Implementation of ZNCC algorithm for disparity map computation using OpenCL

Aleksei Tiulpin
Center for Machine Vision and Signal Analysis
University of Oulu

Iaroslav Melekhov
Department of Computer Science
Aalto University

Abstract—In this report we describe an implementation of zero mean cross-correlation algorithm for stereo correspondence search. In particular, we show the accelerated implementation of the algorithm using OpenCL v.1.2 standard. The performance gain which we managed to achieve with the accelerated version of the algorithm is around 100 times.

I. INTRODUCTION

Estimation of a disparity map of an image sequence plays important role in such large areas of computer vision as multi-view image generation, 3D reconstruction from stereo image pairs and object recognition [1], [2]. Obtaining a precise and accurate depth map is the ultimate goal in these applications.

By the definition, binoclular disparity is positional difference between the two retinal projections of a given point in space. This positional difference is a result of the fact that the two eyes are laterally separated and therefore see the world from two slightly different vantage points [3].

Many different approaches have been developed to estimate the disparity field [1], however in this work we focus on one of the area-based algorithm for the disparity map estimation – zero mean cross correlation (ZNCC) [4]. In addition, we will show how to apply several post-processing techniques in order to compute the depth map out of two disparity fields.

The benefit of the ZNCC algorithm is a simple and straightforward implementation, which is moreover well suited for data-parallel acceleration using Graphical Processing Units (GPU). In our work, OpenCL standard v.1.2 has been used for this purpose. In addition, we have performed a series of experiments and optimized our implementation for NVidia GPU.

The report is organized as follows. In Section II ZNCC algorithms implemented in C language is presented. Section III describes the proposed method of calculating disparity map based on OpenCL technology. Section V presents benchmarks of the algorithm evaluated on two different GPU. In Section IV we briefly discuss about OpenCL optimization and a way how to improve performance further. In the end of this report we summarize our results.

II. ONE THREAD IMPLEMENTATION OF ZNCC ALGORITHM

A. Are-based correspondence search

The goal of the correspondence search is to find the corresponding pixel in the right image (a match) for the given

(a) left image   (b) right image

Fig. 1: Given images for disparity map calculation

pixel on the left image. Usually, comparing just pixels is not enough, therefore, their neighborhoods are compared.

In the area based matching algorithms, rectangular blocks of pixels from a set of two images $W \times H$ (left and right images) are compared and matched. We conducted our experiments for the images presented in Fig. 1. For each block of the left image, designated by reference window, a corresponding block in the right image is sought. During the search process, the correspondent block of the right image is moved with a shift parameter d and compared with the left block using some similarity measure $\otimes$. In this work we will consider disparity only on a horizontal axis, therefore, there is now disparity any shift parameter on the vertical axis.

Let $\mathbf{L}_{xy}$ and $\mathbf{R}_{xy}$ be blocks of left and right images respectively, centered in some point (x, y) and $C_{xy}(d)$ – a cost function of matching blocks $\mathbf{L}_{xy}, \mathbf{R}_{xy}$, defined by a similarity measure $\otimes$. The disparity field $\mathbf{D}$ is estimated then as

$$\mathbf{D} = \{d_{xy}(d) : 0 \leq x < W; 0 \leq y < H\}, \qquad (1)$$

where

$$d_{xy}(d) = \arg\max_{d} C_{xy}(d), \qquad (2)$$

$$C_{xy}(d) = \mathbf{L}(u, v) \otimes \mathbf{R}(u + d, v). \qquad (3)$$

Many different similarity measures $\otimes$ have been referred in the literature [1]. The reasons for a particular selection are usually related to the computational load, achieved performance and algorithmic simplicity. In this work we use ZNCC similarity measure:

$$ZNCC_{xy}(d) = \frac{\left(\mathbf{L}_{xy} - \overline{\mathbf{L}_{xy}}\right) * \left(\mathbf{R}_{(x-d)y} - \overline{\mathbf{R}_{(x-d)y}}\right)}{\|\mathbf{L}_{xy} - \overline{\mathbf{L}_{xy}}\|_2 \|\mathbf{R}_{(x-d)y} - \overline{\mathbf{R}_{(x-d)y}}\|_2} \qquad (4)$$

where $\overline{\mathbf{L}_{xy}}, \overline{\mathbf{R}_{xy}}$ – block means.

