Interoperability with Intel® Media SDK

Sample User's Guide

Intel® SDK for OpenCL Applications - Samples*

Document Number: 325994-004US

Contents

Contents 2
Legal Information 3
About the Interoperability with Intel® Media SDK 4
Introduction 4
Motivation 4
Algorithm 4
Implementation Details 4
OpenCL* Implementation 5
Work-Group Size Considerations 5
Project Structure 5
APIs Used 6
Controlling the Sample 6
References 6

Legal Information

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

About the Interoperability with Intel® Media SDK

The sample for Interoperability with Intel® Media SDK demonstrates how to use Intel® Media SDK and Intel® SDK for OpenCL* Applications together for efficient video decoding and fast post-processing.

Introduction

Intel Media SDK provides fast decoding of h264 and mpeg2 video streams, while Intel SDK for OpenCL Applications is good for pixel processing of video frames (for example OpenCL enables to easy implement pixel parallel algorithm). See Intel Media SDK website at http://software.intel.com/en-us/articles/media/. The Intel Media SDK Interoperability sample shows how to combine Intel Media SDK and Intel SDK for OpenCL Applications to perform fast/smooth video playback with additional post processing effects. Both SDK products are optimized for seamless interoperability. It is optimized for running on Processor Graphics device, which allows efficient data transfers from the Intel Media SDK surface to OpenCL memory object.

Motivation

The sample demonstrates an Intel Media SDK pipeline combined with simple (yet optimized) post-processing filters in OpenCL, showing how to:

- Integrate processing with Intel SDK for OpenCL Applications into Intel Media SDK pipeline and get benefit from hardware-accelerated (if available) video decoding with Intel Media SDK pipeline.
- Organize efficient sharing between MediaSDK frames and OpenCL images via `cl_khr_dx9_media_sharing` extension.
- Implement simple video effects in OpenCL.

Algorithm

The algorithm consists of the following stages:

1. Read the frame data from file.
2. Feed the data to the Intel Media SDK Decoder and get decoded video frame back in NV12 (which is a specific layout for YUV) format.
3. Execute the OpenCL kernel which processes the decoded video frame.
4. Draw image on the screen.

Implementation Details

The Intel Media SDK Interoperability sample demonstrates implementation of basic media processing pipeline. The pipeline consists of two main modules:

1. The **Decoder** module which leverages Intel Media SDK for hardware -assisted decoding mpeg2 and h264 formats.
2. The **Post-Processing** module which relies on Intel SDK for OpenCL Applications for efficient implementation of the video effects. This module actually serves as a plug-in for Intel Media SDK pipeline.

You should follow these requirements to be able to share dx9/DirectX Video Acceleration* surfaces with OpenCL:

1. Use the `Direct3D9Ex` object but not the `Direct3D9`.
2. Create the `dx9` surfaces as `DXVA2_VideoProcessorRenderTarget`.

OpenCL* Implementation

The Post-processing module introduced above is actually a plug-in for Intel Media SDK. The plug-in initializes Intel SDK for OpenCL* Applications and generates a list of filters scanning current folder and detecting all files with `*.cl` extension. Each file is a separate filter implementation.

Each file has at least one kernel which processes pixels. For example `ProcessUV` processes the 2x2 pixel at once that contains two color components and four intensities components processing. The sample contains the following `cl` filters:

- `_color_control.cl` - simple control of intensity and color characteristics inside circle controlled by mouse
- `_wave.cl` - simulates wave propagation and distorts image according to simulated waves.
- Files contain following kernels:
- `ProcessUV` – processes U and V component of four pixels of input frame
- `Mouse` – executes once for each frame to indicate where the mouse pointer is right now

When Intel Media SDK calls Intel SDK for OpenCL Applications plug-in to execute filter for given frame, `clEnqueueTask` and `clEnqueueNDRangeKernel` functions execute kernels from `*.cl` files. The host program keeps a sync event returned by these functions to check (by use of `clGetEventInfo`) whether the post-processing module is done with a frame.

