DISTRIBUTED DENSE STEREO MATCHING FOR 3D RECONSTRUCTION USING PARALLEL-BASED PROCESSING ADVANTAGES

R. Ralha[⋆] *G. Falcao*[⋆] *J. Andrade*[⋆] *M. Antunes*[‡] *J. P. Barreto*[†] *U. Nunes*[†]

[⋆] Instituto de Telecomunicações, Dept. of Electr. and Computer Eng., Univ. of Coimbra, Portugal
[‡]Interdisciplinary Centre for Security, Reliability and Trust (SnT), Univ. of Luxembourg, Luxembourg
[†]Institute of Systems and Robotics, Dept. of Electr. and Computer Eng., Univ. of Coimbra, Portugal

ABSTRACT

Instead of measuring photo-similarity, SymStereo is a stereo vision algorithm that uses new cost functions to measure symmetry differences between pairs of images. In this paper we propose the acceleration of a complete signal processing pipeline for generating 3D volumes based on dense SymStereo. The outputs here generated achieve superior reconstruction quality namely for slant based scenarios, so typical in autonomous systems, that have to capture pairs of images and perform moving decisions in real-time. In particular, we analyse several parallelization strategies for the compute-intensive aggregation procedure using different parameters and evaluate a trade-off between processing time, and higher precision of the calculated depths and quality of the final reconstructed 3D volume. The developed parallel pipeline allows to process more than 4.5 volumes per second for high resolution images using commodity GPUs, which conveniently suits its application in a variety of robotics systems.

Index Terms— Stereo estimation, SymStereo, Parallel processing, 3D Reconstruction, High resolution images

1. INTRODUCTION

Recently, a new algorithm that uses photo-symmetry instead of photo-similarity-based cost functions has been proposed by Antunes et al. [1, 2]. This new pipeline for calculating disparity maps, baptised SymStereo, and in particular its variant logN shows superior performance in recovering the scene's depth for pairs of images with slant (please see the log20 variant in Fig.18 from [2]). This particularity of the algorithm encourages the development of new methods for extracting higher quality from the generated disparity map and 3D volume, which is a fundamental procedure in autonomous systems, namely vehicles and robots, that constantly have to perform analysis of images with slant for making decisions, namely regarding trajectory, preferably in real-time.

This paper investigates the manipulation of the sensitive aggregation phase in the SymStereo processing pipeline [3], namely the algorithmic gains achievable with its parallelization and the corresponding room they create for increasing the complexity of the aggregation procedure that may produce better 3D images. The main contributions of this paper can be summarized as: i) proposing a real-time stereo pipeline that creates realistic 3D volumes for slant-based scenarios; ii) investigating the acceleration that multiple GPUs can provide to the pipeline, for creating a real-time 3D volume generator; and iii) analysing how the quality of the final 3D volume depends on the variation of the aggregation window size.

2. STEREO ALGORITHM PHASES

Stereo Algorithms comprise one or more of the following:

1. Matching cost;
2. Cost (support) aggregation;
3. Disparity computation;
4. Disparity refinement.

This paper focus on the final three steps, as the SymStereo matching cost pipeline was already addressed [3]. Additionally, we add a fifth step to our study, the 'Disparity to 3D' step, where 3D coordinates are calculated to generate a 3D volume of the 2D disparity maps.

2.1. Cost aggregation and disparity computation

After the calculation of matching costs, the best disparity for each pixel must be chosen from the DSI [4].To achieve this, two types of aggregation algorithms can be used: local or global ones. While local algorithms use a window-based approach [5], global algorithms solve a global optimization problem by finding the disparity that minimizes a global cost function that is composed by data and smoothness terms [6].

Despite usually producing better results, global algorithms are computationally heavier and not all can be parallelized. This is the main reason why we use a window-based algorithm for the cost aggregation phase.

This work was supported by the Portuguese Foundation for Science and Technology (FCT) under grants AMS-HMI12: RECI/EEI-AUT/0181/2012, UID/EEA/50008/2013 and SFRH/BD/78238/2011 and also by a Google Research Award from Google Inc.