(a) Left disparity map

(b) Right disparity map
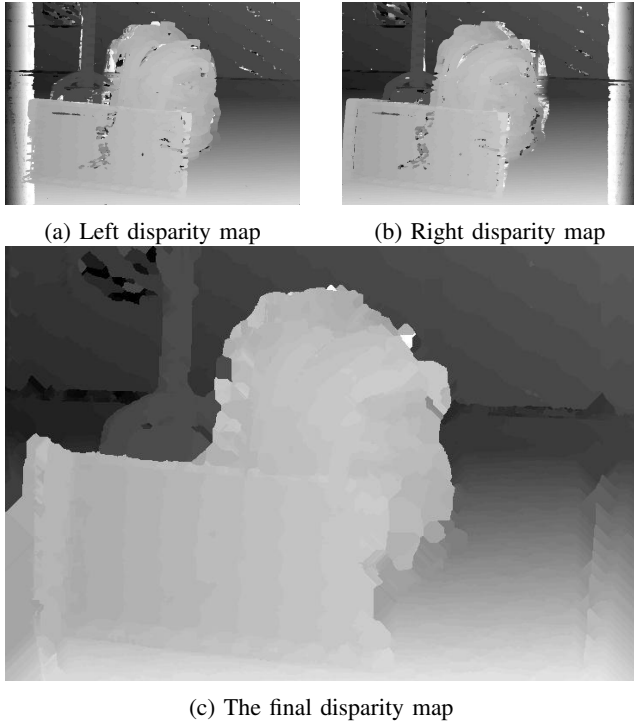

(c) The final disparity map

Fig. 2: Given images for disparity map calculation

Usually, the selection of the size of the reference window is not a simple and trivial task. It has been shown that the probability of a mismatch usually decreases as the size of the reference window increases. However, using large windows leads to an accuracy loss, since the influence of image differences increases greatly with the increase of the considered area. Therefore, this parameter must be great enough so that search window comprises the correspondent block of the given image. Furthermore, a difficult and important trade-off must be done when selecting this window size, since the computational load and processing time usually increase quadratically with the size of the window. Often, a compromise must be made, by adjusting these parameters according to the image size and contents. Taking into account all these factors and evaluating many experiments, we experimentally found that 15×21 is an optimal size of the window for our implementation and input data.

B. ZNCC algorithm design

The original that we had was scaled to a smaller size (4 times) in order to decrease computational time. Next, utilizing Eq. 4, we calculated disparity maps for both of the images and for a disparity value d from 0 to 64. The results are illustrated in Fig. 2a and Fig. 2b respectively. In order to compute Right-Left disparity we made a trick with setting up a disparity range. Specifically, the lower bound was initialized to -64 and the upper bound to 0. Once disparity maps for both images were computed, we perform post-processing consisting of two stages: cross-checking and occlusion filling. Cross-checking is calculated by taking the computed disparity value in one

image, and re-projecting it in the other image. If the difference in the values is higher than a given threshold (equals to 2 in our work), then the pixel value is initialized to 0. Occlusion filling was performed by applying the nearest neighbor interpolation. Neighborhood was chosen to be a square block 256×256. The final disparity map is illustrated in Fig. 2c.

Visually, we can observe, that our result looks quite good after post-processing steps, however, the computational load and elapsed time for this implementation are remarkably high – more than one minute. In the following section, we propose an approach for acceleration of our implementation of ZNCC algorithm based on OpenCL technology which significantly improves performance.

III. OPENCL IMPLEMENTATION

A. Description

In terms of OpenCL, there exist two types of devices – host and device. In our case, the device was a GPU and host – Central Processor (CPU).

