Interoperability between OpenCL* and Microsoft Direct3D*

Sample User's Guide

Intel® SDK for OpenCL* Applications - Samples

Document Number: 329766-003US

Contents

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 Interoperability between OpenCL* and Microsoft Direct3D*

OpenCL* and Microsoft Direct3D* 10 API Interoperability sample demonstrates how to use the Microsoft Direct3D and Intel® SDK for OpenCL Applications together for data processing and real-time rendering.

The sample demonstrates creating a heterogeneous pipeline using OpenCL for processing and Direct3D for real-time rendering in a 3D environment.

Introduction

Microsoft DirectX* SDK provides a hardware-accelerated 3D graphics environment used in gaming and visualization applications. The OpenCL and Microsoft Direct3D 10 API Interoperability sample uses OpenCL for computation of a particle buffer projection and Direct3D graphical capabilities to render the particles in real-time.

Motivation

The OpenCL and Microsoft Direct3D 10 API Interoperability sample demonstrates an OpenCL implementation of multi-dimensional projections, showing how to:

- Integrate data processing using the Intel SDK for OpenCL Applications with Microsoft DirectX* in order to develop a heterogeneous API pipeline.
- Utilize processing power for large datasets and DirectX for real-time 3D rendering.

Algorithm

The algorithm consists of the following stages:

1. Create shared Direct3D/OpenCL memory buffers using the `cl_khr_d3d10_sharing` OpenCL extension, and populate with initial data.
2. Initialize Direct3D environment for real-time graphics.
3. Use OpenCL to compute the distances between all pairs of points in their original multi-dimensional space, utilizing a cache of local memory to optimize this computation on the Intel Processor Graphics device.
4. Use OpenCL to compute the linear projection of the particle buffer by multiplying the multi-dimensional points by a linear projection matrix. The points are now projected to 3-dimensional space.
5. Use OpenCL to compute the distances between all pairs of points in their projected 3-dimensional space, again utilizing a cache of local memory to optimize this computation on the Intel Processor Graphics.
6. Use OpenCL to calculate the difference between the pair-wise distances in the original multi-dimensional space with the pair-wise distances in the projected 3-dimensional space. This is a widely used metric for linear projection error.
7. Render the projected 3D points in the Direct3D environment, using the geometry shader to generate particle sprites on-the-fly.

OpenCL* Implementation

The first part of the application utilizes OpenCL* to project multi-dimensional points to 3 dimensions, and stores the resulting points in a buffer shared by Direct3D which allows them to be rendered in a

real-time 3D environment. After the projection is calculated, the error introduced by projecting the points to lower-dimensional space is calculated and outputted to the console. Projection error, in this case, is defined as the difference between high-dimensional relative distances and low-dimensional relative distances. In order to do this, we calculate the differences between all pairs of points before they are projected, and again after they are projected. We then take the difference between these two buffers of pair-wise distances to get the projection error.

- `Projection.cl` – Computes the linear projection of a multi-dimensional point to 3-dimensional space by multiplying each multi-dimensional point by a projection matrix
- `RelativeDistanceError.cl` – Calculates the distance between each pair of points in high-dimensional space and stores the scalar distance values in another buffer

The OpenCL pipeline first projects each individual point by the projection matrix, then produces two distance matrices representing scalar distances between each pair of points in high-dimensional space and 3-dimensional space. The final kernel calculates the difference between the two scalar distance buffers. This final output is representative of the projection error as a result of the projection matrix.

Local Memory Caching

We use a buffer of local memory to cache blocks of particles and quickly calculate the scalar distances between pairs of particles. We employ the same methods used in GPU Gems 3 extended to OpenCL for Intel, see http://http.developer.nvidia.com/GPUGems3/gpugems3_ch31.html for more information. Blocks of particles are loaded into a local memory cache, whereby their information can be accessed extremely quickly. The distances between all pairs of particles in the local memory block are calculated efficiently, and a new block of particles is loaded into the local memory cache. The process is repeated until all pair-wise distances between particles have been calculated.

Project Structure

The OpenCL and Microsoft Direct3D 10 API Interoperability sample project has the following structure:

- `D3D10BufferSample.cpp` – file contains body of `c++` code which loads OpenCL kernels, initializes Direct3D environment, and runs the main program loop.
- `FreeCamera.cpp` – file contains free moving camera class to navigate in Direct3D environment
- `FreeCamera.h` – header file for FreeCamera.cpp
- `ParticleDraw.fx` – file contains HLSL shader code to transform a vertex and create a textured sprite to visualize it
- `Projection.cl` – file contains OpenCL kernel to project multi-dimensional points by a linear transformation matrix
- `RelativeDistError.cl` – file contains OpenCL kernels to calculate pair-wise distances between particles and differences between distance buffers.

Controlling the Sample

Navigate the camera using keys W, A, D, S as well as clicking and dragging the left-mouse button. Press the Shift key while moving to move faster.

References

- http://msdn.microsoft.com/en-us/directx/
- http://http.developer.nvidia.com/GPUGems3/gpugems3_ch31.html
- http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf