AMS-HMI12: Assisted mobility supported by shared-control and advanced human-machine interfaces
RECI/EEI-AUT/0181/2012
Partners: ISR-UC (Principal Contractor), UC, APCC, IPT
Period: 1/1/2013 -31/12/2015

Technical Report

Fast Parallel SymStereo and Uncapacitated Facility Location Kernels applied to Piecewise Planar 3D reconstruction using hybrid CPU/GPU and multi-GPU architectures

Authors: Carlos Graça
Supervisors: Gabriel Falcão, Urbano Nunes

Abstract: This work consists in benchmarking and accelerating 3D Piecewise Planar Reconstruction (PPR) [1], using hybrid CPU + GPU and multi-GPU parallel computational models and architectures. 3D reconstruction algorithms are composed by distinct time computing expensive tasks, thus we selected the two computationally most intensive ones to parallelize: SymStereo (Stereo Matching using Induced Symmetry) and Uncapacitated Facility Location (UFL). Experimental results show that the accelerated SymStereo algorithm running on multi-GPU systems is on average 39 times faster than the original CPU version and is able to process 29 frames per second. In UFL calculations, the proposed multi-GPU system is on average 35 times faster than the original CPU approach.

Coimbra, December 15th 2015

I. INTRODUCTION

This task consists in accelerating a pipeline for 3D stereo reconstruction that combines the benefits of Piecewise-Planar Reconstruction (PPR) and plane-based odometry by recovering both structure and motion from plane-primitives [1]. The algorithm receives as input an image sequence acquired by a calibrated stereo rig and outputs 3D planes in the scene and the camera motion. Current stereo methods still face difficulties in handling situations of:

- Weak or repetitive texture;
- Variable illumination;
- High surface slant.

PPR methods are employed to overcome these issues. This new method is based on planes detection, thus, unlike the 3D Reconstruction methods based on clouds of points, we describe a 3D model using much less information. As an example, we take a facade of a building as a plane, which can be represented simply by using three XYZ-coordinates. If instead we use clouds of points, we would have millions of points to describe the same facade and three XYZ-coordinate values per single point. With this, not only we can reduce the intensive processing workload, as we are also capable of reducing bandwidth requirements for accessing the data volume (e.g. if we wish to disseminate the 3D model through the Internet).
Also, estimating the motion from plane correspondences is advantageous since plane features are much less numerous than point features. Man-made environments are often dominated by large size planes, leading to:

- Fast correspondence and scalability;
- Correspondence across wide baseline images;
- Resilience to dynamic foreground.

In this work, we develop a full C++ version of planes detection and optimization algorithm used in PPR. With this C++ code, we test and benchmark all functions in order to identify the most time-expensive or relevant task, thus selecting to parallelization these candidates: SymStereo and UFL. This parallelization was developed using CPU + single-GPU and multi-GPU architectures under CUDA 7.5 framework [2].

A. *Related Work*

Other works can build 3D models using a batch of images acquired by a monocular camera using PPR methods [4, 5, 6]. These methods use all images in simultaneous to compute the model. Alternatively, our new PPR method [1] can create a 3D model from a sequence of stereo images using a sliding window approach to concatenate the contributions of consecutive stereo pairs.
In the last few years, GPU parallel processing techniques have been applied to diverse areas of image processing and computer vision, as for example: graph cuts, canny edge detection or stereo matching [7, 8, 9, 10]. Regarding the specific topic of GPU computing and 3D reconstruction we can find some interesting works [11, 12, 13].

B. *Contribution/Development*

The main contributions of this work can be summarized as:

- Single- and multi-GPU parallel solutions that calculate symmetry-based sparse stereo matching (the logN variant of the SymStereo algorithm);
- A fast message-passing algorithm (max-sum) for solving the Uncapacitated Facility Location (UFL) under single- and multi-GPU assemblies.

II. BACKGROUND MATERIAL

A. *Algorithm description*

Below, are shown parts of the algorithm that illustrate the operational basis of the 3D reconstruction algorithm.

