D3D9 Media Surface Sharing Between Intel® Quick Sync Video and OpenCL* on Intel® HD Graphics

Abstract

Intel has defined an extension to OpenCL* version 1.0 and newer, allowing applications to directly access images embedded in Microsoft DirectX* 9 (DX9) media surfaces, without first copying them. For OpenCL v1.2, the Khronos standards organization has defined a standardized extension for the same purpose. Intel® Quick Sync Video is able to decode video images into DX9 surfaces. The Intel and Khronos extensions can significantly improve performance of applications that use OpenCL to process Intel Quick Sync Video frames. These benefits can apply to a range of applications, including video enhancement during playback, video editing with effects, and video generation and encoding

OpenCL and Intel Quick Sync Video Media Surface Sharing

Intel Quick Sync Video supports video decode and encode acceleration by Intel® processor graphics and can be accessed by applications using the Intel® Media SDK. The Intel Media SDK and Intel® SDK for OpenCL are distributed as part of both the Intel® Media Server Studio [1] and the Intel® Integrated Native Development Experience (Intel® INDE) [2]. OpenCL [3] is a framework for developing applications to take advantage of heterogeneous platforms. In the case considered in this article, these platforms consist of a CPU and a programmable graphics unit, with parts of the application executing on both to achieve improved processing efficiency and performance. The Intel defined extension to OpenCL v1.0 (and usable on subsequent versions) enables direct access to DX9 media surfaces created by Intel Quick Sync Video.

Intel Quick Sync Video media surface sharing with OpenCL can avoid many surface copies between Intel Quick Sync Video and OpenCL kernels. If an application tries to apply an OpenCL filter to Intel Quick Sync Video without surface sharing, it will need to copy Intel Quick Sync Video output surfaces to OpenCL image objects so that the OpenCL filter can consume the data. This copy will get more time-consuming as the resolution of input video increases, as there is more data to copy.

Figure 1 shows an example of this, in which OpenCL is used to modify a video stream. Video frames decoded by Intel Quick Sync Video into DX9 surfaces are copied from the GPU to the CPU, then copied to an OpenCL memory object, copied from CPU to GPU, processed in OpenCL with the output copied back to the CPU, then copied into a DX9 surface and finally copied back to the GPU for video encode by Intel Quick Sync Video. With shared DX9 surfaces, none of these copies will be needed.

