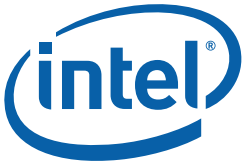


High Dynamic Range Tone Mapping Post Processing Effect

Intel® OpenCL SDK Sample Documentation

Document Number: 325396-002US

Legal Information

Optimization Notice

Intel compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel compilers, associated libraries and associated development tools 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 Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (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.

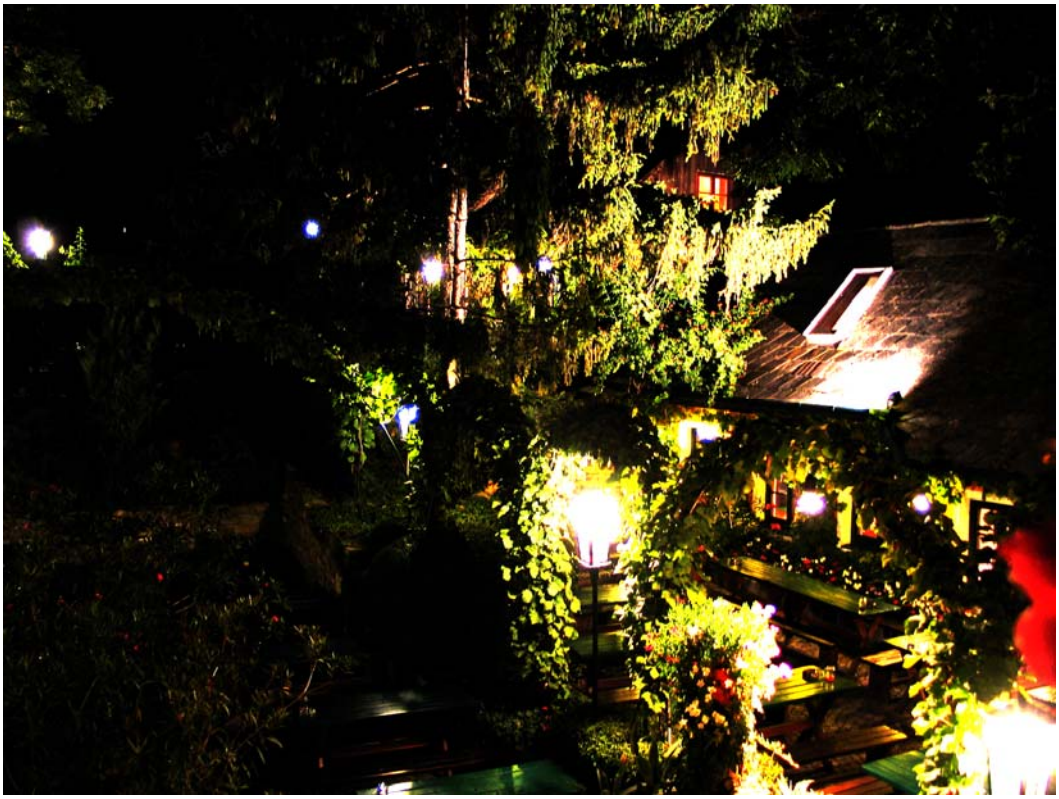
While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20110307

Contents

About Tone Mapping

Tone Mapping sample demonstrates how to use high dynamic range (HDR) rendering with tone mapping effect in OpenCL™. The following figures illustrates straightforward linear tone mapping and advanced tree-component curve based tone-mapping technique proposed by OpenEXR* community [1] applied to HDR image:



Path

Location	Executable
`<INSTALL_DIR>samples\ToneMapping`	`Win32\Release\ToneMapping.exe` – 32-bit executable `x64\Release\ToneMapping.exe` – 64-bit executable `Win32\Debug\ToneMapping.exe` – 32-bit debug executable `x64\Debug\ToneMapping.exe` – 64-bit debug executable

Introduction

The real world scenes we experience in our daily life often have a very wide range of luminance values. Human visual system is capable of perceiving scenes over five orders of magnitude and can gradually adapt to scenes with dynamic ranges of over

nine orders of magnitude. With the rapid advancement of digital imaging technology, there is increasing interest in taking digital photographs that capture the full dynamic range of the scene of view. Although it is conceivable that future digital cameras would be able to capture high dynamic range (HDR) photos by the click of a button, current technology often only enables part of the real world high dynamic scene visible in any one single shot. [2]

The main aim of tone-mapping procedure is mapping HDR image to low dynamic range (LDR) device (computer monitor) perceiving as much details as possible.

