl_context *context*,
cl_mem_flags *flags*,
cl_dx9_media_adapter_type_khr *adapter_type*,
void **surface_info*,
cl_uint *plane*,
cl_int **errcode_ret*)

cl_int **clEnqueueAcquireDX9MediaSurfacesKHR** (
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 **clEnqueueReleaseDX9MediaSurfacesKHR** (
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*)

9.10.4 New Tokens

Accepted by the *media_adapter_type* parameter of **clGetDeviceIDsFromDX9MediaAdapterKHR**:

CL_ADAPTER_D3D9_KHR	0x2020
CL_ADAPTER_D3D9EX_KHR	0x2021
CL_ADAPTER_DXVA_KHR	0x2022

Accepted by the *media_adapter_set* parameter of **clGetDeviceIDsFromDX9MediaAdapterKHR**:

CL_PREFERRED_DEVICES_FOR_DX9_MEDIA_ADAPTER_KHR	0x2023
CL_ALL_DEVICES_FOR_DX9_MEDIA_ADAPTER_KHR	0x2024

Accepted as a property name in the *properties* parameter of **clCreateContext** and **clCreateContextFromType**:

CL_CONTEXT_ADAPTER_D3D9_KHR	0x2025
CL_CONTEXT_ADAPTER_D3D9EX_KHR	0x2026
CL_CONTEXT_ADAPTER_DXVA_KHR	0x2027

Accepted as the property being queried in the *param_name* parameter of **clGetMemObjectInfo**:

CL_MEM_DX9_MEDIA_ADAPTER_TYPE_KHR	0x2028
CL_MEM_DX9_MEDIA_SURFACE_INFO_KHR	0x2029

Accepted as the property being queried in the *param_name* parameter of **clGetImageInfo**:

CL_IMAGE_DX9_MEDIA_PLANE_KHR	0x202A

Returned in the *param_value* parameter of **clGetEventInfo** when *param_name* is CL_EVENT_COMMAND_TYPE:

CL_COMMAND_ACQUIRE_DX9_MEDIA_SURFACES_KHR	0x202B
CL_COMMAND_RELEASE_DX9_MEDIA_SURFACES_KHR	0x202C

Returned by **clCreateContext** and **clCreateContextFromType** if the media adapter specified for interoperability is not compatible with the devices against which the context is to be created:

CL_INVALID_DX9_MEDIA_ADAPTER_KHR	-1010

Returned by **clCreateFromDX9MediaSurfaceKHR** when *adapter_type* is set to a media adapter and the *surface_info* does not reference a media surface of the required type, or if *adapter_type* is set to a media adapter type and *surface_info* does not contain a valid reference to a media surface on that adapter, by **clGetMemObjectInfo** when *param_name* is a surface or handle when the image was not created from an appropriate media surface, and from **clGetImageInfo** when *param_name* is CL IMAGE_DX9_MEDIA_PLANE KHR and image was not created from an appropriate media surface.

CL_INVALID_DX9_MEDIA_SURFACE_KHR	-1011

Returned by **clEnqueueAcquireDX9MediaSurfacesKHR** when any of *mem_objects* are currently acquired by OpenCL

CL_DX9_MEDIA_SURFACE_ALREADY_ACQUIRED_KHR	-1012

Returned by **clEnqueueReleaseDX9MediaSurfacesKHR** when any of *mem_objects* are not currently acquired by OpenCL

CL_DX9_MEDIA_SURFACE_NOT_ACQUIRED_KHR	-1013

9.10.5 Additions to Chapter 4 of the OpenCL 1.2 Specification

In *section 4.4*, replace the description of *properties* under **clCreateContext** with:

"*properties* specifies a list of context property names and their corresponding values. Each property is followed immediately by the corresponding desired value. The list is terminated with zero. If a property is not specified in *properties*, then its default value (listed in *table 4.5*) is used (it is said to be specified implicitly). If *properties* is NULL or empty (points to a list whose first value is zero), all attributes take on their default values."

Add the following to *table 4.5*:

cl_context_properties enum	Property value	Description
CL_CONTEXT_ADAPTER_ D3D9_KHR	IDirect3DDevice9 *	Specifies an IDirect3DDevice9 to use for D3D9 interop.
CL_CONTEXT_ADAPTER_ D3D9EX_KHR	IDirect3DDeviceEx*	Specifies an IDirect3DDevice9Ex to use for D3D9 interop.
CL_CONTEXT_ADAPTER_ DXVA_KHR	IDXVAHD_Device *	Specifies an IDXVAHD_Device to use for DXVA interop.

Add to the list of errors for **clCreateContext**:

- CL_INVALID_ADAPTER_KHR if any of the values of the properties CL_CONTEXT_ADAPTER_D3D9_KHR, CL_CONTEXT_ADAPTER_D3D9EX_KHR or CL_CONTEXT_ADAPTER_DXVA_KHR is non-NULL and does not specify a valid media adapter with which the *cl_device_ids* against which this context is to be created may interoperate.

- CL_INVALID_OPERATION if interoperability is specified by setting CL_CONTEXT_ADAPTER_D3D9_KHR, CL_CONTEXT_ADAPTER_D3D9EX_KHR or CL_CONTEXT_ADAPTER_DXVA_KHR to a non-NULL value, and interoperability with another graphics API is also specified."

Add to the list of errors for **clCreateContextFromType** the same new errors described above for **clCreateContext**.

9.10.6 Additions to Chapter 5 of the OpenCL 1.2 Specification

Add to the list of errors for **clGetMemObjectInfo**:

- CL_INVALID_DX9_MEDIA_SURFACE_KHR if *param_name* is CL_MEM_DX9_MEDIA_SURFACE_INFO_KHR and *memobj* was not created by the function **clCreateFromDX9MediaSurfaceKHR** from a Direct3D9 surface.

Extend *table 5.11* to include the following entry.

cl_mem_info	Return type	Info. returned in *param_value*
CL_MEM_DX9_MEDIA ADAPTER_TYPE_KHR	cl_dx9_media_ adapter_type_khr	Returns the *cl_dx9_media_adapter_type_khr* argument value specified when *memobj* is created using **clCreateFromDX9MediaSurfaceKHR**.

CL_MEM_DX9_MEDIA SURFACE_INFO_KHR	cl_dx9_surface_info_khr	Returns the *cl_dx9_surface_info_khr* argument value specified when *memobj* is created using **clCreateFromDX9MediaSurfaceKHR**.

Add to the list of errors for **clGetImageInfo**:

- CL_INVALID_DX9_MEDIA_SURFACE_KHR if *param_name* is CL_IMAGE_DX9_MEDIA_PLANE_KHR and *image* was not created by the function **clCreateFromDX9MediaSurfaceKHR**.

Extend *table 5.9* to include the following entry.

cl_image_info	Return type	Info. returned in *param_value*
CL_IMAGE_DX9_MEDIA _PLANE_KHR	cl_uint	Returns the *plane* argument value specified when *memobj* is created using **clCreateFromDX9MediaSurfaceKHR**.

Add to *table 5.18* in the **Info returned in param_value** column for *cl_event_info* = CL_EVENT_COMMAND_TYPE:

CL_COMMAND_ACQUIRE_DX9_MEDIA_SURFACES_KHR
CL_COMMAND_RELEASE_DX9_MEDIA_SURFACES_KHR

9.10.7 Sharing Media Surfaces with OpenCL

This section discusses OpenCL functions that allow applications to use media surfaces as OpenCL memory objects. This allows efficient sharing of data between OpenCL and media surface APIs. The OpenCL API may be used to execute kernels that read and/or write memory objects that are also media surfaces. An OpenCL image object may be created from a media surface. OpenCL memory objects may be created from media surfaces if and only if the OpenCL context has been created from a media adapter.