Input: Single stereo pair

Figure 1: Input stereo pair (left and right images)

Intermediate output:

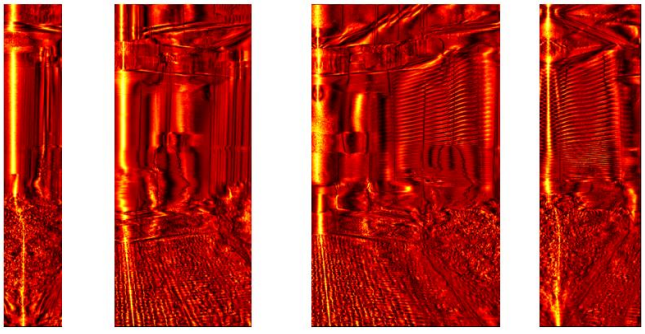

Figure 2: Example of SymStereo energies

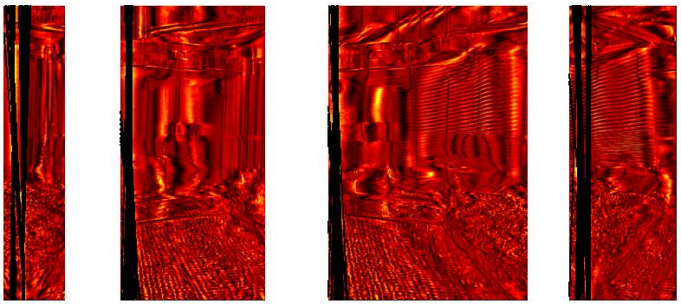

Figure 3: Example of Hough transform to detect lines in energies obtained through SymStereo

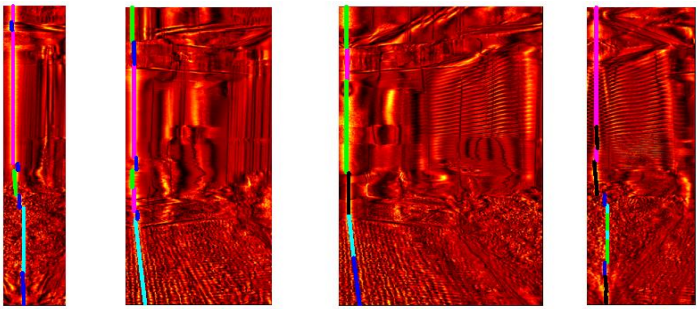

Figure 4: MRF optimization: applied to energy results from SymStereo and Hough lines detected. This step selects the line segments that are better candidates to define each virtual cut plane.

In Figures 1 and 2 we observe the input images (stereo pair images) and SymStereo energies output [1, 3]. To these energies we apply a Hough Transform (depicted in Fig. 3) with a Markov Random Field (MRF) optimization in order to obtain the best line segment candidates to define each virtual cut plane (please see Fig. 4). By applying 3D reconstruction of these line segments and then each set of two lines provides a plane hypothesis. Finally, all plane hypotheses are optimized in order to select the best options has made using UFL algorithm [1] obtaining:

- Final planes that represent the scene (Fig.5).
- Final 3D model (Fig. 6 and 7)