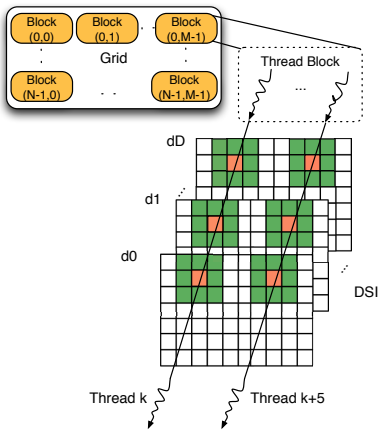


Fig. 1. Aggregation phase with window size 3 on the GPU. (M, N) are the number of blocks in the (x, y) directions.

In Fig.2 we illustrate this phase. We calculate the sum of the matching costs over a square window for each image pixel and each disparity. The most accurate disparity will then be chosen by a Winner-Takes-All (WTA) strategy [6].

2.2. Disparity refinement

The disparity refinement stage can be divided in two sub-stages: left-right consistency check and filling of occluded pixels. Occluded pixels are only visible in one of the images.

2.2.1. Left-Right Consistency Check

The left-right consistency check uses two disparity maps, one computed with the left image as the reference and the other with the right image. This way, we can subtract the disparities of corresponding pixels in each image. If the difference is less than a given threshold, the pixel is considered occluded.

2.2.2. Filling of Occluded Pixels

To fill the occluded pixels, we use an algorithm that performs a 4-way search for the first non-occluded pixel in each way. The disparity selected is the median between the four values that were found.

2.3. From Disparity Maps to 3D

To calculate the 3D coordinates for each pixel, we use the equations that map 2D coordinates to 3D:

$$Z = (f * b)/D; \quad (1)$$

$$X = ((x - c_x) * Z)/f; \quad (2)$$

$$Y = ((y - c_y) * Z)/f; \quad (3)$$

where f is the focal length (in pixels), b is the distance between the two lens (in metres), c_x and c_y are the image centres (in pixels) and D is the disparity of the pixel.

3. PARALLELIZING 3D PIPELINE

In order to calculate 3D maps in real-time, we use two Nvidia GTX Titan GPUs to accelerate processing. We exploit a hybrid architecture, taking advantage of both CPU and GPUs.

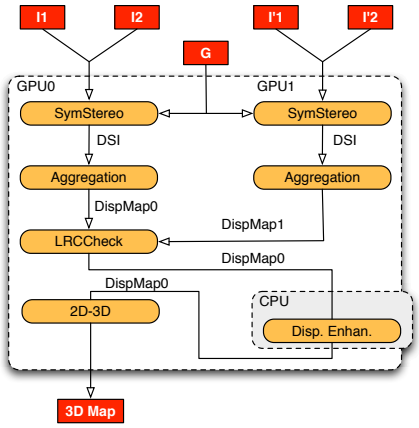


Fig. 2. 3D Pipeline representation, where $I1$ and $I2$ are the left and right images, $I'1$ and $I'2$ are the left and right images flipped and G are the Gabor coefficients.

Since data transfers from the CPU to the GPU consume a significant amount of time to complete, we try to minimize their impact. Data allocations are pageable in the CPU by default. Since the GPU cannot access data from pageable memory, when a transfer is called, data has to be transferred to a temporary pinned array and only then it is transferred to the device. To avoid this, we always make pinned allocations in the host, saving time in data transfers.

3.1. Disparity Calculation

For this step, each thread, corresponding to one pixel, calculates the sum of the matching costs over the defined square window, for each disparity, and chooses the disparity with the highest sum of costs (Fig.2). The amount of data processed depends on the disparity range we choose at the beginning of the pipeline and window size. We can evaluate this in Fig.3. By increasing the window size, not only will there be more processing time involved but there will also be more accesses to global memory. To overcome this problem, we tested the use of shared memory but the quantity of data we would had to transfer for each block penalised execution time.