9.10.7.1 Querying OpenCL Devices corresponding to Media Adapters

Media adapters are an abstraction associated with devices that provide media capabilities.

The function

cl_int **clGetDeviceIDsFromDX9MediaAdapterKHR** (cl_platform_id *platform*,
cl_uint *num_media_adapters*,
cl_dx9_media_adapter_type_khr **media_adapters_type*,
void **media_adapters*,
cl_dx9_media_adapter_set_khr *media_adapter_set*,
cl_uint *num_entries*,
cl_device_id **devices*,
cl_int **num_devices*)

queries a media adapter for any associated OpenCL devices. Adapters with associated OpenCL devices can enable media surface sharing between the two.

platform refers to the platform ID returned by **clGetPlatformIDs**.

num_media_adapters specifies the number of media adapters.

media_adapters_type is an array of *num_media_adapters* entries. Each entry specifies the type of media adapter and must be one of the values described in *table 9.10.1*.

cl_dx9_media_adapter_type_khr	Type of media adapters
CL_ADAPTER_D3D9_KHR	IDirect3DDevice9 *
CL_ADAPTER_D3D9EX_KHR	IDirect3DDevice9Ex *
CL_ADAPTER_DXVA_KHR	IDXVAHD_Device *

Table 9.10.1 *List of cl_dx9_media_adapter_type_khr values*

cl_dx9_media_adapter_set_khr	Description
CL_PREFERRED_DEVICES_FOR_ MEDIA_ADAPTER_KHR	The preferred OpenCL devices associated with the media adapter.
CL_ALL_DEVICES_FOR_MEDIA_ ADAPTER_KHR	All OpenCL devices that may interoperate with the media adapter

Table 9.10.2 *List of cl_dx9_media_adapter_set_khr values*

media_adapters is an array of *num_media_adapters* entries. Each entry specifies the actual adapter whose type is specified by *media_adapter_type*. The *media_adapters* must be one of the types describes in *table 9.10.1*.

media_adapter_set specifies the set of adapters to return and must be one of the values described in *table 9.10.2*.

num_entries is the number of cl_device_id entries that can be added to *devices.* If *devices* is not NULL, the *num_entries* must be greater than zero.

devices returns a list of OpenCL devices found that support the list of media adapters specified. The cl_device_id values returned in *devices* can be used to identify a specific OpenCL device. If *devices* argument is NULL, this argument is ignored. The number of OpenCL devices returned is the minimum of the value specified by *num_entries* or the number of OpenCL devices whose type matches *device_type*.

num_devices returns the number of OpenCL devices. If *num_devices* is NULL, this argument is ignored.

clGetDeviceIDsFromDX9MediaAdapterKHR returns CL_SUCCESS if the function is executed successfully. Otherwise, it returns one of the following errors:

- CL_INVALID_PLATFORM if *platform* is not a valid platform.
- CL_INVALID_VALUE if *num_media_adapters* is zero or if *media_adapters_type* is NULL or if *media_adapters* is NULL.
- CL_INVALID_VALUE if any of the entries in *media_adapters_type* or *media_adapters* is not a valid value.
- CL_INVALID_VALUE if *media_adapter_set* is not a valid value.
- CL_INVALID_VALUE if *num_entries* is equal to zero and *devices* is not NULL or if both *num_devices* and *devices* are NULL.
- CL_DEVICE_NOT_FOUND if no OpenCL devices that correspond to adapters specified in *media_adapters* and *media_adapters_type* were found.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

9.10.7.2 Creating Media Resources as OpenCL Image Objects

The function

```
cl_mem      clCreateFromDX9MediaSurfaceKHR (cl_context context,
                                            cl_mem_flags flags,
                                            cl_dx9_media_adapter_type_khr adapter_type,
                                            void *surface_info,
                                            cl_uint plane,
                                            cl_int *errcode_ret)
```

creates an OpenCL image object from a media surface.

context is a valid OpenCL context created from a media adapter.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of flags. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* can be used.

adapter_type is a value from enumeration of supported adapters described in *table 9.10.1*. The type of *surface_info* is determined by the adapter type. The implementation does not need to support all adapter types. This approach provides flexibility to support additional adapter types in the future. Supported adapter types are CL_ADAPTER_D3D9_KHR, CL_ADAPTER_D3D9EX_KHR and CL_ADAPTER_DXVA_KHR.

If *adapter_type* is CL_ADAPTER_D3D9_KHR, CL_ADAPTER_D3D9EX_KHR and CL_ADAPTER_DXVA_KHR, the *surface_info* points to the following structure:

```
typedef struct _cl_dx9_surface_info_khr
{
     IDirect3DSurface9 *resource;
     HANDLE shared_handle;
} cl_dx9_surface_info_khr;
```

For DX9 surfaces, we need both the handle to the resource and the resource itself to have a sufficient amount of information to eliminate a copy of the surface for sharing in cases where this is possible. Elimination of the copy is driver dependent. *shared_handle* may be NULL and this may result in sub-optimal performance.

surface_info is a pointer to one of the structures defined in the *adapter_type* description above passed in as a void *.

plane is the plane of resource to share for planar surface formats. For planar formats, we use the plane parameter to obtain a handle to thie specific plane (Y, U or V for example). For non-planar formats used by media, *plane* must be 0.

errcode_ret will return an appropriate error code. If *errcode_ret* is NULL, no error code is returned.

clCreateFromDX9MediaSurfaceKHR returns a valid non-zero 2D image object and *errcode_ret* is set to CL_SUCCESS if the 2D image object is created successfully. Otherwise it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context.
- CL_INVALID_VALUE if values specified in *flags* are not valid or if *plane* is not a valid plane of *resource* specified in *surface_info*.

- CL_INVALID_DX9_MEDIA_SURFACE_KHR if *resource* specified in *surface_info* is not a valid resource or is not associated with *adapter_type* (e.g., *adapter_type* is set to CL_ADAPTER_D3D9_KHR and *resource* is not a Direct3D 9 surface created in D3DPOOL_DEFAULT).

- CL_INVALID_DX9_MEDIA_SURFACE_KHR if *shared_handle* specified in *surface_info* is not NULL or a valid handle value.

- CL_INVALID_IMAGE_FORMAT_DESCRIPTOR if the texture format of *resource* is not listed in *tables 9.10.3* and *9.10.4*.

- CL_INVALID_OPERATION if there are no devices in *context* that support *adapter_type*.

- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.

- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The width and height of the returned OpenCL 2D image object are determined by the width and height of the plane of resource. The channel type and order of the returned image object is determined by the format and plane of resource and are described in *tables 9.10.3* and *9.10.4*.

This call will increment the internal media surface count on *resource*. The internal media surface reference count on *resource* will be decremented when the OpenCL reference count on the returned OpenCL memory object drops to zero.

9.10.7.3 Querying Media Surface Properties of Memory Objects created from Media Surfaces

Properties of media surface objects may be queried using **clGetMemObjectInfo** and **clGetImageInfo** with *param_name* CL_MEM_DX9_MEDIA_ADAPTER_TYPE_KHR, CL_MEM_DX9_MEDIA_SURFACE_INFO_KHR and CL_IMAGE_DX9_MEDIA_PLANE_KHR as described in *sections 5.4.3* and *5.3.6*.

9.10.7.4 Sharing Memory Objects created from Media Surfaces between a Media Adapter and OpenCL

The function

cl_int **clEnqueueAcquireDX9MediaSurfacesKHR** (
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*)

is used to acquire OpenCL memory objects that have been created from a media surface. The media surfaces are acquired by the OpenCL context associated with *command_queue* and can therefore be used by all command-queues associated with the OpenCL context.

