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 ZNCC algorithm. In particular, we show the accelerated implementation of the algorithm using OpenCL 1.2 standard. The performance gain which we managed to achive is around 100 times.**

I. Introduction

During the past few years, the estimation of the disparity field of an image sequence has been playing an increasingly important role in a large number of applications of the computer vision area such as multi-view image generation, 3D reconstruction from stereo image pairs and object recognition. Obtaining a precise and accurate depth map is the ultimate goal for 3D shape recovery and 3D image reconstruction. The main purpose of this report is focused on the process of the depth map computation from the images that are captured by the cameras placed in such positions so that a scene is taken from two slightly different views (angles).

By definition, disparity is the difference between two matched point's coordinates. The disparity map is composed by a dense field of disparity vectors, one for each matched pixel of group of pixels. Several different approaches have been developed to solve this correspondence search problem. However, this work concentrates on area based algorithms, where registration algorithms using cross-correlation based similarity measures are used to find the block of image pixels which best matches the one being analyzed in the other image. Area based algorithms still remain very popular and will assume an important position in the next future in correspondence computation tasks. The main reason for this fact relies on their simple and straightforward implementation, well suited for parallel implementations based on GPU computation, and their robustness against certain image transformations. Furthermore, these algorithms directly generate the dense disparity map, whereas in other approaches such as feature based algorithms the interpolation step is required if a dense map of the scene is desired. However, this algorithm presents some serious drawbacks. The most serious one is concerned with the high computational load, usually associated to the dense matching field computation. To overcome this disadvantage, an algorithm evaluating on GPU is proposed to compute dense disparity map. In our work, OpenCL 1.2 standard has been used. In addition, we conducted some experiments with optimization of the proposed algorithm utilizing the local memory of GPU, that allows us to improve performance of the algorithm up to 100 times compared to plain C implementation in absolute values.

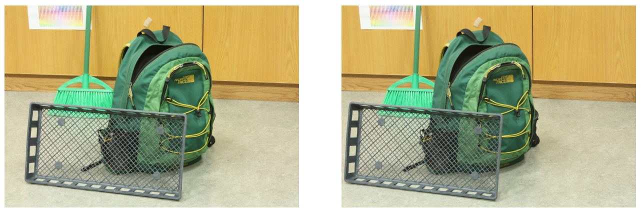

(a) left image (b) right image

Fig. 1: Given images for disparity map calculation

The report is organized as follows. In Section II ZNCC algorithms implemented in C language is presented. Section **??** describes the proposed method of calculating disparity map based on OpenCL technology. Section **??** presents benchmarks of the algorithm evaluated on two different GPU. In Section **??** 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

For every pixel in the left image our goal is to find the corresponding pixel in the right image (a match). Because a single pixel value is typically not discriminative enough to reliably find the corresponding pixel, one usually tries to match small windows around each pixel against all possible windows in the right image on the same row. We chose the window to be centered around the pixel and of constant size. The detailed explanation of setting window size is presented further in current section.

In 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. Disparity map is calculated for images presented in Fig. 1. For each block of the right image, designated by reference window, a correspondent block in the left image is sought using a given similarity measure as main criteria. During the search process, the correspondent block of the left image is moved by integer increments (c, l) around a predefined region, designated by *search window*, and an array of similarity scores $d(c, l)$ is generated for integer disparity values. In this work, we suggest the size of searching and referenced windows is equal. The position (c_m, l_m) of the moving block, corresponding to the maximum computed value of the considered similarity function $\otimes$ for that search window, is selected and chosen to compute the optimum disparity vector corresponding to that reference window. Hence, a matching dense field $\mathbf{D}(x, y)$ is obtained by using as many overlapping search windows as the number of pixels that compose the image, thus obtaining a single disparity vector for each pixel.

$$d(c, l) = \sum_{v=0}^{Rwidth} \sum_{u=0}^{Rlength} R(u, v) \otimes S(c + u, l + v) \tag{1}$$

$$d(c_m, l_m) = argmax\{d(c, l) : 0 \le c < S_w; 0 \le l < S_l\} \tag{2}$$

$$\mathbf{D}(x, y) = \{d_{xy}(c_m, l_m) : 0 \le x < W; 0 \le l < H\} \tag{3}$$

Many different similarity measures $\otimes$ have been referred in the literature. The reasons for a particular selection are usually related to the computational load, achieved performance and algorithmic simplicity. However, this work mostly concentrates on Zero-mean Normalized Cross-Correlation (ZNCC) similarity function:

$$ZNCC(R, S) = \frac{\left(R - \overline{R}\right) * \left(S - \overline{S}\right)}{\|R - \overline{R}\| \|S - \overline{S}\|} \tag{4}$$

