**Assignment 4**

Multiprocessor Programming 521288S

Miika Sikala 2520562, msikala18@student.oulu.fi

**Task:** Sequential Implementation of Stereo Disparity Algorithm (in C/C++) and profiling

**Expected results:** A working version of the implementation, a brief report (max1-2 pages), saved output images all together in the form of a compressed folder (.zip file)

The report should contain about the task solved, brief description of your implement and the final screenshot of your outputs asked to be displayed under the assignment.

Detailed instructions regarding the task are provided under assignment\_4.pdf. Kindly go through them once and get back to me in case there are any questions.

**Introduction**

NOTES:

* If we used 1-channel greyscale image vectors, the calculation would be significantly faster
* Resizing is obviously very simplistic; we merely skip each 4th pixel! ~~We should use some sort of interpolation...~~ Now we do - use mean filter.
* The resize operation shifts the whole image 1 pixel up and 1 left. That may be an easy fix but it is so insignificantly small that it most certainly does not affect the outcome.

In this task, I gave a little bit more structure for the program to make development easier to manage. The implementation consists of few classes that are used (NOTE: all these classes are implemented by me).

**PerfTimer:** Wrapper for windows QPC to measure execution times on sub-microsecond scale.  
**Image:** Holds the image data and has an ability to load, save, convert to grayscale, and filter images. Loading and saving is done using *lodepng*.  
**MiniOCL:** A simple wrapper for OpenCL C library to make interfacing with it much easier. Used by Image class to perform parallel execution when target device is not TARGET\_NONE.

The program outputs execution times of each operation. The kernel execution time is measured separately using OpenCL events.

There is a filter\_t struct that can be used to make custom filters. The struct requires the kernel (mask), mask size and the common divisor that is applied to the kernel. The implementation includes 5x5 mean filter (meanFilter) and 5x5 gaussian filter (gaussianFilter) which can be found in main.cpp.

After converting the image to grayscale, the program saves the grayscale image as “img/gray.png” and after filtering, “img/filtered.png”.

**Usage**

**To change the target device**, you can change the COMPUTE\_DEVICE flag in the beginning of main.cpp. The options are TARGET\_NONE (no parallelization), TARGET\_GPU (OpenCL on GPU) and TARGET\_CPU (OpenCL on CPU). You must re-compile the program for the change to take effect.

The application takes one input parameter, which is the image file name. See the example below, which uses image file img/im0.png.

To change the filter to use, you can change the argument of img.filter(yourFilter). The filter must be of type filter\_t.

Compiling using *g++*:  
g++ main.cpp MiniOCL.cpp lodepng.cpp %OCL\_ROOT%/lib/x86\_64/opencl.lib -Wall -I %OCL\_ROOT%\include -o stereo-disparity.exe

Running:  
stereo-disparity.exe img/im0.png img/im1.png

**Testing and benchmarking the implementations**

The correctness of the implementation was tested by checking the results manually. The screenshots of the outputs of the program are shown below. The resulting images are found in “img” folder and partly in Figure 4.

Figure 1 shows the execution on CPU without OpenCL. Figures 2 and 3 show the execution of the OpenCL programs on CPU and GPU, respectively. The outputs also show the execution time and device information. Figure 4 shows a detail of the processed image.

Text

Description automatically generated

Figure . Output of the program without using OpenCL.

Text

Description automatically generated

Figure . Output of the program using OpenCL on CPU.

Text

Description automatically generated

Figure . Output of the program using OpenCL on GPU.

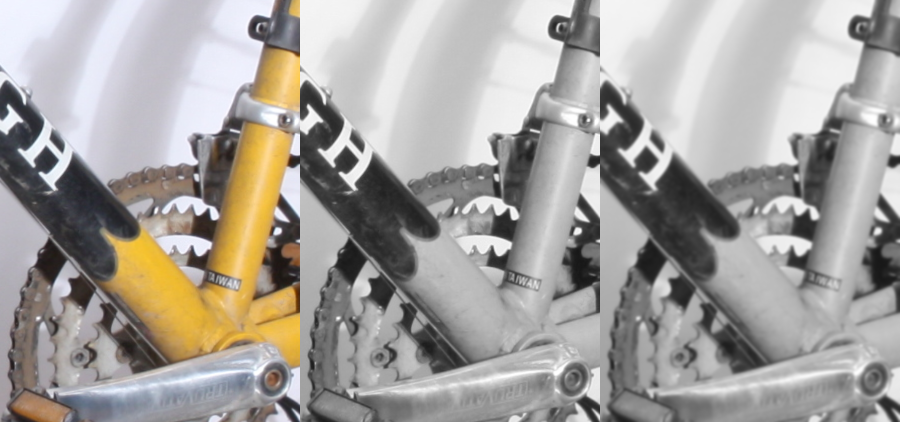


Figure . A detail from the image. From left to right: original image, grayscale, filtered image (5x5 gaussian).

Execution times:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **CPU (sequential)** | **CPU (OpenCL)** | **GPU (OpenCL)** |
| Load + decode | 1.068 s | 1.072 s | 1.064 s |
| Convert to grayscale | 425.390 ms | 79.090 ms | 0.264 ms |
| Filter | 2952 ms | 394.087 ms | 1.114 ms |
| Encode + save | 3.037 s | 2.968 s | 2.965 s |
| Total | 7.482 s | 4.513 s | 4.030 |

As the benchmark shows, the load and save times are pretty much the same, since it is sequential operation in all cases. However, in filtering and grayscale conversion, there is a significant performance difference when using OpenCL, especially GPU.

**Discussion**

While the implementation is quite flexible and well-performing, we could improve performance in the image filtering slightly; since we are filtering grayscale images, all channels except alpha (RGB) have the same value. This would allow filtering a single channel as a simple vector, which would speed up the computing a bit. However, this implementation allows for more flexible operations, as it works well for non-grayscale images as well.

The OpenCL implementations use edge clamping (via sampler\_t), but the sequential implementation does not, which results in a slight darkening on the very edge of the image.

**References**

[1] <https://johnwlambert.github.io/stereo>

[2] *The OpenCl Programming Guide* and *The* *OpenCL Specification 1.2*

Reporting

|  |  |
| --- | --- |
| **Task** | **Hours** |
| C implementation | 15 h |
| OpenCL implementation | 20 h |
| **Total** | **35 h** |