OpenCL memory objects created from media surfaces must be acquired before they can be used by any OpenCL commands queued to a command-queue. If an OpenCL memory object created from a media surface is used while it is not currently acquired by OpenCL, the call attempting to use that OpenCL memory object will return CL_DX9_MEDIA_SURFACE_NOT_ACQUIRED_KHR.

If CL_CONTEXT_INTEROP_USER_SYNC is not specified as CL_TRUE during context creation, **clEnqueueAcquireDX9MediaSurfacesKHR** provides the synchronization guarantee that any media adapter API calls involving the interop device(s) used in the OpenCL context made before **clEnqueueAcquireDX9MediaSurfacesKHR** is called will complete executing before *event* reports completion and before the execution of any subsequent OpenCL work issued in *command_queue* begins. If the context was created with properties specifying CL_CONTEXT_INTEROP_USER_SYNC as CL_TRUE, the user is responsible for guaranteeing that any media adapter API calls involving the interop device(s) used in the OpenCL context made before **clEnqueueAcquireDX9MediaSurfacesKHR** is called have completed before calling **clEnqueueAcquireDX9MediaSurfacesKHR .**

command_queue is a valid command-queue.

num_objects is the number of memory objects to be acquired in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from media surfaces.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is NULL, then this particular command does not wait on any event to complete. If *event_wait_list* is NULL, *num_events_in_wait_list* must be 0. If *event_wait_list* is not NULL, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in

event_wait_list act as synchronization points.

event returns an event object that identifies this particular command and can be used to query or queue a wait for this particular command to complete. *event* can be NULL in which case it will not be possible for the application to query the status of this command or queue a wait for this command to complete. If the *event_wait_list* and the *event* arguments are not NULL, the *event* argument should not refer to an element of the *event_wait_list* array.

clEnqueueAcquireDX9MediaSurfacesKHR returns CL_SUCCESS if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is NULL then the function does nothing and returns CL_SUCCESS. Otherwise it returns one of the following errors:

- CL_INVALID_VALUE if *num_objects* is zero and *mem_objects* is not a NULL value or if *num_objects* > 0 and *mem_objects* is NULL.
- CL_INVALID_MEM_OBJECT if memory objects in *mem_objects* are not valid OpenCL memory objects or if memory objects in *mem_objects* have not been created from media surfaces.
- CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue.
- CL_INVALID_CONTEXT if context associated with *command_queue* was not created from a device that can share the media surface referenced by *mem_objects*.
- CL_DX9_MEDIA_SURFACE_ALREADY_ACQUIRED_KHR if memory objects in *mem_objects* have previously been acquired using **clEnqueueAcquireDX9MediaSurfacesKHR** but have not been released using **clEnqueueReleaseDX9MediaSurfacesKHR**.
- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is NULL and *num_events_in_wait_list* > 0, or *event_wait_list* is not NULL and *num_events_in_wait_list* is 0, or if event objects in *event_wait_list* are not valid events.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

```
cl_int  clEnqueueReleaseDX9MediaSurfacesKHR (
                                        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)
```

is used to release OpenCL memory objects that have been created from media surfaces. The media surfaces are released by the OpenCL context associated with *command_queue*.

OpenCL memory objects created from media surfaces which have been acquired by OpenCL must be released by OpenCL before they may be accessed by the media adapter API. Accessing a media surface while its corresponding OpenCL memory object is acquired is in error and will result in undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

If CL_CONTEXT_INTEROP_USER_SYNC is not specified as CL_TRUE during context creation, **clEnqueueReleaseDX9MediaSurfacesKHR** provides the synchronization guarantee that any calls to media adapter APIs involving the interop device(s) used in the OpenCL context made after the call to **clEnqueueReleaseDX9MediaSurfacesKHR** will not start executing until after all events in *event_wait_list* are complete and all work already submitted to *command_queue* completes execution. If the context was created with properties specifying CL_CONTEXT_INTEROP_USER_SYNC as CL_TRUE, the user is responsible for guaranteeing that any media adapter API calls involving the interop device(s) used in the OpenCL context made after **clEnqueueReleaseDX9MediaSurfacesKHR** will not start executing until after event returned by **clEnqueueReleaseDX9MediaSurfacesKHR** reports completion.

num_objects is the number of memory objects to be released in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from media surfaces.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is NULL, then this particular command does not wait on any event to complete. If *event_wait_list* is NULL, *num_events_in_wait_list* must be 0. If *event_wait_list* is not NULL, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event* returns an event object that identifies this particular command and can be used to query or queue a wait for this particular command to complete. *event* can be NULL in which case it will not be possible for the application to query the status of this command or queue a wait for this command to complete. If the *event_wait_list* and the *event* arguments are not NULL, the *event* argument should not refer to an element of the *event_wait_list* array.

clEnqueueReleaseDX9MediaSurfaceKHR returns CL_SUCCESS if the function is executed successfully. If *num_objects* is 0 and <*mem_objects*> is NULL the function does nothing and returns CL_SUCCESS. Otherwise it returns one of the following errors:

- CL_INVALID_VALUE if *num_objects* is zero and *mem_objects* is not a NULL value or if *num_objects* > 0 and *mem_objects* is NULL.
- CL_INVALID_MEM_OBJECT if memory objects in *mem_objects* are not valid OpenCL memory objects or if memory objects in *mem_objects* have not been created from valid

media surfaces.

- CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue.
- CL_INVALID_CONTEXT if context associated with *command_queue* was not created from a media object.
- CL_DX9_MEDIA_SURFACE_NOT_ACQUIRED_KHR if memory objects in *mem_objects* have not previously been acquired using **clEnqueueAcquireDX9MediaSurfacesKHR**, or have been released using **clEnqueueReleaseDX9MediaSurfacesKHR** since the last time that they were acquired.
- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is NULL and *num_events_in_wait_list* > 0, or *event_wait_list* is not NULL and *num_events_in_wait_list*> is 0, or if event objects in *event_wait_list* are not valid events.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

9.10.7.5 Surface formats for Media Suface Sharing

This section includes the D3D surface formats that are supported when the adapter type is one of the Direct 3D lineage . Using a D3D surface format not listed here is an error. To extend the use of this extension to support media adapters beyond DirectX9 tables similar to the ones in this section will need to be defined for the surface formats supported by the new media adapter. All implementations that support this extension are required to support the NV12 surface format, the other surface formats supported are the same surface formats that the adapter you are sharing with supports as long as they are listed in the *table 9.10.3* and *table 9.10.4*.

FOUR CC code	CL image format (channel order, channel data type)
FOURCC('N','V','1','2'), Plane 0	CL_R, CL_UNORM_INT8
FOURCC('N','V','1','2'), Plane 1	CL_RG, CL_UNORM_INT8
FOURCC('Y','V','1','2'), Plane 0	CL_R, CL_UNORM_INT8
FOURCC('Y','V','1','2'), Plane 1	CL_R, CL_UNORM_INT8
FOURCC('Y','V','1','2'), Plane 2	CL_R, CL_UNORM_INT8

Table 9.10.3 *YUV FourCC codes and corresponding OpenCL image format*

In *table 9.10.3*, NV12 Plane 0 corresponds to the luminance (Y) channel and Plane 1 corresponds to the UV channels. The YV12 Plane 0 corresponds to the Y channel, Plane 1 corresponds to the V channel and Plane 2 corresponds to the U channel. Note that the YUV formats map to CL_R and CL_RG but do not perform any YUV to RGB conversion and vice-versa.