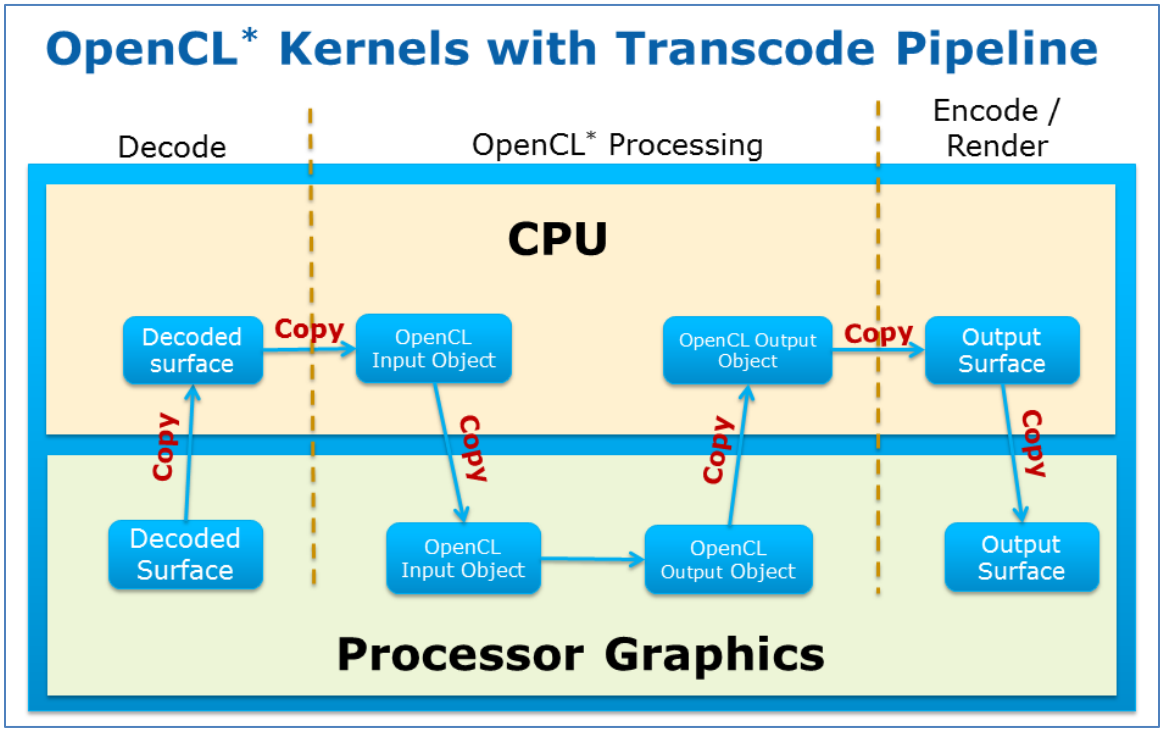


Figure 1: Video transcode and processing pipeline without surface sharing

Because Intel processor graphics shares memory with the CPU, image surfaces might only need to be copied during conversion to a system memory buffer. Once a system memory pointer is available, it can be used to create an OpenCL buffer using that same system memory buffer. The inverse is also true. After OpenCL processing completes into an OpenCL buffer that shares a system memory buffer, that buffer could be copied and converted to a DX9 buffer. So the total memory copies could be reduced to two for Intel processor graphics. Still, it will be better to eliminate all the extra copies.

Using the Intel DirectX 9 media surface sharing extension of OpenCL , a DX9 image surface in the NV12 format (used internally by Intel Quick Sync Video) can be converted directly into an NV12 OpenCL image2D memory object without copying, and vice-versa, as shown in Figure 2. This can avoid the extra data copies to convert video frames going between Intel Quick Sync Video and an OpenCL filter. So for a transcode application, the shared media surface extension can save at least two extra memory copies.