For each frame the Post-Processing module creates OpenCL memory object wrapping the DirectX Video Acceleration surface. Use the `cl_khr_dx9_media_sharing` extension to avoid extra memory copying from the Intel Media SDK frame into Intel SDK for OpenCL Applications memory object.

Work-Group Size Considerations

You can specify any size of workgroup for this kernel aligned with vertical and horizontal sizes of input image in the range `1` to `imagewidth` and `1` to `imageHeight`. Current version use 8x8 local size that is ok with FullHD resolution video and Intel SDK for OpenCL Applications - Optimization Guide

Project Structure

The Intel Media SDK Interoperability sample project has the following structure:

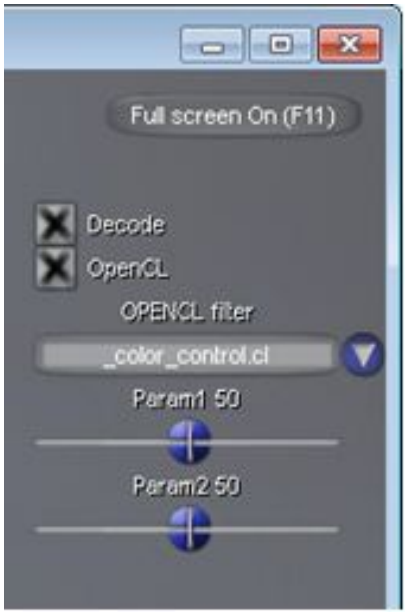
- `src` - folder contain main code, with Intel SDK for OpenCL Applications and Intel Media SDK initialization and processing functions
  - `main.cpp` – file initializes window and GUI controlled and draws the resulting frame to window using DXUT
  - `OCLStruct.cpp` – file contains OpenCL initialization code
  - `pipeline_decode.cpp` – file contains code which constructs processing pipeline for video decoding and post processing
  - `sample_opencl_plugin.cpp` – file contains code of Intel Media SDK plug-in for post processing video frames using Intel SDK for OpenCL Applications
- `src_MediaSDK` – folder contains memory management files of Intel Media SDK samples
  - `base_alllocator.cpp` – base memory allocators
  - `d3d_allocator.cpp` – specific memory allocators which allocate buffers and frames using dx9 surface
  - `sample_defs.h` – some macros and defines used from Intel Media SDK APIs.

APIs Used

The Intel Media SDK Interoperability sample uses the following APIs [3]:

- `clGetDeviceIDs`
- `clCreateContext`
- `clReleaseContext`
- `clCreateCommandQueue`
- `clReleaseCommandQueue`
- `clGetDeviceInfo`
- `clCreateProgramWithSource`
- `clBuildProgram`
- `clReleaseProgram`
- `clGetProgramBuildInfo`
- `clKreateKernel`
- `clReleaseKernel`
- `clReleaseMemObject`
- `clSetKernelArg`
- `clEnqueueTask`
- `clEnqueueNDRangeKernel`
- `clReleaseEvent`
- `clFlush`
- `clGetDeviceIDsFromDX9MediaAdapterKHR`
- `clCreateFromDX9MediaSurfaceKHR`
- `clEnqueueAcquireDX9MediaSurfacesKHR`
- `clEnqueueReleaseDX9MediaSurfacesKHR`

Controlling the Sample



- **Decode** check box switches Intel Media SDK Decoder on/off
- **OpenCL** check box switches OpenCL post processing effect on/off
- **OPENCL filer** combo box enables you to choose current OpenCL program
- **Param1** and **Param2** sliders enable you to change parameters passed to OpenCL kernels
- Press **F11** key to switch on/off full screen mode
- Press **F2** key to show/hide currently executed OpenCL source code.

References

- http://software.intel.com/en-us/articles/media/
- http://software.intel.com/en-us/vcsource/tools/media-sdk
- http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf
- http://www.khronos.org/registry/cl