D3D format[10]	CL image format (channel order, channel data type)
D3DFMT_R32F	CL_R, CL_FLOAT
D3DFMT_R16F	CL_R, CL_HALF_FLOAT
D3DFMT_L16	CL_R, CL_UNORM_INT16
D3DFMT_A8	CL_A, CL_UNORM_INT8
D3DFMT_L8	CL_R, CL_UNORM_INT8
D3DFMT_G32R32F	CL_RG, CL_FLOAT
D3DFMT_G16R16F	CL_RG, CL_HALF_FLOAT
D3DFMT_G16R16	CL_RG, CL_UNORM_INT16
D3DFMT_A8L8	CL_RG, CL_UNORM_INT8
D3DFMT_A32B32G32R32F	CL_RGBA, CL_FLOAT
D3DFMT_A16B16G16R16F	CL_RGBA, CL_HALF_FLOAT
D3DFMT_A16B16G16R16	CL_RGBA, CL_UNORM_INT16
D3DFMT_A8B8G8R8	CL_RGBA, CL_UNORM_INT8
D3DFMT_X8B8G8R8	CL_RGBA, CL_UNORM_INT8
D3DFMT_A8R8G8B8	CL_BGRA, CL_UNORM_INT8
D3DFMT_X8R8G8B8	CL_BGRA, CL_UNORM_INT8

Table 9.10.4 *List of Direct3D and corresponding OpenCL image formats*

[10] Note that D3D9 format names seem to imply that the order of the color channels are switched relative to OpenCL but this is not the case. For example, layout of channels for each pixel for D3DFMT_A32FB32FG32FR32F is the same as CL_RGBA, CL_FLOAT.

9.11 Sharing Memory Objects with Direct3D 11

9.11.1 Overview

The goal of this extension is to provide interoperability between OpenCL and Direct3D 11. This is designed to function analogously to the OpenGL interoperability as defined in *sections 9.7* and *9.8*. If this extension is supported by an implementation, the string **cl_khr_d3d11_sharing** will be present in the CL_PLATFORM_EXTENSIONS or CL_DEVICE_EXTENSIONS string described in *table 4.3*.

9.11.2 Header File

As currently proposed the interfaces for this extension would be provided in `cl_d3d11.h`.

9.11.3 New Procedures and Functions

cl_int **clGetDeviceIDsFromD3D11KHR** (cl_platform_id *platform*,
cl_d3d11_device_source_khr *d3d_device_source*,
void **d3d_object*,
cl_d3d11_device_set_khr *d3d_device_set*,
cl_uint *num_entries*,
cl_device_id **devices*,
cl_uint **num_devices*)

cl_mem **clCreateFromD3D11BufferKHR** (cl_context *context*,
cl_mem_flags *flags*,
ID3D11Buffer **resource*,
cl_int **errcode_ret*)

cl_mem **clCreateFromD3D11Texture2DKHR** (cl_context *context*,
cl_mem_flags *flags*,
ID3D11Texture2D **resource*,
UINT *subresource*,
cl_int **errcode_ret*)

cl_mem **clCreateFromD3D11Texture3DKHR** (cl_context *context*,
cl_mem_flags *flags*,
ID3D11Texture3D **resource*,
UINT *subresource*,
cl_int **errcode_ret*)

cl_int **clEnqueueAcquireD3D11ObjectsKHR** (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 **clEnqueueReleaseD3D11ObjectsKHR** (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*)

9.11.4 New Tokens

Accepted as a Direct3D 11 device source in the *d3d_device_source* parameter of **clGetDeviceIDsFromD3D11KHR**:

CL_D3D11_DEVICE_KHR	0x4019
CL_D3D11_DXGI_ADAPTER_KHR	0x401A

Accepted as a set of Direct3D 11 devices in the *d3d_device_set* parameter of **clGetDeviceIDsFromD3D11KHR**:

CL_PREFERRED_DEVICES_FOR_D3D11_KHR	0x401B
CL_ALL_DEVICES_FOR_D3D11_KHR	0x401C

Accepted as a property name in the *properties* parameter of **clCreateContext** and **clCreateContextFromType**:

CL_CONTEXT_D3D11_DEVICE_KHR	0x401D

Accepted as a property name in the *param_name* parameter of **clGetContextInfo**:

CL_CONTEXT_D3D11_PREFER_SHARED_RESOURCES_KHR	0x402D

Accepted as the property being queried in the *param_name* parameter of **clGetMemObjectInfo**:

CL_MEM_D3D11_RESOURCE_KHR	0x401E

Accepted as the property being queried in the *param_name* parameter of **clGetImageInfo**:

CL_IMAGE_D3D11_SUBRESOURCE_KHR 0x401F

Returned in the *param_value* parameter of **clGetEventInfo** when *param_name* is CL_EVENT_COMMAND_TYPE:

CL_COMMAND_ACQUIRE_D3D11_OBJECTS_KHR 0x4020
CL_COMMAND_RELEASE_D3D11_OBJECTS_KHR 0x4021

Returned by **clCreateContext** and **clCreateContextFromType** if the Direct3D 11 device specified for interoperability is not compatible with the devices against which the context is to be created:

CL_INVALID_D3D11_DEVICE_KHR -1006

Returned by **clCreateFromD3D11BufferKHR** when *resource* is not a Direct3D 11 buffer object, and by **clCreateFromD3D11Texture2DKHR** and **clCreateFromD3D11Texture3DKHR** when *resource* is not a Direct3D 11 texture object.

CL_INVALID_D3D11_RESOURCE_KHR -1007

Returned by **clEnqueueAcquireD3D11ObjectsKHR** when any of *mem_objects* are currently acquired by OpenCL

CL_D3D11_RESOURCE_ALREADY_ACQUIRED_KHR -1008

Returned by **clEnqueueReleaseD3D11ObjectsKHR** when any of *mem_objects* are not currently acquired by OpenCL

CL_D3D11_RESOURCE_NOT_ACQUIRED_KHR -1009

9.11.5 Additions to Chapter 4 of the OpenCL 1.2 Specification

In *section 4.4*, replace the description of *properties* under **clCreateContext** with:

"*properties* specifies a list of context property names and their corresponding values. Each property is followed immediately by the corresponding desired value. The list is terminated with zero. If a property is not specified in *properties*, then its default value (listed in *table 4.5*) is used (it is said to be specified implicitly). If *properties* is NULL or empty (points to a list whose first value is zero), all attributes take on their default values."

Add the following to *table 4.5*:

cl_context_properties enum	Property value	Description
CL_CONTEXT_D3D11_DEVICE_KHR	ID3D11Device *	Specifies the ID3D11Device *

		to use for Direct3D 11 interoperability. The default value is NULL.

Add to the list of errors for **clCreateContext**:

- CL_INVALID_D3D11_DEVICE_KHR if the value of the property CL_CONTEXT_D3D11_DEVICE_KHR is non-NULL and does not specify a valid Direct3D 11 device with which the *cl_device_ids* against which this context is to be created may interoperate.

- CL_INVALID_OPERATION if Direct3D 11 interoperability is specified by setting CL_INVALID_D3D11_DEVICE_KHR to a non-NULL value, and interoperability with another graphics API is also specified."

Add to the list of errors for **clCreateContextFromType** the same new errors described above for **clCreateContext**.

Add the following row to *table 4.7*:

cl_context_info	Return Type	Information returned in param_value
CL_CONTEXT_D3D11_PREFER_SHARED_RESOURCES_KHR	cl_bool	Returns CL_TRUE if Direct3D 11 resources created as shared by setting *MiscFlags* to include D3D11_RESOURCE_MISC_SHARED will perform faster when shared with OpenCL, compared with resources which have not set this flag. Otherwise returns CL_FALSE.