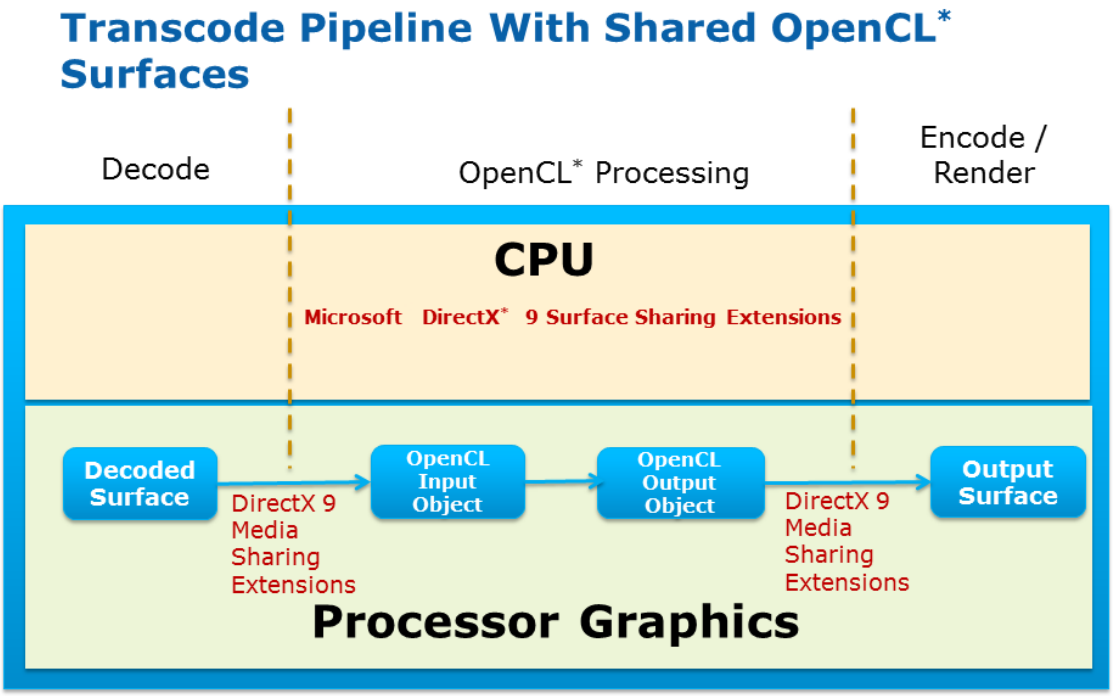


Figure 2: Video transcode and processing pipeline using MSDK/OCL surface sharing extensions

OpenCL DirectX 9 Media Sharing APIs

Intel has defined vendor-specific extension for OpenCL version 1.0 (and beyond), called **cl_intel_DX9_media_sharing**[3]. That OpenCL extension API includes four functions that allow applications to share DirectX 9 media surfaces with OpenCL:

- clGetDeviceIDsFromDX9INTEL
- clCreateFromDX9MediaSurfaceINTEL
- clEnqueueAcquireDX9ObjectsINTEL
- clEnqueueReleaseDX9ObjectsINTEL

Using the above API extension, the OpenCL API can be enabled to execute kernels that read and/or write memory objects that are also DirectX 9 resources. An OpenCL 2D image object can be created from a DirectX 9 media surface resource. OpenCL memory objects may be created from DirectX 9 objects if and only if the OpenCL context has been created from a DirectX 9 device. That is, the OpenCL **clCreateContext** or **clCreateContextFromType** functions must be called with a properties list that includes the property CL_CONTEXT_D3D9EX_DEVICE_INTEL paired with a pointer to the IDirect3DDevice9 created for Intel Quick Sync Video. (More information on this can be found in "Intel Extension for DirectX 9 Media Sharing". [4])

clGetDeviceIDsFromDX9INTEL: This function is used to query OpenCL devices corresponding to DirectX 9 device on a particular platform. The application should create the OpenCL context from DirectX 9 device.

clCreateFromDX9MediaSurfaceINTEL: This function is used to create an OpenCL 2D image object from a DirectX 9 media surface or a plane of a DirectX 9 media surface.

Applications can query the properties of the media surface object created from the DirectX 9 resource by using clGetMemObjectInfo and clGetImageInfo functions.

clEnqueueAcquireDX9ObjectsINTEL: This function is used to acquire OpenCL memory objects that have been created from the DirectX9 resources. The OpenCL memory objects created from DirectX 9 resources must be acquired before they can be used by any OpenCL commands queued to a command-queue. This function provides synchronization guaranteeing that any DirectX calls made before **clEnqueueAcquireDX9ObjectsINTEL** is called will complete execution before the execution of any subsequent OpenCL APIs submitted to the command-queue.

clEnqueueReleaseDX9ObjectsINTEL: This function is used to release OpenCL memory objects that have been created from the DirectX9 resources. This function provides the synchronization guaranteeing that any calls to DirectX9 resource made after the call to clEnqueueReleaseDX9ObjectsINTEL will not start executing until after all events in event list of this API are complete and all work already submitted to the command-queue completes execution.

To use this extension API, the application must first obtain a pointer to each of the corresponding functions. For OpenCL v1.0 and v1.1, use the standard OpenCL function **clGetExtensionFunctionAddress**, passing it the extension function name as a string. The file **cl_ext.h** in the Intel® SDK for OpenCL™ distribution contains function prototypes (with "_fn" appended to the API name) that can be used to correctly cast the function pointers.