For high dynamic range mapping, there are at least two requirements. Firstly, it has to ensure that all features, from the darkest to the brightest, to be visible simultaneously. Secondly, it has to preserve the original scene's visual contrast to produce a visual sensation matching that of the original scene. In a sense, these two are conflicting requirements. With a reduction in dynamic range, the available values for displaying the scene are limited. If one makes all features visible, we may lose contrast. On the other hand, if one makes the display well contrast, then some features may not be visible. A good tone reproduction method has to strike a good balance between these two conflicting requirements under the constraint of limited available display dynamic range. [2]

This sample implements HDR image display algorithm proposed by OpenEXR* [1] to implement tone mapping effect.

Motivation

An implemented tone mapping effect requires many complex math calculations performed over floating point HDR pixel values. As well the algorithm contains 2 separate branches depending on pixel value. This sample implementation uses data level parallelism on pixel level (SIMD instructions over RGBA pixel channels) and image level (processing image tiles in separate tasks in parallel).

This sample demonstrates a CPU-optimized implementation of the tone mapping effect, showing how to:

- implement calculation kernels using OpenCL C99

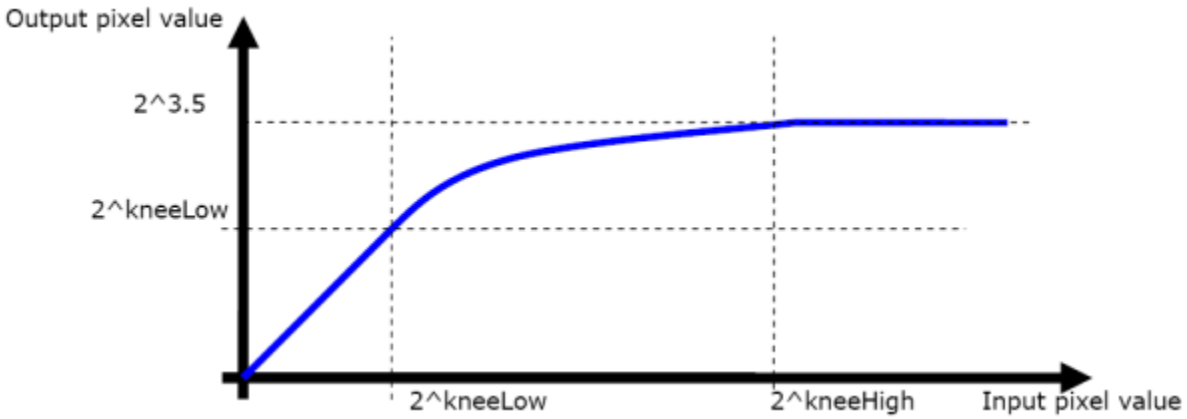
- parallelize the kernels by running several work-groups in parallel
- organize data exchange between the host and the OpenCL device
- store the final image on the hard drive.

Algorithm

The original algorithm [1] consists of the following stages:

1. Compensate for fogging by subtracting defog from the raw pixel values.
2. Multiply the defogged pixel values by $2^{(\text{exposure} + 2.47393)}$.
3. Values, which are now 1.0, are called "middle gray." If defog and exposure are both set to 0.0, then middle gray corresponds to a raw pixel value of 0.18. In step 6, middle gray values will be mapped to intensity 3.5 f-stops below the display's maximum intensity.
4. Apply a knee function. The knee function has two parameters, kneeLow and kneeHigh. Pixel values below 2^{kneeLow} are not changed by the knee function. Pixel values above kneeLow are lowered according to a logarithmic curve, such that the value 2^{kneeHigh} is mapped to $2^{3.5}$ (in step 6, this value will be mapped to the display's maximum intensity).
5. Gamma-correct the pixel values, assuming that the screen's gamma is 2.2.
6. Scale the values such that middle gray pixels are mapped to 84.66 (or 3.5 f-stops below the display's maximum intensity).
7. Clamp the values to [0, 255].

