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 faster.
- 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.

The main structure of the program is similar to the previous assignment, the `Image` class being the most important part.

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

Downscale: First apply a dynamically created mean filter to the image and then simply downsample the image taking every Nth pixel, depending on the factor.

CalcZNCC: Calculates the disparity map.

CrossCheck: Cross checks the two disparity maps using threshold 8 (can be adjusted).

OcclusionFill: Fills the empty pixels with the first non-zero value on the left. Different approaches were experimented with.

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 two input parameters, which are the image file names for the left and right images. See the example below, which uses image files `img/im0.png` and `img/im1.png`.

A makefile is provided for compiling with g++.

<u>Compiling using *g++*:</u>

```
make
```

<u>Running:</u>

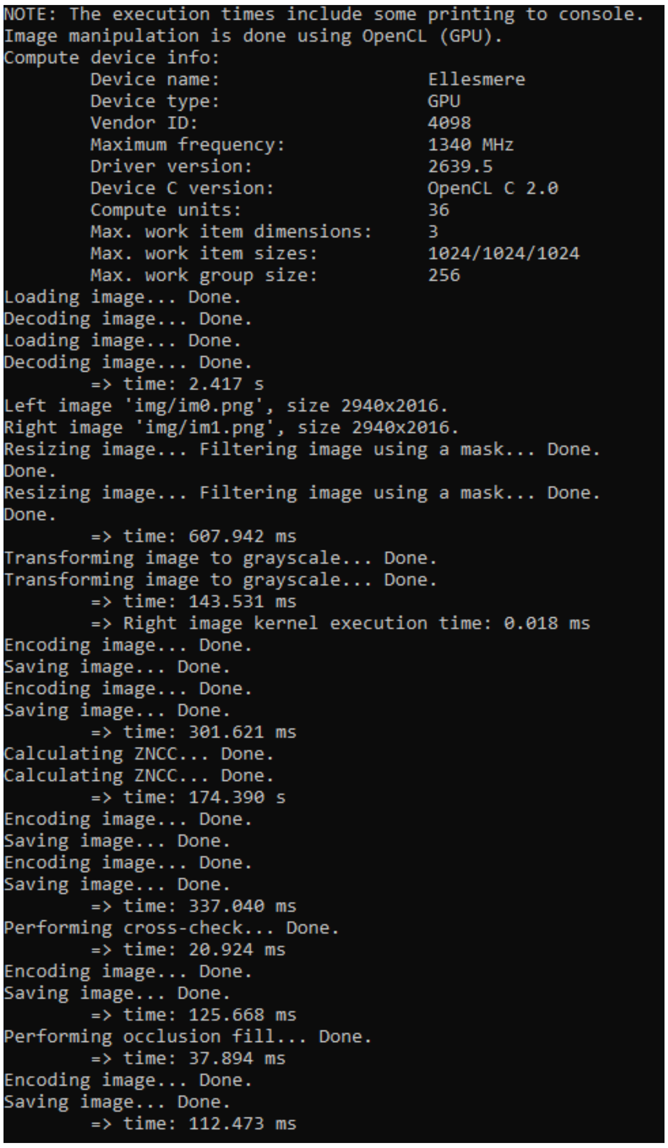
```
stereo-disparity.exe img/im0.png img/im1.png
```

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 ().

Figure 1 shows the execution on CPU. Th grayscale conversion and filtering were done using OpenCL in GPU, but other parts are done sequentially. The settings used are:

maximum search distance:	55
window size:	15
cross-check threshold:	8



```
NOTE: The execution times include some printing to console.
Image manipulation is done using OpenCL (GPU).
Compute device info:
        Device name:                    Ellesmere
        Device type:                    GPU
        Vendor ID:                      4098
        Maximum frequency:              1340 MHz
        Driver version:                 2639.5
        Device C version:               OpenCL C 2.0
        Compute units:                  36
        Max. work item dimensions:      3
        Max. work item sizes:           1024/1024/1024
        Max. work group size:           256
Loading image... Done.
Decoding image... Done.
Loading image... Done.
Decoding image... Done.
        => time: 2.417 s
Left image 'img/im0.png', size 2940x2016.
Right image 'img/im1.png', size 2940x2016.
Resizing image... Filtering image using a mask... Done.
Done.
Resizing image... Filtering image using a mask... Done.
Done.
        => time: 607.942 ms
Transforming image to grayscale... Done.
Transforming image to grayscale... Done.
        => time: 143.531 ms
        => Right image kernel execution time: 0.018 ms
Encoding image... Done.
Saving image... Done.
Encoding image... Done.
Saving image... Done.
        => time: 301.621 ms
Calculating ZNCC... Done.
Calculating ZNCC... Done.
        => time: 174.390 s
Encoding image... Done.
Saving image... Done.
Encoding image... Done.
Saving image... Done.
        => time: 337.040 ms
Performing cross-check... Done.
        => time: 20.924 ms
Encoding image... Done.
Saving image... Done.
        => time: 125.668 ms
Performing occlusion fill... Done.
        => time: 37.894 ms
Encoding image... Done.
Saving image... Done.
        => time: 112.473 ms
```