9.11.6 Additions to Chapter 5 of the OpenCL 1.2 Specification

Add to the list of errors for **clGetMemObjectInfo**:

- CL_INVALID_D3D11_RESOURCE_KHR if *param_name* is CL_MEM_D3D11_RESOURCE_KHR and *memobj* was not created by the function **clCreateFromD3D11BufferKHR**, **clCreateFromD3D11Texture2DKHR**, or **clCreateFromD3D11Texture3DKHR**."

Extend *table 5.11* to include the following entry.

cl_mem_info	Return type	Info. returned in *param_value*
CL_MEM_D3D11_ RESOURCE_KHR	ID3D11Resource *	If *memobj* was created using **clCreateFromD3D11BufferKHR**, **clCreateFromD3D11Texture2DKHR**, or **clCreateFromD3D11Texture3DKHR**, returns the *resource* argument specified when *memobj* was created.

Add to the list of errors for **clGetImageInfo**:

- CL_INVALID_D3D11_RESOURCE_KHR if *param_name* is CL_MEM_D3D11_SUBRESOURCE_KHR and *image* was not created by the function **clCreateFromD3D11Texture2DKHR**, or **clCreateFromD3D11Texture3DKHR**."

Extend *table 5.9* to include the following entry.

cl_image_info	Return type	Info. returned in *param_value*
CL_MEM_D3D11_ SUBRESOURCE_KHR	ID3D11Resource *	If *image* was created using **clCreateFromD3D11Texture2DKHR**, or **clCreateFromD3D11Texture3DKHR**, returns the *subresource* argument specified when *image* was created.

Add to *table 5.18* in the **Info returned in param_value** column for *cl_event_info* = CL_EVENT_COMMAND_TYPE:

CL_COMMAND_ACQUIRE_D3D11_OBJECTS_KHR
CL_COMMAND_RELEASE_D3D11_OBJECTS_KHR

9.11.7 Sharing Memory Objects with Direct3D 11 Resources

This section discusses OpenCL functions that allow applications to use Direct3D 11 resources as OpenCL memory objects. This allows efficient sharing of data between OpenCL and Direct3D 11. The OpenCL API may be used to execute kernels that read and/or write memory objects that are also Direct3D 11 resources. An OpenCL image object may be created from a Direct3D 11 texture resource. An OpenCL buffer object may be created from a Direct3D 11 buffer resource. OpenCL memory objects may be created from Direct3D 11 objects if and only if the OpenCL context has been created from a Direct3D 11 device.

9.11.7.1 Querying OpenCL Devices Corresponding to Direct3D 11 Devices

The OpenCL devices corresponding to a Direct3D 11 device may be queried. The OpenCL devices corresponding to a DXGI adapter may also be queried. The OpenCL devices corresponding to a Direct3D 11 device will be a subset of the OpenCL devices corresponding to the DXGI adapter against which the Direct3D 11 device was created.

The OpenCL devices corresponding to a Direct3D 11 device or a DXGI device may be queried using the function

cl_int **clGetDeviceIDsFromD3D11KHR** (cl_platform_id *platform*,
cl_d3d11_device_source_khr *d3d_device_source*,
void **d3d_object*,
cl_d3d11_device_set_khr *d3d_device_set*,
cl_uint *num_entries*,
cl_device_id **devices*,
cl_uint **num_devices*)

platform refers to the platform ID returned by **clGetPlatformIDs**.

d3d_device_source specifies the type of *d3d_object*, and must be one of the values shown in *table 9.11.1*.

d3d_object specifies the object whose corresponding OpenCL devices are being queried. The type of *d3d_object* must be as specified in *table 9.11.1*.

d3d_device_set specifies the set of devices to return, and must be one of the values shown in *table 9.11.2*.

num_entries is the number of cl_device_id entries that can be added to *devices*. If *devices* is not NULL then *num_entries* must be greater than zero.

devices returns a list of OpenCL devices found. The cl_device_id values returned in *devices* can be used to identify a specific OpenCL device. If *devices* is NULL, this argument is ignored. The number of OpenCL devices returned is the minimum of the value specified by *num_entries* and the number of OpenCL devices corresponding to *d3d_object*.

num_devices returns the number of OpenCL devices available that correspond to *d3d_object*. If *num_devices* is NULL, this argument is ignored.

clGetDeviceIDsFromD3D10KHR returns CL_SUCCESS if the function is executed successfully. Otherwise it may return

- CL_INVALID_PLATFORM if *platform* is not a valid platform.
- CL_INVALID_VALUE if *d3d_device_source* is not a valid value, *d3d_device_set* is not a

valid value, *num_entries* is equal to zero and *devices* is not NULL, or if both *num_devices* and *devices* are NULL.

- CL_DEVICE_NOT_FOUND if no OpenCL devices that correspond to *d3d_object* were found.

cl_d3d_device_source_khr	Type of *d3d_object*
CL_D3D11_DEVICE_KHR	ID3D11Device *
CL_D3D11_DXGI_ADAPTER_KHR	IDXGIAdapter *

Table 9.11.1 *Types used to specify the object whose corresponding OpenCL devices are being queried by* ***clGetDeviceIDsFromD3D11KHR***

cl_d3d_device_set_khr	Devices returned in *devices*
CL_PREFERRED_DEVICES_FOR_D3D11_KHR	The preferred OpenCL devices associated with the specified Direct3D object.
CL_ALL_DEVICES_FOR_D3D11_KHR	All OpenCL devices which may interoperate with the specified Direct3D object. Performance of sharing data on these devices may be considerably less than on the preferred devices.

Table 9.11.2 *Sets of devices queriable using* ***clGetDeviceIDsFromD3D11KHR***

9.11.7.2 Lifetime of Shared Objects

An OpenCL memory object created from a Direct3D 11 resource remains valid as long as the corresponding Direct3D 11 resource has not been deleted. If the Direct3D 11 resource is deleted through the Direct3D 11 API, subsequent use of the OpenCL memory object will result in undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

The successful creation of a cl_context against a Direct3D 11 device specified via the context create parameter CL_CONTEXT_D3D11_DEVICE_KHR will increment the internal Direct3D reference count on the specified Direct3D 11 device. The internal Direct3D reference count on that Direct3D 11 device will be decremented when the OpenCL reference count on the returned OpenCL context drops to zero.

The OpenCL context and corresponding command-queues are dependent on the existence of the Direct3D 11 device from which the OpenCL context was created. If the Direct3D 11 device is deleted through the Direct3D 11 API, subsequent use of the OpenCL context will result in

undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

9.11.7.3 Sharing Direct3D 11 Buffer Resources as OpenCL Buffer Objects

The function

cl_mem **clCreateFromD3D11BufferKHR** (cl_context *context*,
cl_mem_flags *flags*,
ID3D11Buffer **resource*,
cl_int **errcode_ret*)

creates an OpenCL buffer object from a Direct3D 11 buffer.

context is a valid OpenCL context created from a Direct3D 11 device.

flags is a bit-field that is used to specify usage information. Refer to table 5.3 for a description of *flags*. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* can be used.

resource is a pointer to the Direct3D 11 buffer to share.

errcode_ret will return an appropriate error code. If *errcode_ret* is NULL, no error code is returned.

clCreateFromD3D11BufferKHR returns a valid non-zero OpenCL buffer object and *errcode_ret* is set to CL_SUCCESS if the buffer object is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context.
- CL_INVALID_VALUE if values specified in *flags* are not valid.
- CL_INVALID_D3D11_RESOURCE_KHR if *resource* is not a Direct3D 11 buffer resource, if *resource* was created with the D3D11_USAGE flag D3D11_USAGE_IMMUTABLE, if a cl_mem from *resource* has already been created using **clCreateFromD3D11BufferKHR**, or if *context* was not created against the same Direct3D 11 device from which *resource* was created.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The size of the returned OpenCL buffer object is the same as the size of *resource*. This call will increment the internal Direct3D reference count on *resource*. The internal Direct3D reference count on *resource* will be decremented when the OpenCL reference count on the returned

OpenCL memory object drops to zero.