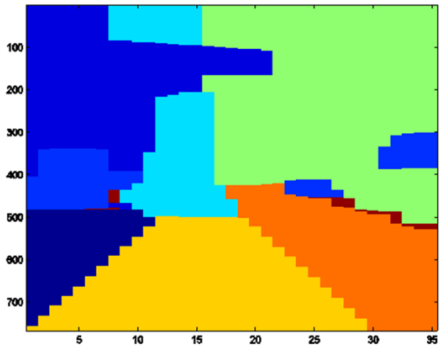

Figure 5: Planes detection result

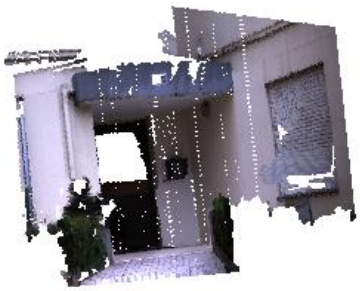

Figure 6: 3D Final Model

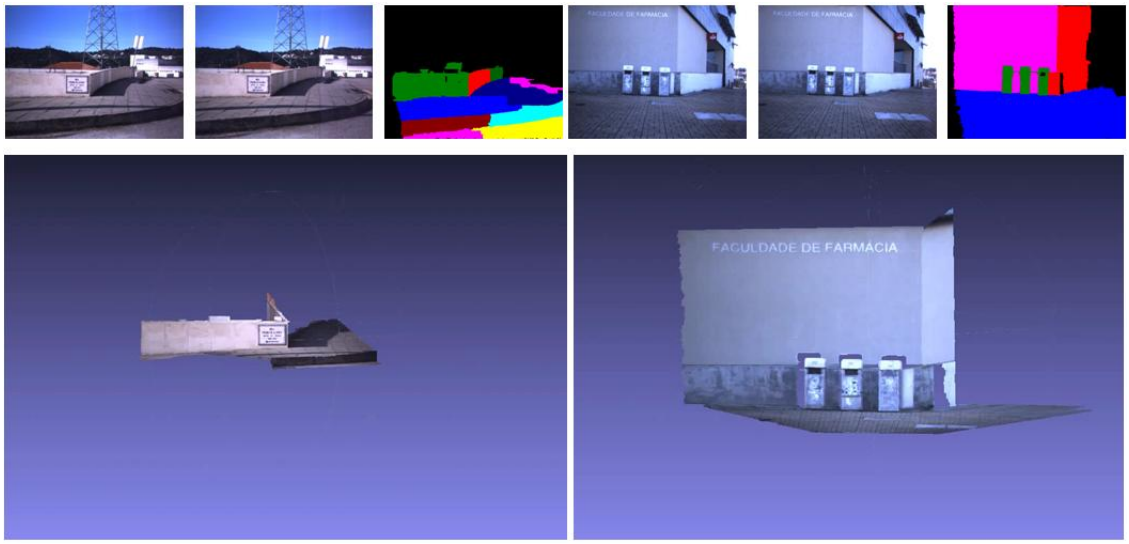

Figure 7: 3D Models obtained using single stereo pair

Input: Stereo pair sequence

By applying the full algorithm for 3D reconstruction in a stereo sequence we obtain the final 3D model as depicted in Fig. 9 (Fig. 8 shows the left image from the input sequence).

Figure 8: Input stereo sequence (only left image)

Figure 9: Final 3D model (generated from the input sequence)

B. *GPU Computing*

Recently, the common desktop Central Processing Units (CPUs) started incorporating a few cores (typically 4 / 8 cores) mostly optimized for sequential execution, while Graphics Processing Units (GPUs) represent massively parallel architectures including thousands of smaller processing units (or cores) and very high-memory bandwidths (Fig. 10).

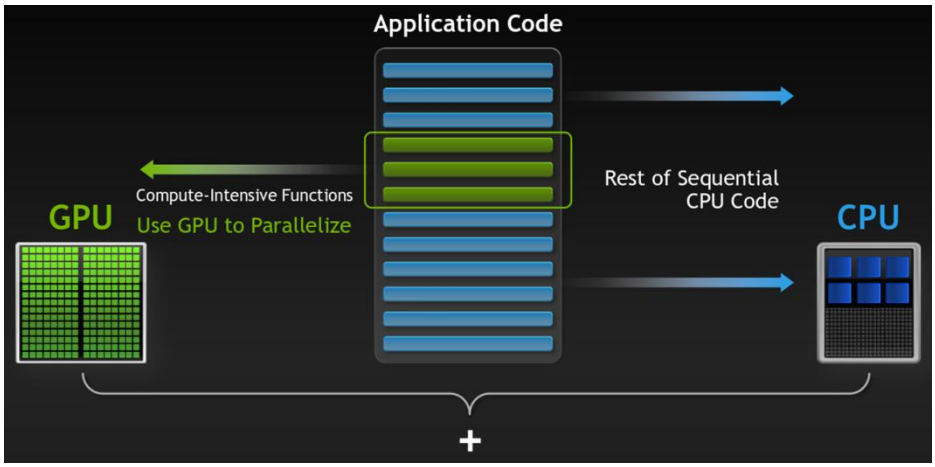

Figure 10: CUDA application schema: Workload distribution between GPU and CPU

These GPU architectures are becoming increasingly efficient for dealing with compute-intensive workloads, offering high speedups when compared to CPUs. With this, it is possible to process several pixels of an image concurrently. To overcome the main obstacle regarding the developments of parallel programs on GPUs, the CUDA programming interface and framework disseminated worldwide a new parallel programming style based on extensions to the C language which is accessible to a broader community of programmers and new application models.

