USING THE GPU FOR FAST SYMMETRY-BASED DENSE STEREO MATCHING IN HIGH RESOLUTION IMAGES

Vasco Mota[†] *Gabriel Falcao*[⋆] *Michel Antunes*[†] *Joao Barreto*[†] *Urbano Nunes*[†]

[†]Institute of Systems and Robotics, Dept. of Electr. & Computer Eng., University of Coimbra, Portugal
[⋆] Instituto de Telecomunicações, Dept. of Electr. & Computer Eng., University of Coimbra, Portugal

ABSTRACT

SymStereo is a new algorithm used for stereo estimation. Instead of measuring photo-similarity, it proposes novel cost functions that measure symmetry for evaluating the likelihood of two pixels being a match. In this work we propose a parallel approach of the LogN matching cost variant of SymStereo capable of processing pairs of images in real-time for depth estimation. The power of the graphics processing units utilized allows exploring more efficiently the bank of log-Gabor wavelets developed to analyze symmetry, in the spectral domain. We analyze tradeoffs and propose different parameterizations of the signal processing algorithm to accommodate image size, dimension of the filter bank, number of wavelets and also the number of disparities that controls the space density of the estimation, and still process up to 53 frames per second (fps) for images with size 288×384 and up to 3 fps for 768×1024 images.

Index Terms— Stereo estimation, Stereo rangefinding, SymStereo, Parallel processing, High resolution images

1. INTRODUCTION

Recent developments have shown that stereo disparity estimation can be improved by using symmetry, instead of photo-similarity, for evaluating the likelihood of two pixels being a match [1] . In this new SymStereo framework the pixel association problem is solved by mapping one view into the other according to the homography induced by a virtual plane passing in-between the cameras, which is followed by measuring local signal symmetry to detect the contour where the virtual plane intersects the scene. The experiments show that SymStereo compares favorably against top performing matching costs that rely in photo-similarity [1], and that its variant logN is especially effective for Stereo-rangefinding that consists in recovering the scene depth exclusively along a pre-defined virtual scan plane.

This work was supported by the Portuguese Foundation for Science and Technology (FCT) under Grants PDCS10:PTDC/EEA-AUT/113818/2009 and AMS-HMI12: RECI/EEI-AUT/0181/2012, and also by a Google Research Award from Google Inc.

This work investigates for the first time the use of the processing power of graphics processing units (GPU) for evaluating symmetry-based stereo matching costs in pairs of high-definition (HD) images in order to accomplish real-time depth estimation. In particular, we focus in the logN variant of the algorithm where the image symmetry analysis is carried using a bank of log-Gabor wavelets. Since the log-Gabor is an analytical signal, the filtering must be carried in the spectral domain and the result stored in memory for subsequent processing. This poses important challenges in terms of both parallel processing and in particular regarding the GPU's memory hierarchy management, since we need that a huge amount of data is maintained in memory for processing.

The following contributions of the paper can be highlighted: *i)* Investigating till which extent GPUs can improve its computational performance. This becomes even more important since full HD is becoming the mainstream. Therefore, there is an urge to perform stereo matching in HD images in real-time. *ii)* The symmetry calculation is performed in the frequency domain, where the convolution (i.e., the wavelets processing) must be performed, which can have benefits in terms of processing time after a certain dimension of the image (e.g., HD images). *iii)* Also, several tradeoffs are analyzed and different parameterizations proposed for the algorithm. For example, in case of higher speed requirements, we can sacrifice spatial resolution without sacrificing depth resolution.

2. SYMSTEREO AND DEPTH MAPS

The concept behind passive methods for stereo correspondence (like SymStereo) is to find the correct depth map over a scene. This map associates each pixel on an image with its depth on the corresponding scene.

This mapping procedure is calculated using a process that involves two main steps: *i)* the first one consist of using a matching cost function across all possible disparities and pixel locations, producing a Disparity Space Image (DSI); *ii)* in the second step, the DSI goes trough an evaluation process whose ultimate result is the depth map. Our work focus on the first step.