9.11.7.4 Sharing Direct3D 11 Texture and Resources as OpenCL Image Objects

The function

cl_mem **clCreateFromD3D11Texture2DKHR** (cl_context *context*,
cl_mem_flags *flags*,
ID3D11Texture2D **resource*,
UINT *subresource*,
cl_int **errcode_ret*)

creates an OpenCL 2D image object from a subresource of a Direct3D 11 2D texture.

context is a valid OpenCL context created from a Direct3D 11 device.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of *flags*. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* can be used.

resource is a pointer to the Direct3D 11 2D texture to share.

subresource is the subresource of *resource* to share.

errcode_ret will return an appropriate error code. If *errcode_ret* is NULL, no error code is returned.

clCreateFromD3D11Texture2DKHR returns a valid non-zero OpenCL image object and *errcode_ret* is set to CL_SUCCESS if the image object is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context.
- CL_INVALID_VALUE if values specified in *flags* are not valid or if *subresource* is not a valid subresource index for *resource*.
- CL_INVALID_D3D11_RESOURCE_KHR if *resource* is not a Direct3D 11 texture resource, if *resource* was created with the D3D11_USAGE flag D3D11_USAGE_IMMUTABLE, if *resource* is a multisampled texture, if a cl_mem from subresource *subresource* of *resource* has already been created using **clCreateFromD3D11Texture2DKHR**, or if *context* was not created against the same Direct3D 10 device from which *resource* was created.
- CL_INVALID_IMAGE_FORMAT_DESCRIPTOR if the Direct3D 11 texture format of

resource is not listed in *table 9.11.3* or if the Direct3D 11 texture format of *resource* does not map to a supported OpenCL image format.

- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The width and height of the returned OpenCL 2D image object are determined by the width and height of subresource *subresource* of *resource*. The channel type and order of the returned OpenCL 2D image object is determined by the format of *resource* by *table 9.11.3*.

This call will increment the internal Direct3D reference count on *resource*. The internal Direct3D reference count on *resource* will be decremented when the OpenCL reference count on the returned OpenCL memory object drops to zero.

The function

cl_mem **clCreateFromD3D11Texture3DKHR** (cl_context *context*,
cl_mem_flags *flags*,
ID3D11Texture3D **resource*,
UINT *subresource*,
cl_int **errcode_ret*)

creates an OpenCL 3D image object from a subresource of a Direct3D 11 3D texture.

context is a valid OpenCL context created from a Direct3D 11 device.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of *flags*. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* can be used.

resource is a pointer to the Direct3D 11 3D texture to share.

subresource is the subresource of *resource* to share.

errcode_ret will return an appropriate error code. If *errcode_ret* is NULL, no error code is returned.

clCreateFromD3D11Texture3DKHR returns a valid non-zero OpenCL image object and *errcode_ret* is set to CL_SUCCESS if the image object is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context.

- CL_INVALID_VALUE if values specified in *flags* are not valid or if *subresource* is not a valid subresource index for *resource*.

- CL_INVALID_D3D11_RESOURCE_KHR if *resource* is not a Direct3D 11 texture resource, if *resource* was created with the D3D11_USAGE flag D3D11_USAGE_IMMUTABLE, if *resource* is a multisampled texture, if a cl_mem from subresource *subresource* of *resource* has already been created using **clCreateFromD3D11Texture3DKHR**, or if *context* was not created against the same Direct3D 11 device from which *resource* was created.

- CL_INVALID_IMAGE_FORMAT_DESCRIPTOR if the Direct3D 11 texture format of *resource* is not listed in *table 9.11.3* or if the Direct3D 11 texture format of *resource* does not map to a supported OpenCL image format.

- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The width, height and depth of the returned OpenCL 3D image object are determined by the width, height and depth of subresource *subresource* of *resource*. The channel type and order of the returned OpenCL 3D image object is determined by the format of *resource* by *table 9.9.3*.

This call will increment the internal Direct3D reference count on *resource*. The internal Direct3D reference count on *resource* will be decremented when the OpenCL reference count on the returned OpenCL memory object drops to zero.

DXGI format	CL image format (channel order, channel data type)
DXGI_FORMAT_R32G32B32A32_FLOAT	CL_RGBA, CL_FLOAT
DXGI_FORMAT_R32G32B32A32_UINT	CL_RGBA, CL_UNSIGNED_INT32
DXGI_FORMAT_R32G32B32A32_SINT	CL_RGBA, CL_SIGNED_INT32
DXGI_FORMAT_R16G16B16A16_FLOAT	CL_RGBA, CL_HALF_FLOAT
DXGI_FORMAT_R16G16B16A16_UNORM	CL_RGBA, CL_UNORM_INT16
DXGI_FORMAT_R16G16B16A16_UINT	CL_RGBA, CL_UNSIGNED_INT16
DXGI_FORMAT_R16G16B16A16_SNORM	CL_RGBA, CL_SNORM_INT16
DXGI_FORMAT_R16G16B16A16_SINT	CL_RGBA, CL_SIGNED_INT16
DXGI_FORMAT_R8G8B8A8_UNORM	CL_RGBA, CL_UNORM_INT8
DXGI_FORMAT_R8G8B8A8_UINT	CL_RGBA, CL_UNSIGNED_INT8
DXGI_FORMAT_R8G8B8A8_SNORM	CL_RGBA, CL_SNORM_INT8
DXGI_FORMAT_R8G8B8A8_SINT	CL_RGBA, CL_SIGNED_INT8
DXGI_FORMAT_R32G32_FLOAT	CL_RG, CL_FLOAT
DXGI_FORMAT_R32G32_UINT	CL_RG, CL_UNSIGNED_INT32
DXGI_FORMAT_R32G32_SINT	CL_RG, CL_SIGNED_INT32
DXGI_FORMAT_R16G16_FLOAT	CL_RG, CL_HALF_FLOAT
DXGI_FORMAT_R16G16_UNORM	CL_RG, CL_UNORM_INT16

DXGI_FORMAT_R16G16_UINT	CL_RG, CL_UNSIGNED_INT16
DXGI_FORMAT_R16G16_SNORM	CL_RG, CL_SNORM_INT16
DXGI_FORMAT_R16G16_SINT	CL_RG, CL_SIGNED_INT16
DXGI_FORMAT_R8G8_UNORM	CL_RG, CL_UNORM_INT8
DXGI_FORMAT_R8G8_UINT	CL_RG, CL_UNSIGNED_INT8
DXGI_FORMAT_R8G8_SNORM	CL_RG, CL_SNORM_INT8
DXGI_FORMAT_R8G8_SINT	CL_RG, CL_SIGNED_INT8
DXGI_FORMAT_R32_FLOAT	CL_R, CL_FLOAT
DXGI_FORMAT_R32_UINT	CL_R, CL_UNSIGNED_INT32
DXGI_FORMAT_R32_SINT	CL_R, CL_SIGNED_INT32
DXGI_FORMAT_R16_FLOAT	CL_R, CL_HALF_FLOAT
DXGI_FORMAT_R16_UNORM	CL_R, CL_UNORM_INT16
DXGI_FORMAT_R16_UINT	CL_R, CL_UNSIGNED_INT16
DXGI_FORMAT_R16_SNORM	CL_R, CL_SNORM_INT16
DXGI_FORMAT_R16_SINT	CL_R, CL_SIGNED_INT16
DXGI_FORMAT_R8_UNORM	CL_R, CL_UNORM_INT8
DXGI_FORMAT_R8_UINT	CL_R, CL_UNSIGNED_INT8
DXGI_FORMAT_R8_SNORM	CL_R, CL_SNORM_INT8
DXGI_FORMAT_R8_SINT	CL_R, CL_SIGNED_INT8

Table 9.11.3 *List of Direct3D 11 and corresponding OpenCL image formats*

9.11.7.5 Querying Direct3D properties of memory objects created from Direct3D 11 resources

Properties of Direct3D 11 objects may be queried using **clGetMemObjectInfo** and **clGetImageInfo** with *param_name* CL_MEM_D3D11_RESOURCE_KHR and CL_IMAGE_D3D11_SUBRESOURCE_KHR respectively as described in *sections 5.4.3* and *5.3.6*.