When performing GPU computing, firstly we need to transfer data to be processed to the GPU memory, launch parallel execution kernels on the device and then collect results from the GPU back to the CPU again. All these steps are orchestrated by the CPU (also called host) (see Fig. 11).

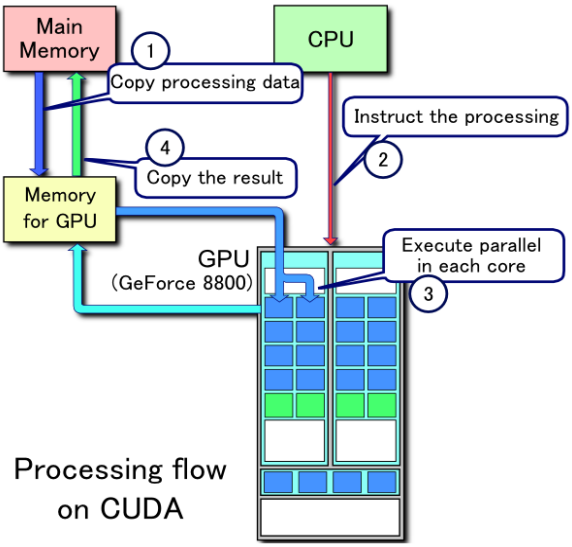

Figure 11: Simple CUDA workflow illustration

III. METHOD / NEW DEVELOPMENT

MULTI-GPU SCHEMA

By exploiting the use of GPUs, we develop C/C++ code that runs in parallel inside the GPU card. Also, propose multi-GPU systems that besides the internal parallelism of the GPU, exploit the simultaneous execution of distinct GPUs in simultaneous (Fig. 12).

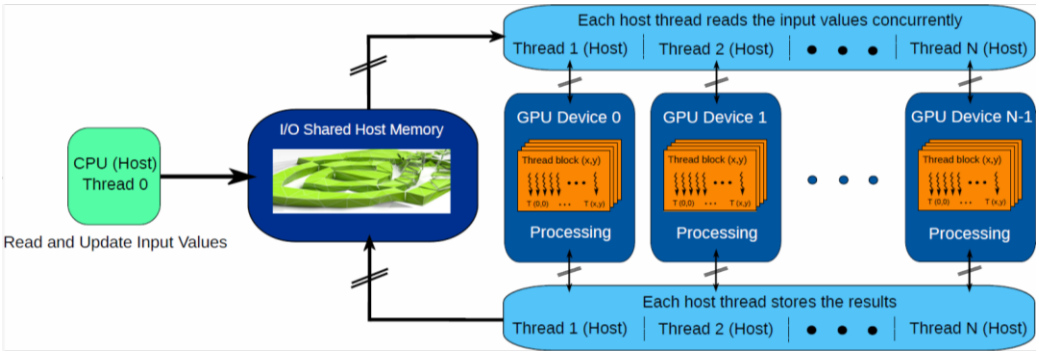

Figure 12: Multi-GPU system orchestrated by a master CPU (thread 0 in the figure)

The second approach raises the degree of parallelism that these machines are capable to offer. In Fig. 12 we can observe a host thread reading input data, which is stored in the shared memory zone. Shared memory can be seen by other threads and accessed concurrently. Thus, creating one host thread per GPU we are able to run the same parallel code in several GPUs.

IV. EXPERIMENTAL/SIMULATION SETUP

Below, we show the external software and hardware setup used. Table 1 shows the processing platform used to run the planes detection pipeline on CPU, while Table 2 and 3 show the GPUs used for single- and multi-GPU configurations.