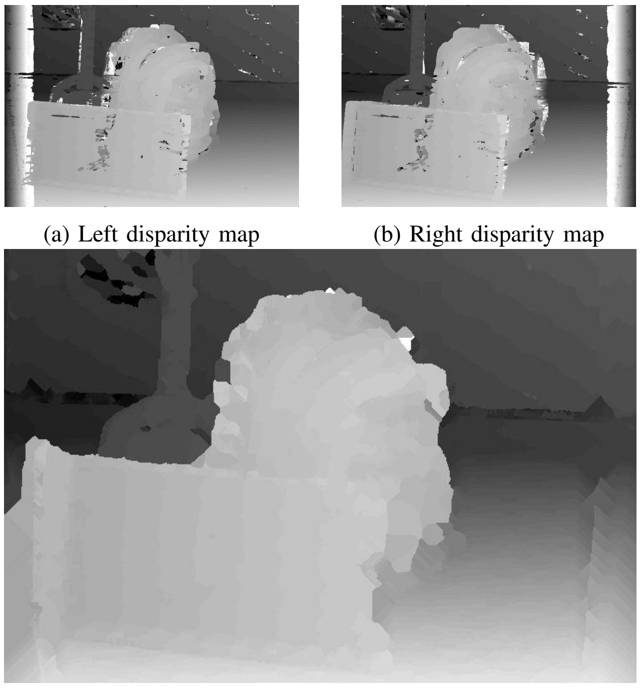
where numerator is the scalar product of the reference and search windows. R denotes a reference window pixel, S a search window pixel, $\overline{R}$ the local mean of the reference window: $\overline{R} = \sum_{v=0}^{Rlength} \sum_{u=0}^{Rwidth} R(u, v)$, and $\overline{S(c, l)}$ the pixel mean in the block of the search window being compared: $\overline{S(c, l)} = \sum_{v=0}^{Rlength} \sum_{u=0}^{Rwidth} S(c + u - d, l + v)$. The Euclidean norm is noted:

$$\|R\| = \sqrt{\sum_{v=0}^{Rlength} \sum_{u=0}^{Rwidth} \left(R(u, v) - \overline{R}\right)^2} \tag{5}$$

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 found that 3×21 is an optimal size of the window.

The estimation of dense disparity maps is usually performed by taking in consideration a set of constrains relating the two images being analyzed, making it possible to decrease the dimension and reduce the ambiguity of this problem, thus decreasing the total processing time. In this work, we admit two assumptions:

- *Epipolar Constrain:* allow to convert 2D search into 1D search. For the epipolar rectified image pair, each point in the left image lies on the same horizontal line (epipolar line) as in the right image. This approach is used to reduce a search space for depth map computation algorithms.
- *Unicity:* imposes that each pixel can have, at most, one correspondent pixel in the other image.


(a) Left disparity map (b) Right disparity map

(c) The final disparity map

Fig. 2: Given images for disparity map calculation

Utilizing Eq. 4, we calculated disparity map for both images and for a disparity value d from 0 to 64. The results are illustrated in Fig. 2a and Fig. 2b respectively. However, 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. Computing the disparities is trivial: simply choose at each pixel the disparity associated with the minimum cost value. Thus, these methods perform a local 'winner-take-all' (WTA) optimization at each pixel. Once disparity map for both images was 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 neighbour interpolation. The final disparity map is illustrated in Fig. 2c. It can clearly be seen that the proposed algorithm and post-processing steps allow to achieve good accuracy in calculation of disparity map. However, the computational load and elapsed time for this implementation are remarkably high. In following section, we propose an approach based on OpenCL technology which significantly improves performance.

III. "Naive" OpenCL implementation

A. Description

In the so called "Naive implementation", we have not used any optimization recommended by manufacturers. We have created a 2-dimensional grid of work-groups of different sizes, where each work-item corresponded one pixel.

We have decided to create a separte kernel for each of the subproblems (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 on the CPU. 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.
```

The Naive implementation of the kernels does not utilize any local memory optimization tricks and basically uses two dimensional workers' identifiers instead of indicies of loops in the one thread implementation.

IV. Benchmarks

We have tested the Naive 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. As a global size we took teh new image size – 504×735. We enumerated all possible values of workgroup sizes using divisors of 504 and 735 and found out, that the work group size, which gives the best performance in our implementation of ZNCC is 3×21. We have also run 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 improve the benchmark result, but decided to fucus only on GPU implementations. Our benchmark results (average of 3 launches) are presented in Table I.

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

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 this report, we consider it as our final result.

V. Optimization of OpenCL kernels

A. 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 performance improvement (improved value is presented in the Table I).

B. Local memory usage

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.

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 our 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
```

VII. Discussion and Conclusions

In this work we have presented a Naive implementation of ZNCC algorithm using OpenCL. Despite of the fact, that We did not use any local memory tricks, we have shown, that it is possible to achieve a quite good result of performance gain – 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 in more than twice.