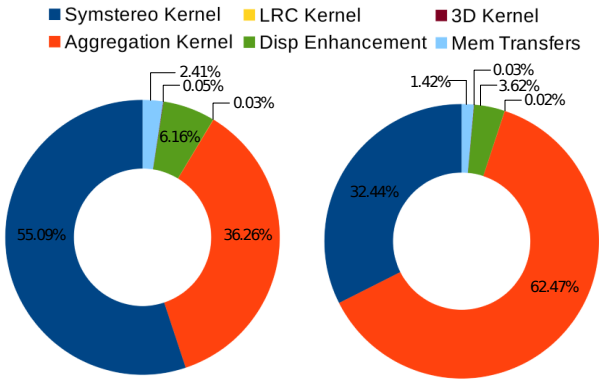


Fig. 3. Workload variation by changing the aggregation window from 9 to 15 on a 768×1024 pixels image.

3.2. Pixel consistency and filling

In order to perform the consistency check, we have to calculate two disparity maps. To accelerate this step, we used two GPUs in parallel, each one calculating one of the necessary maps. As in the previous step, each thread will be responsible for the verification of the consistency of a pixel. To fill the occluded pixels we use an algorithm that cannot be parallelized. This way, we have to transfer data from the GPU to the CPU. At the end of the process, data is copied back to the GPU.

3.3. 3D Reconstruction

For each pixel, a thread is responsible for calculating the three coordinates necessary to generate the 3D map. When all the pixels are processed, data is transferred back to the host.

4. APPARATUS AND EXPERIMENTAL RESULTS

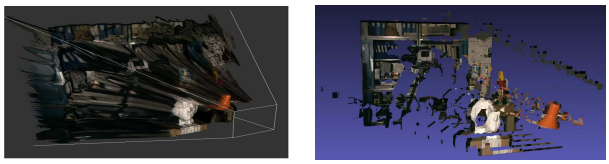
The reconstructed images were processed on a GeForce GTX Titan dual-GPU workstation with an i7-4770k @ 3.5 GHz using CUDA 6.5 and GCC 4.4.7. To visualize the 3D maps we used MeshLab v1.3.2. The developed framework scales with the number of available hardware resources and can be ported to run in other multicore architectures [7].

4.1. 3D Reconstruction Results

In order to enhance our results, we decided to alter the window size of the aggregation stage. This alteration was applied to three sets of images, one from the Tsukuba set (288x324 pixels), one from the Kitty Dataset [8] (375x1242 pixels) and another one captured by us (768x1024 pixels). We can observe the aggregation phase processing times depending on the window size and image dimensions in Table 1. To perform the 3D Reconstruction of the images, we used parameter values for the Tsukuba, KITTI and our image datasets provided by the authors [6, 8] and by our camera.

Table 1. Aggregation time (ms) varying the window size

Aggregation Window Size	Our dataset 768×1024	KITTI 375×1242	Tsukuba 288×384
9	78.46	51.46	1.99
11	127.26	84.76	2.99
13	171.63	115.94	4.11
15	212.48	140.19	5.46



(a) Tsukuba 3D from [9] (b) Tsukuba 3D from our method

Fig. 4. Tsukuba 3D reconstruction comparing [9] – compressed vertically – with our method for 16 disparities.

