Second Assignment
GPGPU & Accelerator Programming
COMP10065

Issue Date: Monday, March 6th, 2017
Due Date: **5pm, Friday, March 31st, 2017**

Accelerating the Unsharp Mask

The unsharp mask is a method of sharpening an image, simply by subtracting a blurred version of the original. In this assignment you will take an existing serial C++ unsharp mask code, and attempt to improve its performance using an appropriate GPGPU language or API (though not CUDA). For example, you may consider Thrust, OpenCL, SYCL [1] , or C++ AMP. Describe your approach, and illustrate your timing and other results.

You should work individually. Submit the code you develop, along with a pdf report of approximately 2000 words; as a single zip to Moodle by the date noted above.

Marking Scheme

The assignment is worth 60% of the marks awarded for the entire COMP10065 module. The following provides a breakdown of the marking scheme:

Degree of speedup over serial version	30%
Interesting development approaches (clearly explained)	10%
Creation of the sharpened image file	10%
Quality of code	20%
Written report	20%
Use of figures in report	10%

The Serial Unsharp Mask Code

The serial unsharp mask code is available through the module's Moodle page, and consists of a single C++11 source file, `unsharp_mask.cpp`, and three header

[1] A lack of speedup results for a SYCL VM solution can be discussed with the lecturer.

1	1	1
1	1	1
1	1	1

Figure 1: The kernel of a (radius 2) 3×3 box linear filter

files; `ppm.hpp`, `blur.hpp` and `add_weighted.hpp`. The program calculates the unsharp mask of an ASCII 24-bit PPM image file, provided via the first input argument; before saving the result using the file path specified by the second argument. A third argument, an integer, is used to control the radius of the box blur filter component of the unsharp mask operation; see Figure 1. This is a *crucial* parameter to vary and consider while exploring GPGPU performance improvements.

Resources

In addition to the C++ code, you are provided with a 3840×2160 resolution image file, `ghost-town-8k.ppm`; stored within the 7-zip archive: `ghost-town-8k.7z`. This file is of a suitable size to investigate changes in your program's runtime as you extend it.

Methodology

The GPGPU program you develop should maintain the ternary programmatic interface outlined above; i.e. file in; file out; blur radius. Otherwise, you are free to make substantial changes to the original program. Coarse timing code is included, though you may also report more detailed timings. Note that the time to copy data to and from the GPU device memory should not be ignored. Aim to demonstrate excellent performance with a *range* of blur-radius values.

Suggestions

Consider the following as you develop your solution:

- Does packing each RGB datum within 32-bits rather than 24-bits improve performance?
- Explore MSVC++ options such as those under `C/C++` and `Optimization`
- Keep in mind that there are GPU device event-profiling mechanisms
- Why is the box blur applied three times? Would a *gaussian* blur improve the quality of the sharpened image?
- You may consider generating your own PPM test files at different resolutions

- Which high-level parameters of the GPGPU language/API can affect performance?

Advanced

The following are advanced topics, but may be explored by students aiming for the very highest marks:

- Utilise shared/local memory as a cache for data-reuse
- Exploit fixed-function graphcs hardware via OpenCL images and bilinear filtering
- Interactive graphical visualisation of varying blur radii