Following image illustrates underlying HDR->LDR 3 component transfer function:



OpenCL™ Implementation

This sample applies above described stages of the tone mapping algorithm to an HDR image.

Code Highlights

The `ToneMapping` OpenCL™ kernel of the `ToneMapping.cl` file performs the tone mapping effect. Every input image raw has a unique global ID that the kernel uses for their identification. The tone mapping effect sequence consists of OpenCL kernel call performed in `ExecuteToneMappingKernel()` function of `ToneMapping.cpp` file.

Work-group Size Considerations

You can specify any work-group size for this kernel aligned with vertical size of input image in the range 1 – imageHeight/2. However, the kernel achieves peak performance for 1600x1200 two-dimensional HDR image with work-group size ranging from 1 to 16 elements.

Understanding OpenCL Performance Characteristics

Benefits of Using Vector Data Types

This sample implements the tone mapping effect algorithm using vector data types. Explicit usage of vector types, such as `float4`, enables the following CPU optimizations:

- You can work with quadruples instead of single floats. This removes unnecessary branches, saves memory bandwidth, and optimizes CPU cache usage.
- You can use tone mapping effect for a single four-color channels pixel item. Consequently, you can perform tone mapping effect for four-color channels of an image pixel (RGBA pixel) and for four monochrome pixels simultaneously. The current version uses vector `float4` data types.

Limitations

The current implementation uses global tone mapping operator which don't cares about local lighting conditions for various part of the input image. To enhance output result more complex operator with neighboring pixels brightness analysis need to be implemented.

Future Work and Enhancements

The sample performs all calculations in floating-point values. Each image pixel consists of four 32-bit floating-point values representing red, green, blue, and alpha (RGBA) image channels. You can improve this sample performance by introducing the following:

- more compact data representation (half float)
- auto-adjusting or tone-mapping parameters for the whole image/frame (auto-exposure)

- replacing a global tone mapping operator with a local tone mapping operator to adjust its parameters according to the local lighting conditions on the image/frame.

Project Structure

This sample project has the following structure:

- `ToneMapping.cpp` - the host code, with OpenCL initialization and processing functions
- `ToneMapping.cl` – source code of the OpenCL tone mapping kernel
- `ToneMappingNative.cpp` – source code of the native tone mapping kernel implementation (SIMD)
- `ToneMapping.vcproj` – Microsoft* Visual Studio* 2008 project file.

APIs Used

This sample uses the following APIs:

- `clKreateKernel`
- `clCreateContextFromType`
- `clGetContextInfo`
- `clCreateCommandQueue`
- `clCreateProgramWithSource`
- `clBuildProgram`
- `clCreateBuffer`
- `clSetKernelArg`
- `clEnqueueNDRangeKernel`
- `clEnqueueReadBuffer`
- `clReleaseMemObject`
- `clReleaseKernel`
- `clReleaseProgram`
- `clReleaseCommandQueue`
- `clReleaseContext.`

Reference (Native) Implementation

Reference implementation is done in `ExecuteToneMappingReference()` routine of `ToneMapping.cpp` file. This is single-threaded code that performs exactly the same tone mapping effect sequence as the OpenCL implementation, but using conventional nested loop in C with SSE optimizations. Native kernel `EvaluateRaw()` that processes input HDR image is located in `ToneMappingNative.cpp`.

Controlling the Sample

The sample executable is a console application. The current implementation has no command line arguments.

References

[1] http://www.openexr.com/using.html

[2] "Fast Tone Mapping for High Dynamic Range Images" by Jiang Duan and Guoping Qiu