CPU Versions (Under Intel Core i7-4790k)
MATLAB (R2013b)
Sequential C (g++ v4.4.7)

Table 1: CPU Platforms

Single-GPU Setup
NVIDIA GTX680
NVIDIA Tesla K40c
NVIDIA GTX TITAN
NVIDIA GTX TITAN X

Table 2: GPU Hardware for single-GPU

Multi-GPU Setup
GTX680 / Tesla K40c
Tesla K40c / GTX TITAN
Tesla K40c / GTX TITAN X
GTX TITAN / GTX TITAN X
Tesla K40c / GTX TITAN / GTX TITAN X

Table 3: GPU Hardware for multi-GPU

Software Requirements:

- MATLAB R2013b
- OpenCV 2.4.11
- GCO Library v3.0
- Vlfeat v0.9.16
- LSD v1.6 (Line Segment Detector)
- CUDA Framework 7.5

Code Compilers:

- g++ v4.4.7
- gcc v4.4.7
- nvcc v7.5.17

V. EXPERIMENTAL/SIMULATION RESULTS AND DISCUSSION

Naturally, it wouldn't be fair comparing execution times using MATLAB on the CPU against parallel kernels running CUDA on the GPU. Therefore, it was initially constructed a new version of planes detection algorithm coded in C/C++ to use in this 3D reconstruction pipeline (see Table 4). This sequential C version helps to identify the time-expensive tasks (Table 5).

A. *Planes detection running on CPU*

Processing Platform under Intel Core i7-4790k	Execution time (ms)
MATLAB	82697
C Sequential	42722

Table 4: Time comparison between MATLAB and C sequential code

Function under C code running on Intel Core i7-4790k	Executions Time (ms)
Uncapacitated Facility Location (Message Passing)	31890
Hough Transform	5575
Graph Cut Optimization	2611
SymStereo (logN)	1365
Time SUM	41441 (97%)

Table 5: Execution times for different tasks running under CPU

According to the information presented in Table 5, we choose the UFL algorithm to be parallelized because it represents the more expensive task. The second choice was SymStereo, which is a new and very promising algorithm that can add real value to computer vision applications if running in real-time.

B. *Parallel SymStereo*

In Tables 6 and 7, we depict SymStereo execution times, frames per second (FPS) and speedups obtained for single- and multi-GPU implementations with default configuration (35 virtual cut planes and images of size 1024x768 pixels). With the fastest single-GPU assembly we are able to compute SymStereo with a throughput of 11 FPS and a speedup associated that represents a gain of 16 times when compared to the CPU approach. In the fastest multi-GPU assembly we achieve 29 FPS, running 39 times faster than the CPU implementation.

Processing Platform	Executions Time (ms)	FPS	Speedup
Intel Core i7-4790k	1365	0.73	N/A
Nvidia GTX680	146.49	6.83	9.20x
Nvidia Tesla K40c	116.5	8.58	11.88x
Nvidia GTX TITAN	104.1	9.61	13.07x
Nvidia GTX TITAN X	83.58	11.96	16.68x

Table 6: SymStereo execution times running on single-GPU comparing against a CPU Intel Core i7-4790k

Processing Platform	Executions Time (ms)	FPS	Speedup
GTX680 / Tesla K40c	62.95	15.89	21.68x
Tesla K40c / GTX TITAN	52.07	19.21	26.22x
Tesla K40c / GTX TITAN X	47.03	21.26	29.02x
GTX TITAN / GTX TITAN X	45.69	21.89	29.88x
Tesla K40c / GTX TITAN / GTX TITAN X	34.34	29.12	39.75x

Table 7: SymStereo execution times running on multi-GPU comparing against a CPU Intel Core i7-4790k