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

2.1. Disparity and the DSI

Consider two images, I (left image) and I' (right image). Consider also a given epipolar line on each image ($I(q_1)$ and $I'(q_1)$). Disparity is the distance, in pixels, from a pixel in line $I(q_1)$ to its equivalent in $I'(q_1)$. Two pixels are equivalent if they represent the same information on the scene. If the disparity is 0, the object is on infinity. The bigger the disparity, the closer the object is to the focal center.

The DSI consists is a 3D volume that, for each pair $\{p_i, d_j\}$, (where p_i stands for a pixel of the reference image and d_j stands for a disparity value) associates the corresponding matching cost, which in this case is the joint energy.

2.2. The cost function

The cost function can be divided in two steps: the filtering and the computation of the joint energy for each pair $\{p_i, d_j\}$.

2.2.1. Filtering Phase

On the *filtering phase*, N log-Gabor wavelets are used. These are 1D analytical filters and filtering must occur in the spectral domain. The wavelets' wavelength grows as follows:

$$\lambda_k = m.\lambda_{k-1}, \tag{1}$$

where m represents the multiplicative factor. The matrix of coefficients of the filter is represented by G. The result of filtering, for a given wavelet k, is $\mathcal{I}^T.G_k$, where G_k is a wavelet, with the same length as the epipolar line and $\mathcal{I} = \mathcal{F}(I)$, where $\mathcal{F}$ denotes the 1D Fourier transform along the epipolar lines of I. After the IDTFT ($\mathcal{F}^{-1}$) we obtain, for a given epipolar line q_1 and a given wavelet k, the following complex signal:

$$\begin{cases} s_k(q_1) + ia_k(q_1) = \mathcal{F}^{-1}(\mathcal{I}(q_1).G_k) \\ s'_k(q_1) + ia'_k(q_1) = \mathcal{F}^{-1}(\mathcal{I'}_f(q_1).G_k) \end{cases}. \tag{2}$$

I'_f represents I' flipped horizontally. I' is flipped before the filtering occurs and flipped back once again after returning to the time-domain.

2.2.2. Energy Computation Phase

To calculate the Energy, first, we calculate the Symmetry (s^S and a^S) and AntiSymmetry coefficients (s^A and a^A). For a given pair $\{p_i, d_j\}$,

$$\begin{aligned} &s^S_k(p_i, d_j) + ja^S_k(p_i, d_j) = \\ &= \big(s_k(p_i) + s'_k(p_i - d_j)\big) + j\big(a_k(p_i) + a'_k(p_i - d_j)\big) \end{aligned} \tag{3}$$

$$\begin{aligned} &s^A_k(p_i, d_j) + ja^A_k(p_i, d_j) = \\ &= \big(s_k(p_1) - s'_k((p_i - d_j)\big) + j\big(a_k(p_i) - a'_k((p_i - d_j)\big). \end{aligned} \tag{4}$$

These coefficients allow us to calculate the Symmetry Energy (E^S):

$$E^S(p_i, d_j) = \frac{\sum_{k=1}^{N} |s^S_k(p_i, d_j)| - |a^S_k(p_i, d_j)|}{\sum_k \sqrt{(s^S_k(p_i, d_j))^2 + (a^S_k(p_i, d_j))^2}} \tag{5}$$

and the Anti-Symmetry Energy (E^A):

$$E^A(p_i, d_j) = \frac{\sum_{k=1}^{N} |a^A_k(p_i, d_j)| - |s^A_k(p_i, d_j)|}{\sum_k \sqrt{(s^A_k(p_i, d_j))^2 + (a^A_k(p_i, d_j))^2}}. \tag{6}$$

Combining both equations, we calculate the joint energy E:

$$E = E^A \times E^S. \tag{7}$$

3. PARALLELIZING SYMSTEREO

The pipeline of SymStereo is depicted on Algorithm 1. The most intensive processing is performed on the GPU device. We use the Compute Unified Device Architecture (CUDA) parallel programming model [2] that allows exploiting multithread-based execution on the GPU device. The host system is a CPU that communicates with the GPU using the PCIe bus.

Algorithm 1 SymStereo Pipeline

1: Creation of the Gabor Coefficients, G
2: (CPU to GPU data transfer) I, I' and G are copied to device
3: - - - - - - - - - - (On GPU)
4: Flip I' horizontally
5: DTFT of I and I'_f
6: Filtering of $\mathcal{I}$ and $\mathcal{I}'_f$
7: IDTFT of $G.\mathcal{I}$ and $G.\mathcal{I}'_f$
8: Flip back $\mathcal{F}^{-1}(G.\mathcal{I}'_f)$
9: Energy computation
10: - - - - - - - - - - (Back to CPU)
11: (GPU to CPU data transfer) DSI is copied back to host

Next we describe the main strategies used to perform the parallelization of the algorithm.

3.1. Fourier Transform and LogGabor Filtering in the Frequency-domain

For the DTFT and IDTFT, the optimized CUFFT API [3] was used to perform the FFT calculation on the GPU. On the filtering phase, each line of the spectrum is multiplied by the same matrix of filter's coefficients, G.

3.2. Calculating Energy

The energy computation implements (5) and (6), in order to compute the joint energy from (7). This value is saved in the DSI. Each thread is responsible for an entry of the DSI. Therefore, it is necessary to perform the sum of the wavelets for a

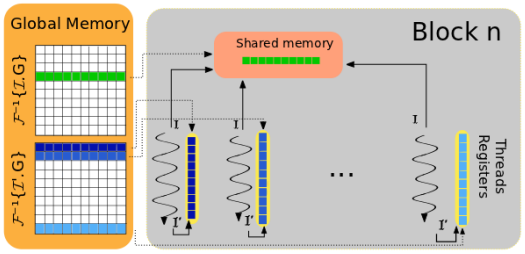


Fig. 1. The usage of different types of memory during Energy calculations. All threads in the same block access the same info from $\mathcal{F}^{-1}(\mathcal{I}.G)$

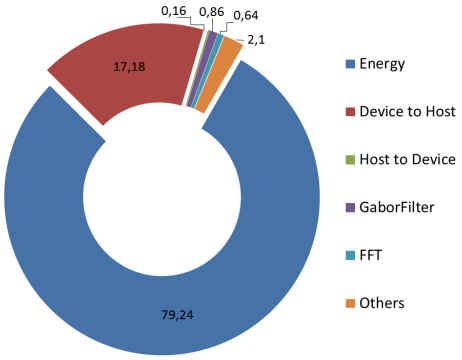


Fig. 2. Percentage of the execution time spent on each task

given pair $\{p_i, d_j\}$. Each thread's input can be represented as 2 columns: the entries from $\mathcal{F}^{-1}(\mathcal{I}.G)$ and $\mathcal{F}^{-1}(\mathcal{I}'_f.G)$ referring to the pair $\{p_i, d_j\}$. Due to the considerable size of data, all these entries are stored in the global memory after flipping $\mathcal{F}^{-1}(\mathcal{I}'_f.G)$.

Consider I as the reference image. If we group the computation by the pixels of I, we know that the threads on the same group will be accessing the same data from $\mathcal{F}^{-1}(\mathcal{I}.G)$, but not from $\mathcal{F}^{-1}(\mathcal{I}'_f.G)$, since that for each disparity value we need a different pixel from that image.

By putting data from $\mathcal{F}^{-1}(\mathcal{I}.G)$ on the faster shared memory, we greatly enhance the throughput performance. The process is depicted in Fig. 1. Consequently, each thread first copies data from $\mathcal{F}^{-1}(\mathcal{I}'_f.G)$ to its registers, accesses the shared memory for $\mathcal{F}^{-1}(\mathcal{I}.G)$ and then computes the values of equations (5) and (6), a value of k at a time. In the end, all the N results are used to calculate (7). Finally, values are saved on the DSI.

The DSI dimensions are considerably high. Therefore, even the parallel version of the Energy calculation consumes the most significant part of global processing time as Fig. 2 shows. The DSI dimension depends on the image resolution times the disparity range and, for each entry, we process 2 columns of N elements each. Therefore, for a 768×1024 pixels image, with $d = 170$ disparities and a $N = 45$ wavelets filter, in this stage we access 510 MByte of data.

4. APPARATUS AND EXPERIMENTAL RESULTS

The program was developed using CUDA 5.0 with runtime 4.2. The C/C++ code was compiled with GCC-4.4.7. The host system is based on an Intel Core i7 950 @ 3.07GHz that runs the GNU/Linux kernel 3.2.1-030201-i7. The GPU device is a Geforce GTX 680 powered with 1536 CUDA cores.

4.1. Experimental Results for Photo-Symmetry

Our experiments are based on variations of the following parameters: the number of wavelets of the filter (N), the disparity range (d) and the multiplicative factor of the filter (m) (see (1)). These experiments were performed in 2 different sets of images. The first is the Tsukuba stereo set (288×384 pixels) and the second one is a pair of larger images (768×1024 pixels) captured by our group. As m does not change the computational effort, we do not show results for variations of this parameter. However, it should be noted that it does influence the quality of the result. So, we averaged the results for different values of m. The parameters used are presented in table 1. We measured the temporal results of computations being performed on the GPU. For the first set, results are shown in table 2. For this image format, we can get from 20 fps up to 50 fps, which is the frame rate used for video.

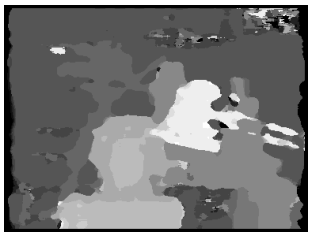
Table 1. Parameters used on our tests: number of wavelets (N), disparity range (d) and filter's multiplicative factor (m)

	Set 1	Set 2
N	{15, 20, 30}	{20, 30, 25}
d	{7, 15, 31}	{70, 128, 170}
m	{1.02, 1.05, 1.08 }	

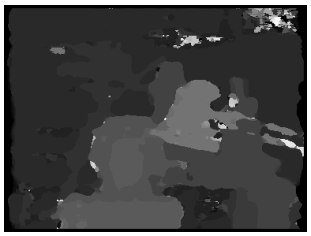
Table 2. Execution time (in ms) for the first set of images

	$d = 7$	$d = 15$	$d = 31$
$N = 15$	18.70	20.50	24.42
$N = 20$	23.80	25.69	29.16
$N = 30$	37.98	39.91	43.69

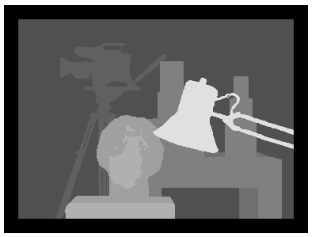
In Fig. 3, we compare our results with the algorithm available in [4] and implemented in [5]. Images (a) and (b) show SymStereo's output. Image (c) presents the Tsukuba's ground truth disparity, which are the correct values of disparity for the given set. Image (d) is obtained by using the algorithm implemented on [5] with the Tsukuba set as input. As it is observable, our results present less noise, but a lower level of definition on the discontinuities. The use of a large number of scales enables Symstereo to have a good performance in tex-



(a) $N = 30$,$d = 15$, $m = 1.05$



(b) $N = 30$,$d = 31$, $m = 1.05$



(c)Tsukuba's ground truth disparity

(d) Results using [5]

Fig. 3. Tsukuba set results comparing SymStereo with [5]

ture less regions with the disadvantage of more blurred edges close to discontinuities in the disparity image.

For the second set of higher resolution images, we can see from table 3 that we are able to process up to 3 fps. Given the GPU's memory size limitations, we were not able to test $N > 45$. Image 4 shows the final depth map obtained form

Table 3. Execution time (in ms) for the second set of images

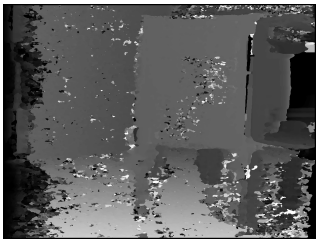
	$d = 69$	$d = 128$	$d = 170$
$N = 20$	358.87	448.73	630.26
$N = 30$	528.2	647.7	922.1
$N = 45$	877.2	987.7	1412.7

the DSI. The original image from Fig. 4 (a) represents a challenge for stereo estimation since it has few textures and many reflexions. Nevertheless, we can see on the image that, on the right, the most distant plans, outside, are very well reconstructed. Also, the reflexions on the ground are correct: since the information perceived on the image is the light from the outside reflected on the floor, instead of the floor itself, the depth calculated corresponds to the source of such light: the surfaces on the outside. It is also noticeable the board on a closer plan than the wall on the background.