In OpenCL version 1.2 and newer, Khronos has added an equivalent standard Khronos extension.[5] Using that extension, the equivalents of the above API names become:

- clGetDeviceIDsFromDX9MediaAdapterKHR
- clCreateFromDX9MediaSurfaceKHR
- clEnqueueAcquireDX9MediaSurfacesKHR
- clEnqueueReleaseDX9MediaSurfacesKHR

Function prototypes for these APIs can be found in the **cl_DX9_media_sharing.h** file available from Khronos.org. [6] However, OpenCL v 1.2 has deprecated **clGetExtensionFunctionAddress** in favor of a new API function **clGetExtensionFunctionAddressForPlatform**, as described in the Khronos.org document

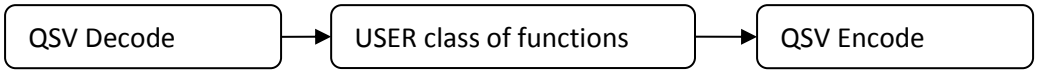
"OpenCL 1.2 Extensions Specification. [5]" **clGetExtensionFunctionAddressForPlatform** requires both a platform ID and the Khronos extension API name as a string.

For the KHR DX9 media surface sharing extension, as with the Intel DX9 media surface sharing extension, Intel Quick Sync Video media surfaces can only be shared into an OpenCL context created using the property **CL_CONTEXT_ADAPTER_D3D9EX_KHR** paired with a pointer to the IDirect3DDevice9 created for Intel Quick Sync Video.

For both Intel and KHR extensions, the OpenCL functions **clGetPlatformInfo** and **clGetDeviceInfo** can be used to get a list of named extensions supported by an OpenCL implementation, to verify that the desired extension is supported—either **cl_intel_dx9_media_sharing** for the Intel vendor-specific extension to OpenCL v1.0 or v1.1, or **cl_khr_dx9_media_sharing** for the OpenCL v1.2 Khronos standard extension. It is also worth noting that other DirectX surface sharing extensions have been created, and are also documented on the Khronos.org site.

Intel® Media SDK User Functions

Intel Media SDK provides a USER class of functions to allow user-defined functions to participate in Intel Quick Sync Video pipeline. This allows the application to integrate OpenCL based kernels into the Intel Quick Sync Video pipeline as a "plug-in." In this particular case, the USER class of functions is implemented to use OpenCL kernels.



The application needs to do the following to use an OpenCL plug-in inside Intel Media SDK:

- Initialize an OpenCL plug-in, registering a set of callback functions (see later) through the MFXVideoUSER_Register function. The Intel Media SDK invokes these callback functions at appropriate times in the Intel Quick Sync Video pipeline.
- Once initialized, the application can use the OpenCL plug-ins through the Intel Media SDK function MFXVideoUSER_ProcessFrameAsync to process data. The function returns a sync point for result synchronization, which is similar to other Intel Media SDK async functions.
- Close the OpenCL plug-in by unregistering it via the MFXVideoUSER_Unregister function.

The application needs to include "mfxplugin.h" in addition to other Intel Media SDK files.

The application needs to implement the following callback functions, which are registered through MFXVideoUSER_Register:

- PluginInit: Intel Media SDK calls this function to initialize the plug-in components and allocate internal resources
- PluginClose: Intel Media SDK calls this function to close the plug-in components and free the internal resources.
- GetPluginParam: Intel Media SDK calls this function to obtain plug-in configuration parameters.
- Submit: Intel Media SDK calls this function to check the validity of I/O parameters and submit a task to SDK for execution.
- Execute: Intel Media SDK calls this function to execute the submitted task after resolving all input data dependencies.
- FreeResources: Intel Media SDK calls this when task execution finishes or to cancel the queued task.

The Intel Media SDK kit includes a document, "Intel Media Software Development Kit – Extensions for User-Defined Functions API version 1.3 [7]" that provides the details about all these APIs.

Intel Quick Sync Video and OpenCL Media sharing Extension flow

There are multiple use cases in which Intel Quick Sync Video and OpenCL can interoperate.