Figure 1. Execution of the program.

Execution times:

Operation	CPU (sequential)	GPU (OpenCL)
Load + decode	2.417 s	-
Resize (1/4x)	607.942 ms	-
Convert to grayscale	-	0.264 ms
Encode + save	301.621 ms	-
Calculate ZNCC (both)	**174.390 s**	-
Cross-check	20.924 ms	-
Occlusion fill	37.894 ms	-
Total	177.775 s	

Notice that the ZNCC calculation took extremely long time, due to having sequential implementation.

Figure 2 shows the resulting image.

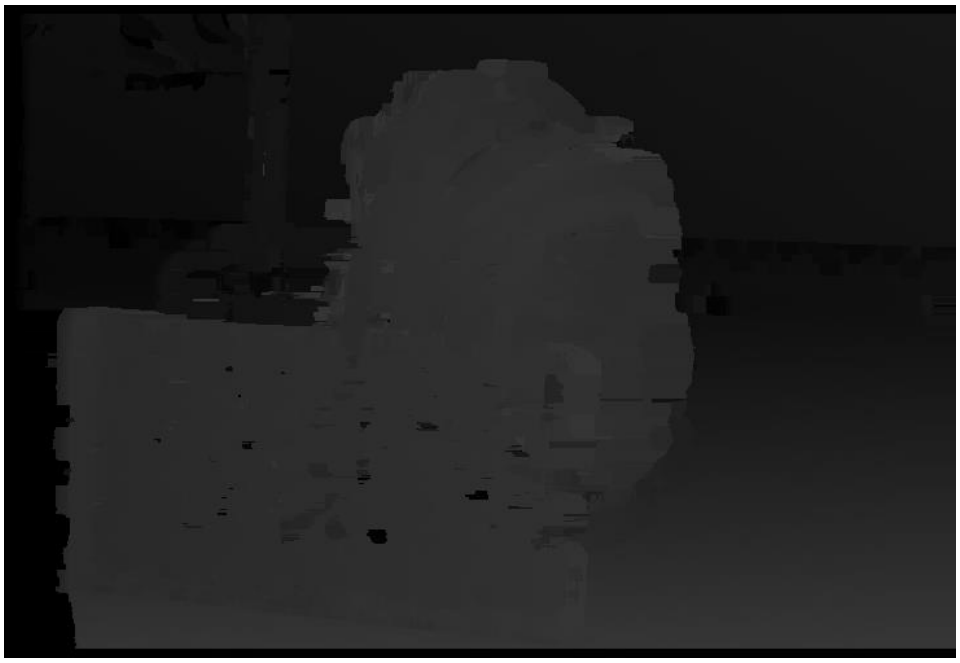


Figure 2. Resulting image.

Memory

The memory consumption was recorded using Visual Studio debugger. The maximum memory was consumed during ZNCC calculation, which required the left image and right image, as well as the disparity map to reside in the memory at the same time. This consumed total of **254 MB**.

CPU usage during ZNCC was 12,5 %. This is very inefficient. Since I have 8-core CPU and the implementation only used 1 core, a parallel execution would significantly speed up the calculation.

Discussion

While I experimented with many different options and found a satisfying setup, there are many parts that can be optimized in the future. For example, resizing and many other parts could easily be done using OpenCL. Also, different occlusion fill algorithms, as well as different parameters (maximum search distance, window size) could be tested more rigorously.

Reporting

Task	**Hours**
C implementation	32 h
Total	**32 h**