9.11.7.6 Sharing memory objects created from Direct3D 11 resources between Direct3D 11 and OpenCL contexts

The function

cl_int **clEnqueueAcquireD3D11ObjectsKHR** (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*)

is used to acquire OpenCL memory objects that have been created from Direct3D 11 resources. The Direct3D 11 objects are acquired by the OpenCL context associated with *command_queue* and can therefore be used by all command-queues associated with the OpenCL context.

OpenCL memory objects created from Direct3D 11 resources must be acquired before they can be used by any OpenCL commands queued to a command-queue. If an OpenCL memory object created from a Direct3D 11 resource is used while it is not currently acquired by OpenCL, the call attempting to use that OpenCL memory object will return CL_D3D11_RESOURCE_NOT_ACQUIRED_KHR.

If CL_CONTEXT_INTEROP_USER_SYNC is not specified as CL_TRUE during context creation, **clEnqueueAcquireD3D11ObjectsKHR** provides the synchronization guarantee that any Direct3D 11 calls involving the interop device(s) used in the OpenCL context made before **clEnqueueAcquireD3D11ObjectsKHR** is called will complete executing before *event* reports completion and before the execution of any subsequent OpenCL work issued in *command_queue* begins. If the context was created with properties specifying CL_CONTEXT_INTEROP_USER_SYNC as CL_TRUE, the user is responsible for guaranteeing that any Direct3D 11 calls involving the interop device(s) used in the OpenCL context made before **clEnqueueAcquireD3D11ObjectsKHR** is called have completed before calling **clEnqueueAcquireD3D11ObjectsKHR.**

command_queue is a valid command-queue.

num_objects is the number of memory objects to be acquired in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from Direct3D 11 resources.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is NULL, then this particular command does not wait on any event to complete. If *event_wait_list* is NULL, *num_events_in_wait_list* must be 0. If *event_wait_list* is not NULL, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points.

event returns an event object that identifies this particular command and can be used to query or queue a wait for this particular command to complete. *event* can be NULL in which case it will not be possible for the application to query the status of this command or queue a wait for this command to complete. If the *event_wait_list* and the *event* arguments are not NULL, the *event* argument should not refer to an element of the *event_wait_list* array.

clEnqueueAcquireD3D11ObjectsKHR returns CL_SUCCESS if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is NULL then the function does nothing and returns CL_SUCCESS. Otherwise it returns one of the following errors:

- CL_INVALID_VALUE if *num_objects* is zero and *mem_objects* is not a NULL value or if

num_objects > 0 and *mem_objects* is NULL.

- CL_INVALID_MEM_OBJECT if memory objects in *mem_objects* are not valid OpenCL memory objects or if memory objects in *mem_objects* have not been created from Direct3D 11 resources.

- CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue.

- CL_INVALID_CONTEXT if context associated with *command_queue* was not created from an Direct3D 11 context.

- CL_D3D11_RESOURCE_ALREADY_ACQUIRED_KHR if memory objects in *mem_objects* have previously been acquired using **clEnqueueAcquireD3D11ObjectsKHR** but have not been released using **clEnqueueReleaseD3D11ObjectsKHR**.

- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is NULL and *num_events_in_wait_list* > 0, or *event_wait_list* is not NULL and *num_events_in_wait_list* is 0, or if event objects in *event_wait_list* are not valid events.

- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

cl_int **clEnqueueReleaseD3D11ObjectsKHR** (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*)

is used to release OpenCL memory objects that have been created from Direct3D 11 resources. The Direct3D 11 objects are released by the OpenCL context associated with *command_queue*.

OpenCL memory objects created from Direct3D 11 resources which have been acquired by OpenCL must be released by OpenCL before they may be accessed by Direct3D 11. Accessing a Direct3D 11 resource while its corresponding OpenCL memory object is acquired is in error and will result in undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

If CL_CONTEXT_INTEROP_USER_SYNC is not specified as CL_TRUE during context creation, **clEnqueueReleaseD3D11ObjectsKHR** provides the synchronization guarantee that any calls to Direct3D 11 calls involving the interop device(s) used in the OpenCL context made after the call to **clEnqueueReleaseD3D11ObjectsKHR** will not start executing until after all events in

event_wait_list are complete and all work already submitted to *command_queue* completes execution. If the context was created with properties specifying CL_CONTEXT_INTEROP_USER_SYNC as CL_TRUE, the user is responsible for guaranteeing that any Direct3D 11 calls involving the interop device(s) used in the OpenCL context made after **clEnqueueReleaseD3D11ObjectsKHR** will not start executing until after event returned by **clEnqueueReleaseD3D11ObjectsKHR** reports completion.

num_objects is the number of memory objects to be released in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from Direct3D 11 resources.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is NULL, then this particular command does not wait on any event to complete. If *event_wait_list* is NULL, *num_events_in_wait_list* must be 0. If *event_wait_list* is not NULL, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event* returns an event object that identifies this particular command and can be used to query or queue a wait for this particular command to complete. *event* can be NULL in which case it will not be possible for the application to query the status of this command or queue a wait for this command to complete. If the *event_wait_list* and the *event* arguments are not NULL, the *event* argument should not refer to an element of the *event_wait_list* array.

clEnqueueReleaseD3D11ObjectsKHR returns CL_SUCCESS if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is NULL the function does nothing and returns CL_SUCCESS. Otherwise it returns one of the following errors:

- CL_INVALID_VALUE if *num_objects* is zero and *mem_objects* is not a NULL value or if *num_objects* > 0 and *mem_objects* is NULL.
- CL_INVALID_MEM_OBJECT if memory objects in *mem_objects* are not valid OpenCL memory objects or if memory objects in *mem_objects* have not been created from Direct3D 11 resources.
- CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue.
- CL_INVALID_CONTEXT if context associated with *command_queue* was not created from a Direct3D 11 device.
- CL_D3D11_RESOURCE_NOT_ACQUIRED_KHR if memory objects in *mem_objects* have not previously been acquired using **clEnqueueAcquireD3D11ObjectsKHR**, or have been released using **clEnqueueReleaseD3D11ObjectsKHR** since the last time that they were acquired.
- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is NULL and *num_events_in_wait_list* > 0, or *event_wait_list* is not NULL and

num_events_in_wait_list> is 0, or if event objects in *event_wait_list* are not valid events.

- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

9.12 OpenCL Installable Client Driver (ICD)

9.12.1 Overview

This is a platform extension which defines a simple mechanism through which the Khronos OpenCL installable client driver loader (ICD Loader) may expose multiple separate vendor installable client drivers (Vendor ICDs) for OpenCL. An application written against the ICD Loader will be able to access all cl_platform_ids exposed by all vendor implementations with the ICD Loader acting as a demultiplexor. If this extension is supported by an implementation, the string **cl_khr_icd** will be present in the CL_PLATFORM_EXTENSIONS string described in *table 4.1*.