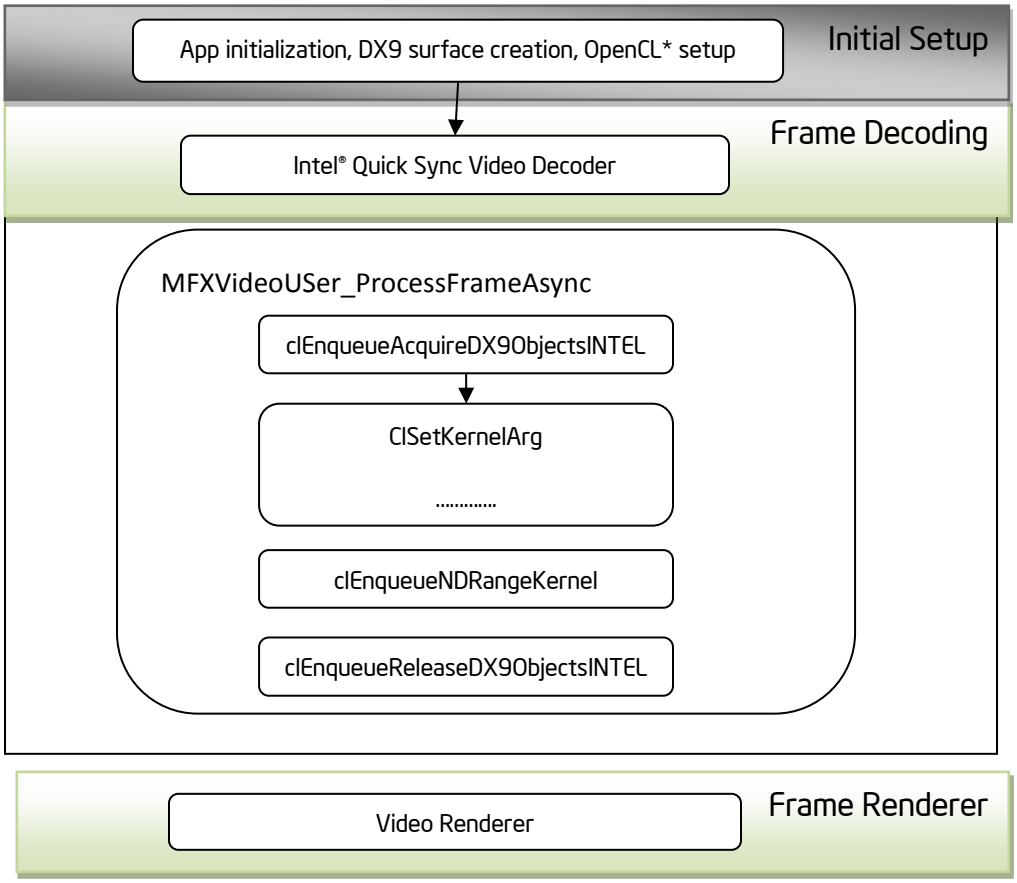


Figure 3: Intel® QSV Decoder and OpenCL* interoperability

For example, Figure 3 shows the flow of a decoder application, where an OpenCL filter is applied before displaying the frames. Figure 4 shows the flow of a video transcoding application that uses Intel Quick Sync Video to decode and encode the video. It applies some OpenCL filters to decoded video frames before encoding them.