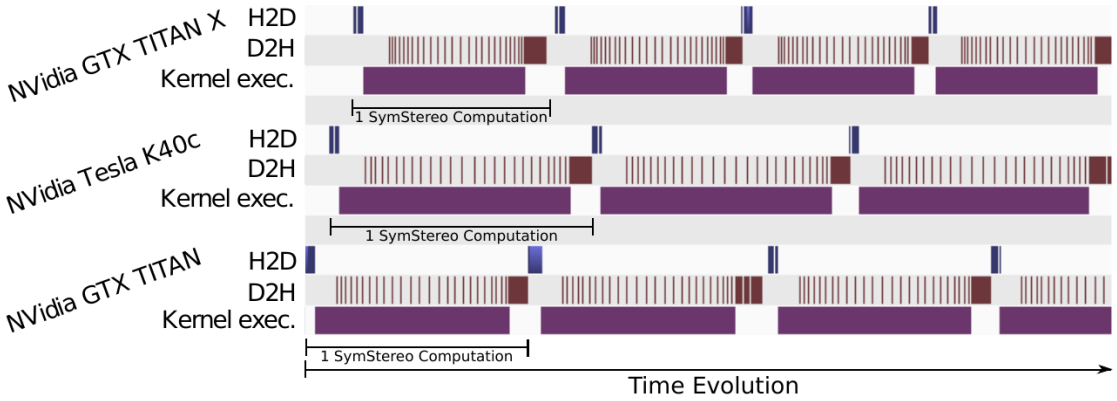

Figure 13: Multi-GPU processing times: workload distribution between GPUs (SymStereo algorithm running in a multi-GPU assembly without significant idle times)

In Fig. 13 we show how multi-GPU system works; each GPU reads a stereo pair of images from host shared memory zone (H2D), processes images (SymStereo kernel execution) and collects the results back to the CPU (D2H).

C. *Parallel Uncapacitated Facility Location*

Through Tables 8 and 9 we show UFL execution times and speedups obtained for single- and multi-GPU implementations with a default configuration (UFL Input Matrix size: 26880x407 and maximum number of iterations = 300). With the fastest single-GPU assembly we are able to compute UFL with a speedup associated 15 times faster than CPU implementation and 35 times faster in the best multi-GPU configuration.

Processing Platform	Executions Time (ms)	Speedup
Intel Core i7-4790k	31890	N/A
Nvidia GTX680	4742	6.73x
Nvidia Tesla K40c	3269	9.76x
Nvidia GTX TITAN	2882	11.06x
Nvidia GTX TITAN X	2001	15.94x

Table 8: UFL execution times and speedup running on single-GPU comparing against a CPU Intel Core i7-4790k

Processing Platform	Executions Time (ms)	Speedup
GTX680 / Tesla K40c	1945.52	16.39x
Tesla K40c / GTX TITAN	1562.99	20.40x
Tesla K40c / GTX TITAN X	1261.21	25.29x
GTX TITAN / GTX TITAN X	1245.63	25.60x
Tesla K40c / GTX TITAN / GTX TITAN X	908.91	35.09x

Table 9: UFL execution times and speedup running on multi-GPU comparing against a CPU Intel Core i7-4790k

In Fig. 14 for each UFL procedure the GPU reads an input matrix from host shared memory zone (H2D), processes the data (UFL kernel execution) and collects the results back to the CPU (D2H).

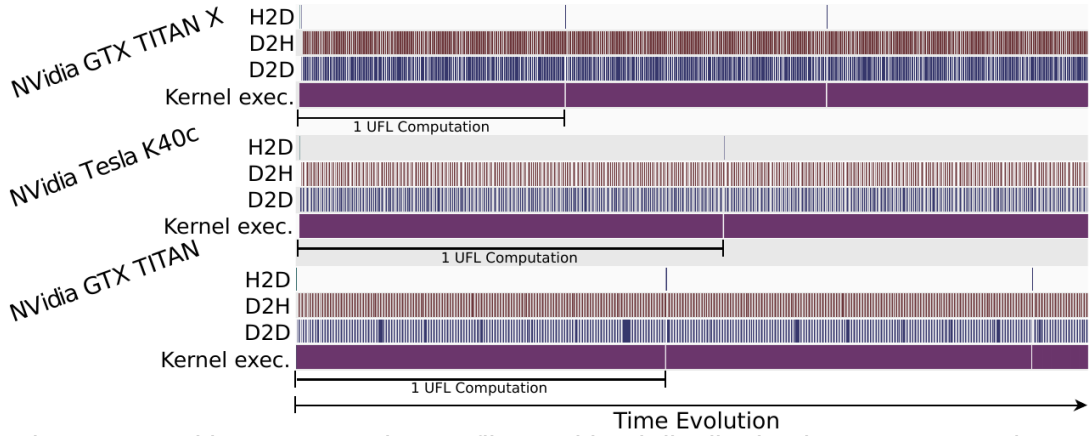

Figure 14: Multi-GPU execution profile: workload distribution between GPUs (UFL algorithm running in a multi-GPU assembly without significant idle times)