4.2. Speedup

In terms of speedup, the computation on the CPU of the image from set 1, with $N = 30$, $d = 15$ and $m = 1.05$ takes 1.712 seconds to complete. This represents a speedup of $43\times$. On the second set, for a setup with $N = 45$, $d = 170$ and $m = 1.06$, the CPU takes approximately 158.78 seconds. The achieved speedup for this image of higher dimensions is $112\times$ regarding CPU execution.

(a) Original left image



(b) Depth map

Fig. 4. SymStereo results for the second set's image with $d = 170$, $m = 106$ and $N = 45$

Regarding disparities/second (disp/s) (measuring unit used in [6]), our algorithm is able to perform 0.078×10^9 disp/s for set 2 and 0.083×10^9 disp/s for set 1. If we allow to downgrade quality, using a smaller N, for the second set we are able, with $N = 20$, to achieve, 0.21×10^9 disp/s.

5. RELATION TO PRIOR WORK

Our work is the first attempt to use GPGPU on dense stereo, using the SymStereo Framework presented in [1], where Antunes and Barreto worked on a fast CPU algorithm (2 fps), but only for sparse stereo. In [7] Antunes *et al.* discuss the utilization of this approach for laser stereo range finding.

Several approaches have been made on the GPGPU usage for stereo. A very important variable is the computational complexity of the cost function. In [8], the authors analyze this aspect in terms of speed and quality of different solutions. Another review regarding speed is also performed in [6] in terms of disparities/second. Our algorithm performs around 0.08×10^9 disp/s. Yang *et al.* [9], one of the first works on the field, achieved 6 to 8 fps on 256×256 depth map with a 100 disparity search. Zhao *et al.* [10] creates a very fast framework for stereo with moving objects, using adaptive window sizes, achieving 30 fps on a 768×1024 stereo pair with a 256 pixel disparity range. By adopting hybrid techniques using both the GPU and the CPU, Miura *et al.* [11] perform their computations using a set of reference points on the image. With 6000 points, they achieve a 15 fps rate.

6. CONCLUSIONS

In this work we developed a parallel approach for SymStereo, a depth estimation signal processing algorithm based on photo-symmetry, rather than photo-similarity. Depending on parameters such as image size or number of wavelets used, we were able to process up to 53 fps for small images and 3 fps for high resolution images.

Given the data rates achieved, in the near future we expect to develop multiview systems for higher quality of depth estimation and reconstructed images.

7. REFERENCES

[1] M. Antunes and J.P. Barreto, "Stereo estimation of depth along virtual cut planes," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, 2011, pp. 2026–2033.

[2] Victor Podlozhnyuk, Mark Harris, and Eric Young, "Nvidia cuda c programming guide," NVIDIA Corporation, 2012.

[3] NVIDIA Developer Technology, "cuFFT user guide," 2012.

[4] D. Gallup, J. m. Frahm, and J. Stam, "CUDA stereo," June 2009.

[5] S. J. Kim, D. Gallup, J. m. Frahm, A. Akbarzadeh, Q. Yang, R. Yang, D. Nistr, and M. Pollefeys, "Gain adaptive real-time stereo streaming," in *Int. Conf. on Vision Systems*, 2007.

[6] R. Kalarot and J. Morris, "Comparison of FPGA and GPU implementations of real-time stereo vision," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, 2010, pp. 9–15.

[7] M. Antunes, J.P. Barreto, C. Premebida, and U. Nunes, "Can stereo vision replace a laser rangefinder?," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012, pp. 5183–5190.

[8] Ratheesh Kalarot, John Morris, David Berry, and James Dunning, "Analysis of real-time stereo vision algorithms on GPU," 2011.

[9] Ruigang Yang and M. Pollefeys, "Multi-resolution real-time stereo on commodity graphics hardware," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, 2003, vol. 1, pp. I–211–I–217 vol.1.

[10] Y. Zhao and G. Taubin, "Real-time stereo on GPGPU using progressive multi-resolution adaptive windows," *Image and Vision Computing*, 2011.

[11] M. Miura, K. Fudano, K. Ito, T. Aoki, H. Takizawa, and H. Kobayashi, "GPU implementation of phase-based stereo correspondence and its application," in *Image Processing (ICIP), 2012 19th IEEE International Conference on*, 2012, pp. 1697–1700.