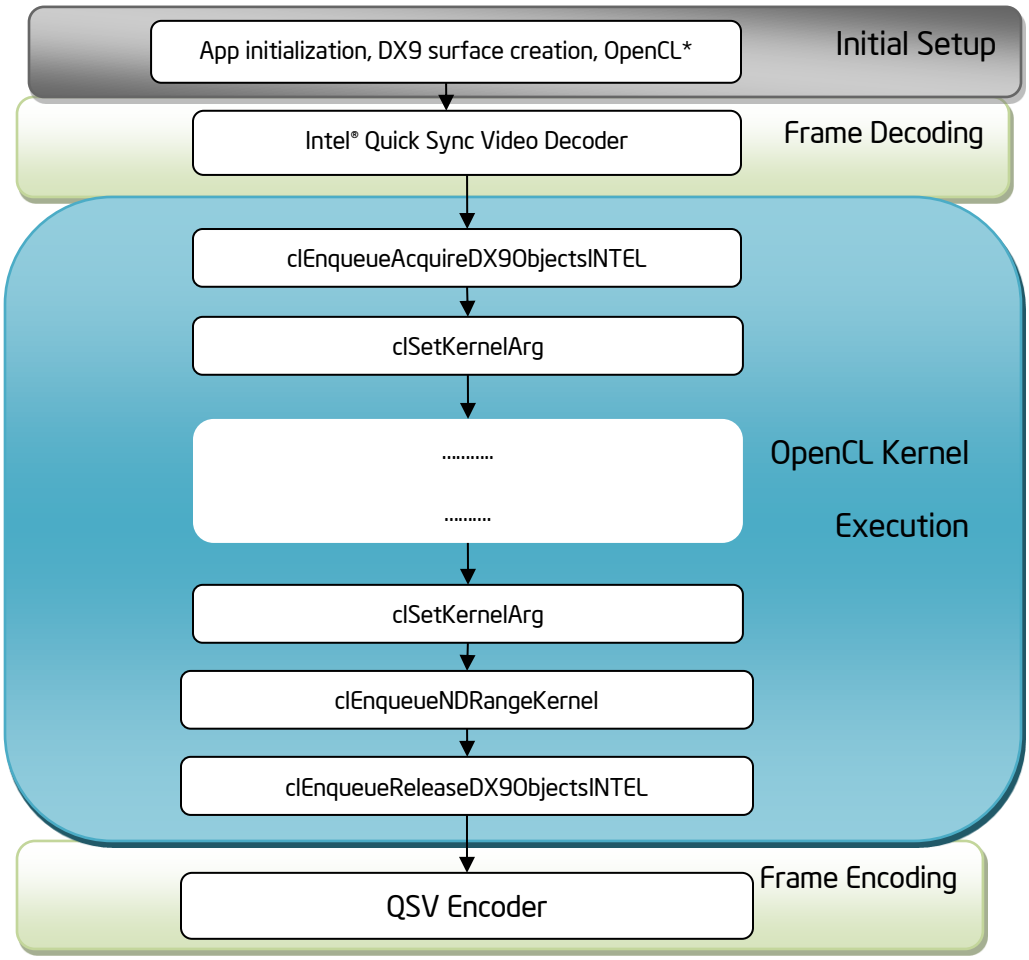


Figure 4: QSV and OpenCL* Interoperability

In each of these cases, the four distinct steps are: initial setup, frame decoding, OpenCL filters, and final video frame processing.

- Initial Setup: In this step the application must initialize Intel Quick Sync Video components using Intel Media SDK. It must also perform DX9 surface allocation. The application will use the Intel Media SDK API function **MFXVideoUSER_Register** to register the OpenCL plug-in. After that, the application can initialize other components and create the OpenCL surface using the **clCreateFromDX9MediaSurfaceINTEL** function and map to DX9 surfaces created by Intel Media SDK.

- Frame Decoding: After initializing, the application can start decoding frames. When a decoded frame is available, the applications calls the **MFXVideoUSER_ProcessFrameAsync** function. This stage could be left off if the application is only generating video frames for Intel Quick Sync Video encode.

- At this point Intel Media SDK calls the "Submit" and then "Execute" plugin callback functions to execute the OpenCL kernel. The "Execute" plug-in callback will implement the OpenCL host side part of the kernel. It will call **clEnqueueAcquireDX9ObjectsINTEL** to lock the frame, then standard OpenCL API functions like clSetKernelArgs and clEnqueueNDRangeKernel to run OpenCL kernels to process the frame, followed by **clEnqueueReleaseDX9ObjectsINTEL**.