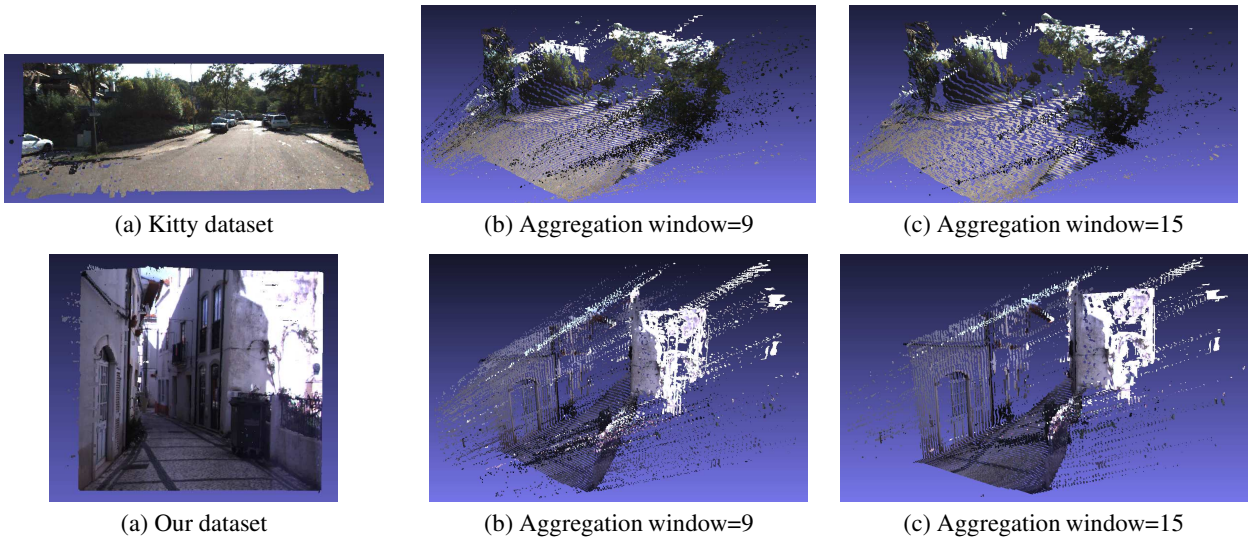
In Fig.4, we compare the 3D reconstruction for the Tsukuba image set. We selected a disparity range ranging from 0 to 15, as suggested in [6], for our method. Despite some discontinuity errors, mainly in the top right corner, our reconstruction is pretty accurate.

The 3D reconstructions of the KITTI dataset image and our own image are shown in Fig.5. These were computed with a disparity range of 15 to 125, since they are images with a larger resolution. In the analysis we notice some bad reconstructed pixels. The SymStereo matching cost struggles with shadows, reflections and luminosity variations between the left and right image. By increasing the length of the aggregation window, we minimize these effects but lose definition on the discontinuities.

Comparing the image of the KITTI dataset with ours, we

Table 2. Pipeline tasks time (ms) and individual kernel speedup – in brackets – for each image dimension.

Image Resolution [Processor] Kernel	Our dataset 768×1024	KITTI 375×1242	Tsukuba 288×384
[CPU] SymStereo	32082.00	19513.00	784.00
[CPU] Aggregation	24024.00	14318.00	495.00
[CPU] LRCCheck	47.02	29.32	4.71
[CPU] Disp. Enhan.	15.98	8.58	1.11
[CPU] 2D-3D	10.62	6.67	1.69
[GPU] SymStereo	[252×] 127.51	[207×] 94.28	[160×] 4.91
[GPU] Aggregation	[306×] 78.46	[278×] 51.46	[249×] 1.99
[GPU] LRCCheck	[775×] 0.06	[647×] 0.05	[303×] 0.02
[CPU] Disp. Enhan.	[N/A] 15.98	[N/A] 8.58	[N/A] 1.11
[GPU] 2D-3D	[105×] 0.10	[88×] 0.08	[63×] 0.03



(a) Kitty dataset (b) Aggregation window=9 (c) Aggregation window=15

(a) Our dataset (b) Aggregation window=9 (c) Aggregation window=15

Fig. 5. Aggregation window size influence in 3D reconstruction: a) Front reconstructed image; b, c) Side reconstructed image.

see that our image presents better results. Despite both images having a high level of slanted surfaces, the image of the KITTI Dataset has more discontinuities (e.g. trees, cars, signs), than our image. This corroborates with what was shown in [2], that symmetry-based algorithms have a superior behaviour with less textured and high slanted surfaces.

In Table 2, we can verify the computation times that each phase take on the CPU and GPU. For the Tsukuba image, we were able to achieve up to 124 frames per second (FPS), for the KITTI dataset image we obtained up to 6.5 FPS and for our image we measured up to 4.5 FPS.