We have decided to create a separate kernel for each of the sub-problems of our implementation (resizing, zncc computation, cross-checking and occlusion filling): *resize.cl*, *zncc.cl*, *cross_check.cl* and *occlusion.cl* respectively. We performed normalization of the disparity map (between 0 and 255) on the host, since it would take a lot of efforts to implement it in OpenCL and it would not give a significant performance improvement. The structure of the host code is presented in Algorithm 1.

Algorithm 1 Host algorithm

Input: Images filenames.
1: Check arguments.
2: Read images.
3: Chose the vendor and the device.
4: Allocate memory on the device for the original images and the intermediate results.
5: Copy the original images to the device, to the certain preallocated buffers.
6: Consequently set arguments load kernels to the queue.
7: Wait until the kernels execution is finished.
8: Copy the result into host memory.
9: Normalize the disparity map.
10: Save the result to *depthmap.png*.

IV. OPTIMIZATION OF OPENCL KERNELS

A. Local memory tricks

In our work we did not use any local memory tricks, but the potential usage of it would be in caching the blocks for the calculation of mean and standard deviation.

B. If-statements removal

First kernel optimization which we have made was to remove extra if-statements in kernels which check the boundaries. The outcome of such modification was a slight perfor-

TABLE I: Benchmarking of the naive OpenCL implementation

Device	Plain C time [sec]	GPU time [sec]	Perfor- mance gain
NVIDIA GTX960 OC	84.9350	0.8356	101.65
NVIDIA GTX750 Titan	77.5322	1.7745	43.7

mance improvement (improved value is presented in the Table **??**).

C. Work-group optimization

We considered work-groups sizes to be divisors of width and height of the resized images. As a global size we took the new image size – 504×735. We enumerated all possible values of work-group sizes using divisors of 504 and 735 and found out, that the work group size giving the best performance in our implementation of ZNCC is 3×21.

V. BENCHMARKS

We have tested our OpenCL implementation on two different machines with Ubuntu 14.04 64 bit. Both machines have NVIDIA GPUs, the same version graphics drivers and CUDA 7.5. We also ran our implementation on the laptop with Intel CPU 6200U Skylake. However, OpenCL implementation turned out to be slower than the original C version. Most probably, it happened because of the communication overhead. We could have improved the benchmark result, but decided to focus only on GPU implementations. Our benchmark results (average of 3 launches) are presented in Table **??**.

In our testing equipment, the machine with the more powerful GPU had less powerful CPU than the machine with GTX750 (Intel(R) Core(TM) i5-3470 vs. Intel(R) Core(TM) i5-4570). By changing the processors, our performance gain would be still good – 92.79. In our work we report it as a final result.

VI. COMPILING AND RUNNING

We have tested our implementation on the following software: Ubuntu 14.04 64 bit, GCC compiler version 4.8.4, NVIDIA drivers 352.93, NVIDIA CUDA 7.5. In order to compile the code, the user just need to use *make* utility (see the Listing 1).

Listing 1: Compiling commands

```
# Compile and run OpenCL code
make cl runcl
# Compile and run plain C code
make c runc
```

Our implementations of ZNCC are available on GitHub. The code is available by the following link – https://github.com/lext/zncc

VII. DISCUSSION AND CONCLUSIONS

In this work we have presented an implementation of ZNCC algorithm using OpenCL. Despite of the fact that we did not use any local memory tricks, which usually give a good performance gain, we have shown, that it is possible to achieve a quite good result even without them – 92.79 on a machine with Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz and GPU NVIDIA GTX960. We have also shown, that by changing the GPU to the newer generation, our performance gain has increased more than twice.

REFERENCES

[1] A. Redert, E. Hendriks, and J. Biemond, "Correspondence estimation in image pairs," 1999.
[2] A. Arsenio and J. Marques, "Performance analysis and characterization of matching algorithms," in *International Symposium on Intelligent Robotic Systems*, 1997.
[3] N. Qian, "Binocular disparity and the perception of depth," *Neuron*, vol. 18, no. 3, pp. 359 – 368, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0896627300812386
[4] P. Aschwanden and W. Guggenbül, "Experimental results from a comparative study on correlation type registration algorithms," in *Robust computer vision: Quality of Vision Algorithms*, 1992, pp. 268–282.