- Frame Encoding / Frame Rendering: After frame processing, the DX9 surfaces are ready for the next stage, which could be encoding or display depending on the application.

Use Cases

OpenCL can be used to accelerate a range of video processing applications, and using the Intel or Khronos DX9 media sharing extensions will help insure the best possible performance by minimizing surface copy overhead.

Examples of applications using a combination of OpenCL and Intel Quick Sync Video include video editing, in which OpenCL is used to apply special effects or video transitions (such as blending two decoded video frames, and encoding the result); video playback with image enhancement prior to rendering for display; and algorithmic video synthesis, for example creating motion video by manipulating and animating still images and subsequently encoding the resulting sequence of frames.

Again, it is important to note that the minimum amount of image surface copies can be obtained only by having OpenCL accept and process frames in NV12 format from Intel Quick Sync Video, and/or deliver to Intel Quick Sync Video in the same format. If color conversion is required, for example NV12 to (or from) an RGB format, a developer may be able to merge that into the OpenCL processing pipeline as part of the first (or last) kernel to process the image2D object, thereby avoiding an added pass through memory. If Intel Quick Sync Video is asked to do the color conversion as part of its post processing, it will be a separate processing stage and additional pass through memory.

Example Code

The best example code for understanding how to integrate OpenCL-based plugins into a Intel Quick Sync Video pipeline is provided as sample code, available on the Intel® Developers Zone.[8] The Intel Media SDK code samples also include an OpenCL sample, but that sample does not use the DX9 media surface sharing extension.

About the Author

Tom Craver is an Intel application engineer in Chandler, Arizona. He is currently focused on performance of applications using OpenCL on Intel processor graphics, but has extensive background in SIMD, OpenCL and threaded parallel coding for performance, primarily for media applications such as audio and video codecs and video effect processing.

References:

1. **Intel® Media Server Studio** is available for download at https://software.intel.com/en-us/intel-media-server-studio/try-buy

2. **Intel® Integrated Native Development Experience** is available for download at https://software.intel.com/en-us/intel-inde

3. **OpenCL Specifications**, versions 1.0 and later, are available at www.Khronos.org under sub-sections "OpenCL" and "Specs & Headers"

4. **Intel extension for DirectX 9 Media Sharing** is at www.Khronos.org for OpenCL version 1.0 and newer, under the "OpenCL" and "Specs & Headers" subsections in the Extension Specifications section, listed as cl_intel_dx9_media_sharing

5. **OpenCL 1.2 Extensions Specification**, with the DirectX 9 Media Sharing extension for OpenCL version 1.2 and newer at www.Khronos.org under subsections "OpenCL" and "Specs & Headers", in the list of specification files for OpenCL v1.2 and newer

6. The file **cl_DX9_media_sharing.h** with the Khronos DirectX 9 Media Sharing extension for OpenCL version 1.2 and newer, is available at www.Khronos.org under subsections "OpenCL" and "Specs & Headers"

7. **Intel Media Software Development Kit – Extensions for User-Defined Functions** is distributed with the Intel Media SDK version 1.3 or newer, located in <install-folder>\doc\mediasdkuser-man.pdf

8. Various **Sample code** for integrating OpenCL into an Intel Media SDK pipeline is available at http://software.intel.com/en-us/vcsource/samples/opencl-and-intel-media-sdk and in the Media SDK samples linked to from the Intel Media SDK file download.htm located in <install-folder>\samples

9. **Intel Media Software Development Kit – Reference Manual** Version 1.3 or newer distributed with the Intel Media SDK and located in <install-folder>\doc\ mediasdk-man.pdf

Notices

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Intel and the Intel logo are trademarks of Intel Corporation in the US and/or other countries.

OpenCL and the OpenCL logo are trademarks of Apple Inc. and are used by permission by Khronos.

*Other names and brands may be claimed as the property of others.