4.2. Speedup

For our experiment, dedicating two GPUs is a major advantage since the two disparity maps, necessary for left-right consistency check, are computed in parallel. Hereupon, with an aggregation window of size 9, the Tsukuba image takes approximately 2.5 seconds to process on the CPU [2]. We managed a total speedup of $318\times$. For the KITTI dataset image, we accelerated our program $438\times$, since its serial counterpart takes 68 seconds to complete. Finally, for our image, we achieved a total speedup of $505\times$, as it consumes 112 seconds to generate a 3D map. The individual speedups achieved for each method are broken down in Table 2.

5. RELATION TO PRIOR WORK

Using GPUs for stereo matching has become a recurring practice nowadays. With the parallel power of these devices, algorithms are becoming increasingly faster, which enables to achieve real-time stereo matching performance. Adding more stages to the stereo algorithm adds complexity but it also yields better results in the final output. Like us, Kowalczuk *et al.* [10] implement a complex stereo algorithm on a GPU, using an iterative refinement technique for correspondences with adaptive support-weight. With two aggregation stages, two refinement stages and consistency check, they achieve a rate of 62 FPS in low resolution images. Our method has less stages and achieves 124 FPS for the same dimension.

Regarding 3D reconstruction, Denker *et al.* [9] uses multi-camera systems for face recognition and achieves a frame rate of about 4 FPS with a 1392×1032 resolution, while in [11] developed a real-time 3D face-measurement system capable of analysing 6000 to 7000 3D points in 15 FPS. Only once was the SymStereo framework presented in [2] brought on to the GPU. Mota *et al.* [3] implemented the algorithm and achieved 53 FPS for low resolution images and 3 FPS for high resolution images. We improved on their work, accelerating the framework, adding three more stages for better visual results and 3D reconstruction, and implementing them on a dual-GPU system. With it, we achieved a frame rate of 124 FPS for low resolution images and of 4.5 FPS for high resolution images.

6. CONCLUSIONS AND FUTURE WORK

This work presented a real-time pipeline for 3D reconstruction that achieves high rates, with 124 FPS for low resolution images and 4.5 FPS for high resolution ones. We intend to investigate and apply new types of parallel aggregation algorithms with the objective of enhancing the generated 3D map, and also to use CUDA streams with the developed methods.

7. REFERENCES

[1] M. Antunes and J.P. Barreto, "Stereo Estimation of Depth Along Virtual Cut Planes," in *IEEE Int. Conf. on Computer Vision Workshops*, 2011, pp. 2026–2033.

[2] Michel Antunes and João P Barreto, "SymStereo: Stereo Matching using Induced Symmetry," *Int. Journal of Computer Vision*, pp. 1–22, 2014.

[3] Vasco Mota, Gabriel Falcao, Michel Antunes, Joao Barreto, and Urbano Nunes, "Using the GPU for Fast Symmetry-based Dense Stereo Matching in High Resolution Images," in *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*. IEEE, 2014, pp. 7520–7524.

[4] R. Szeliski and D. Scharstein, "Sampling the Disparity Space Image," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 26, no. 3, pp. 419–425, March 2004.

[5] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda, "Classification and Evaluation of Cost Aggregation Methods for Stereo Correspondence," in *IEEE Conf. on Computer Vision and Pattern Recognition*, June 2008, pp. 1–8.

[6] Daniel Scharstein and Richard Szeliski, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," *Int. Journal of Computer Vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[7] G. Falcao, V. Silva, L. Sousa, and J. Andrade, "Portable LDPC Decoding on Multicores Using OpenCL [Applications Corner]," *IEEE Signal Proc. Mag.*, vol. 29, no. 4, pp. 81–109, July 2012.

[8] Andreas Geiger, Philip Lenz, and Raquel Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2012.

[9] Klaus Denker and Georg Umlauf, "Accurate Real-Time Multi-Camera Stereo-Matching on the GPU for 3D Reconstruction," *Journal of WSCG*, vol. 19, no. 1, pp. 9–16, 2011.

[10] J. Kowalczuk, E.T. Psota, and L.C. Perez, "Real-Time Stereo Matching on CUDA Using an Iterative Refinement Method for Adaptive Support-Weight Correspondences," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 23, no. 1, pp. 94–104, Jan 2013.

[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 *IEEE Int. Conf. on Image Processing*, 2012, pp. 1697–1700.