9.12.2 Inferring Vendors from Function Calls from Arguments

At every OpenCL function call, the ICD Loader infers the vendor ICD function to call from the arguments to the function. An object is said to be ICD compatible if it is of the following structure:

```
struct _cl_<object>
{
     struct _cl_icd_dispatch *dispatch;
     // ... remainder of internal data
};
```

<object> is one of `platform_id`, `device_id`, `context`, `command_queue`, `mem`, `program`, `kernel`, `event`, or `sampler`.

The structure `_cl_icd_dispatch` is a function pointer dispatch table which is used to direct calls to a particular vendor implementation. All objects created from ICD compatible objects must be ICD compatible.

A link to source code which defines the entries in the function table structure `_cl_icd_dispatch` is available in the Sample Code section of this document. The order of the functions in `_cl_icd_dispatch` is determined by the ICD Loader's source. The ICD Loader's source's `_cl_icd_dispatch` table is to be appended to only.

Functions which do not have an argument from which the vendor implementation may be inferred are ignored, with the exception of **clGetExtensionFunctionAddress** which is described below.

9.12.3 ICD Data

A Vendor ICD is defined by two pieces of data:

- The Vendor ICD library specifies a library which contains the OpenCL entrypoints for the vendor's OpenCL implementation. The vendor ICD's library file name should include the vendor name, or a vendor-specific implementation identifier.
- The Vendor ICD extension suffix is a short string which specifies the default suffix for extensions implemented only by that vendor. See Additions to Chapter 9 for details on the mechanism through which this is accomplished. The vendor suffix string is optional.

9.12.4 ICD Loader Vendor Enumeration on Windows

To enumerate Vendor ICDs on Windows, the ICD Loader scans the values in the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Khronos\OpenCL\Vendors`. For each value in this key which has DWORD data set to 0, the ICD Loader opens the dynamic link library specified by the name of the value using LoadLibraryA.

For example, if the registry contains the following value

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Khronos\OpenCL\Vendors]
"c:\\vendor a\\vndra_ocl.dll"=dword:00000000
```

then the ICD will open the library "`c:\vendor a\vndra_ocl.dll`".

9.12.5 ICD Loader Vendor Enumeration on Linux

To enumerate vendor ICDs on Linux, the ICD Loader scans the files in the path `/etc/OpenCL/vendors`. For each file in this path, the ICD Loader opens the file as a text file. The expected format for the file is a single line of text which specifies the Vendor ICD's library. The ICD Loader will attempt to open that file as a shared object using `dlopen()`. Note that the library specified may be an absolute path or just a file name.

For example, if the following file exists `/etc/OpenCL/vendors/VendorA.icd` and contains the text `libVendorAOpenCL.so` then the ICD Loader will load the library "`libVendorAOpenCL.so`".

9.12.6 Adding a Vendor Library

Upon successfully loading a Vendor ICD's library, the ICD Loader queries the following

functions from the library: **clIcdGetPlatformIDsKHR**, **clGetPlatformInfo**, and **clGetExtensionFunctionAddress**. If any of these functions are not present then the ICD Loader will close and ignore the library.

Next the ICD Loader queries available ICD-enabled platforms in the library using **clIcdGetPlatformIDsKHR**. For each of these platforms, the ICD Loader queries the platform's extension string to verify that **cl_khr_icd** is supported, then queries the platform's Vendor ICD extension suffix using **clGetPlatformInfo** with the value CL_PLATFORM_ICD_SUFFIX_KHR.

If any of these steps fail, the ICD Loader will ignore the Vendor ICD and continue on to the next.

9.12.7 New Procedures and Functions

cl_int **clIcdGetPlatformIDsKHR** (cl_uint *num_entries*,
cl_platform_id **platforms*,
cl_uint **num_platforms*);

9.12.8 New Tokens

Accepted as *param_name* to the function **clGetPlatformInfo**

CL_PLATFORM_ICD_SUFFIX_KHR	0x0920

Returned by **clGetPlatformIDs** when no platforms are found

CL_PLATFORM_NOT_FOUND_KHR	-1001

9.12.9 Additions to Chapter 4 of the OpenCL 1.2 Specification

In section 4.1, replace the description of the return values of clGetPlatformIDs with:

"clGetPlatformIDs returns CL_SUCCESS if the function is executed successfully and there are a non zero number of platforms available. It returns CL_PLATFORM_NOT_FOUND_KHR if zero platforms are available. It returns CL_INVALID_VALUE if <num_entries> is equal to zero and <platforms> is not NULL or if both <num_platforms> and <platforms> are NULL."

In *section 4.1*, add the following after the description of **clGetPlatformIDs**:

"The list of platforms accessible through the Khronos ICD Loader can be obtained using the

following function:

cl_int **clIcdGetPlatformIDsKHR** (cl_uint *num_entries*,
cl_platform_id **platforms*,
cl_uint **num_platforms*);

num_entries is the number of cl_platform_id entries that can be added to *platforms*. If *platforms* is not NULL, then *num_entries* must be greater than zero.

platforms returns a list of OpenCL platforms available for access through the Khronos ICD Loader. The cl_platform_id values returned in *platforms* are ICD compatible and can be used to identify a specific OpenCL platform. If the *platforms* argument is NULL, then this argument is ignored. The number of OpenCL platforms returned is the minimum of the value specified by *num_entries* or the number of OpenCL platforms available.

num_platforms returns the number of OpenCL platforms available. If *num_platforms* is NULL, then this argument is ignored.

clIcdGetPlatformIDsKHR returns CL_SUCCESS if the function is executed successfully and there are a non zero number of platforms available. It returns CL_PLATFORM_NOT_FOUND_KHR if zero platforms are available. It returns CL_INVALID_VALUE if *num_entries* is equal to zero and *platforms* is not NULL or if both *num_platforms* and *platforms* are NULL."

Add the following to *table 4.1*:

cl_platform_info enum	Return Type	Description
CL_PLATFORM_ICD_SUFFIX_KHR	char[]	The function name suffix used to identify extension functions to be directed to this platform by the ICD Loader.

9.12.10 Additions to Chapter 9 of the OpenCL 1.2 Extension Specification

Add the following paragraph to the end of Section 9.2:

"For functions supported by the ICD Loader, **clGetExtensionFunctionAddress** will return the function pointer of the ICD Loader implementation. For extension functions which the ICD Loader is unaware of, the function **clGetExtensionFunctionAddress** will determine the vendor implementation to return based on the string passed in. The ICD Loader will return the result from querying **clGetExtensionFunctionAddress** on the vendor ICD enumerated by the ICD Loader whose ICD suffix is a suffix of the function name being queried. If no such vendor exists

or the suffix of the function is KHR or EXT then **clGetExtensionFunctionAddress** will return NULL."

9.12.11 Issues

1. Some OpenCL functions do not take an object argument from which their vendor library may be identified (e.g, clUnloadCompiler), how will they be handled?

RESOLVED: Such functions will be a noop for all calls through the ICD.

2. How are OpenCL extension to be handled?

RESOLVED: OpenCL extension functions may be added to the ICD as soon as they are implemented by any vendor. The suffix mechanism provides access for vendor extensions which are not yet added to the ICD.

3: How will the ICD handle a NULL cl_platform_id?

RESOLVED: The NULL platform is not supported by the ICD.

4. There exists no mechanism to unload the ICD, should there be one?

RESOLVED: As there is no standard mechanism for unloading a vendor implementation, do not add one for the ICD.

Index - APIs