VI. WORK IN PROGRESS / FUTURE WORK

A. *Work in Progress*

For the continuous evolution of this research project we need to acquire a large set of new samples of pairs of stereo images that hopefully will allow us to capture the essence of new constraints and test the robustness of the proposed fast 3D reconstruction algorithms. A new dataset of high definition stereo images (3000 x 2200 pixels) has been captured by 2x GoPro HERO 4 in urban areas of Helsinki, Finland, from the top of the buildings. Our plan consists of following an approach that allows the capture of large data collections without (or with minimal) human intervention. For the current approach we plan to use a drone flying over the city with a camera setup

mounted as shown in Fig. 15 and later on automatically applying the eventual radial distortion correction (see Fig. 16).

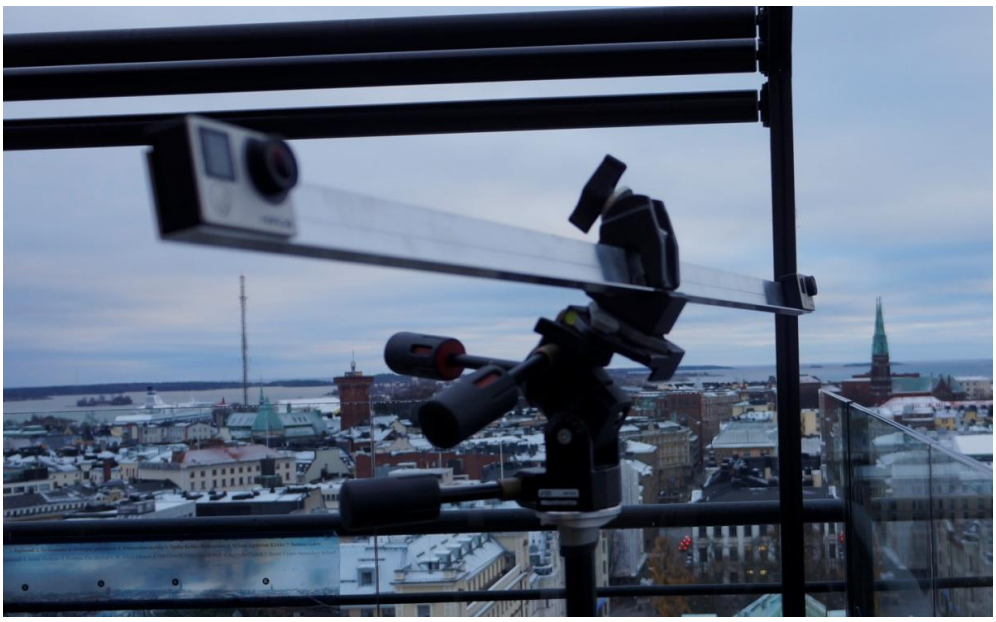

Figure 15: Camera setup to mount in drone

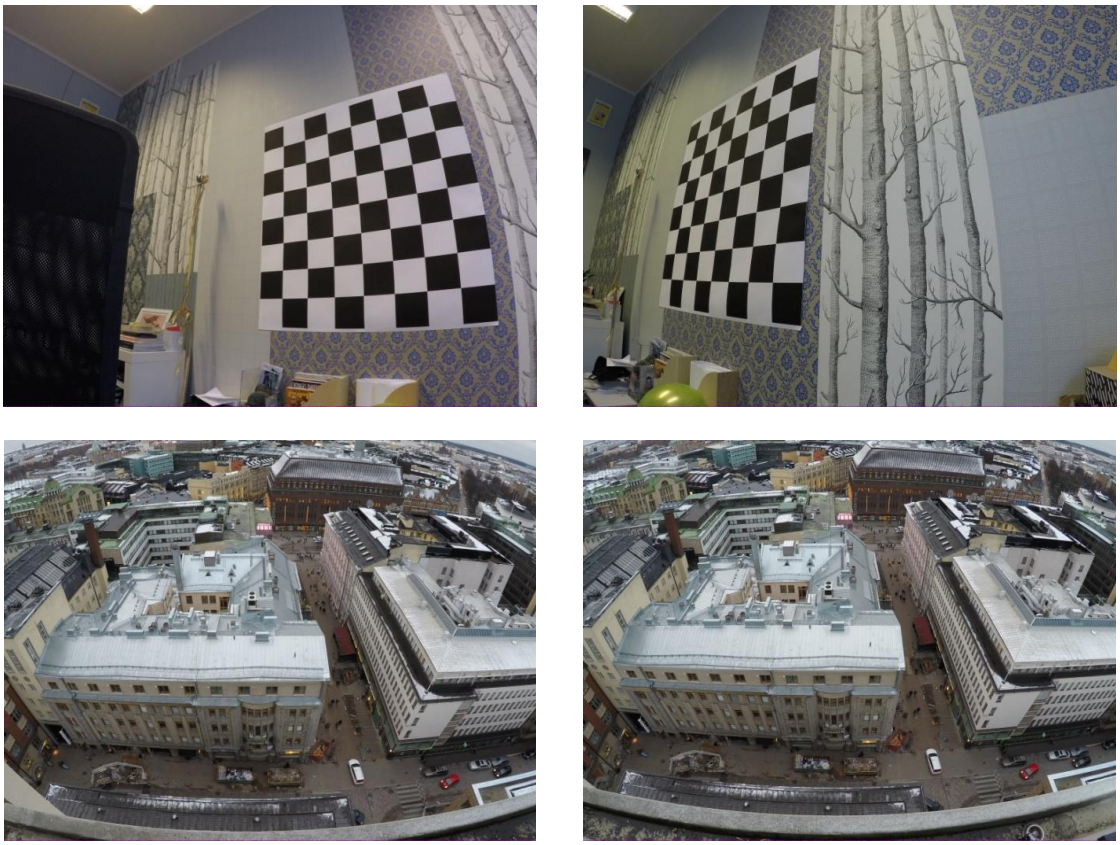

Figure 16: New stereo dataset from urban areas of Helsinki, using a drone to capture stereo images. First row represents calibration images and second row the captured

images

B. *Future Work*

- A larger dataset can be captured with some closed loops to test the algorithm efficiency and continuing the optimization and prototyping of this new PPR method;

- Parallel implementations of SymStereo and UFL algorithms can be integrated into new OpenCV libraries combining CPU and single-GPU implementations, advancing the state-of-the-art in the sense that these new and faster widely used algorithms can be made accessible to the computer vision community;

- Parallelizing the full 3D reconstruction pipeline including the functions identified in this work: Hough transform and graph cut optimization.

VII. CONCLUSION

We develop a parallel framework for SymStereo and UFL computation, which can be helpful in the context of the current project and also in many other computer vision applications. These parallel implementations are carried out by exploiting single- and multi-GPU assemblies, thus providing a significant throughput performance improvement. This novel approach allows processing multiple pixels of an image or matrix entries concurrently, and multi-GPU assemblies in particular allow processing more than one image/matrix at the same time in distinct GPU devices, thus sustaining the obtained throughput levels. By using our best single-GPU implementation (NVIDIA GTX TITAN X), we are able to perform SymStereo computations with a throughput of 11 FPS, obtaining an average speedup of 16.68 times compared to an Intel Core i7-4790k. Regarding multi-GPU architectures, the best configuration from our workstations (Tesla K40c / GTX TITAN / GTX TITAN X) can process 29 FPS with an average speedup of 39.75 compared to the same Intel Core i7-4790k. In UFL acceleration, the fastest single-GPU system can process each matrix with an average speedup of 15.94 times compared to its sequential CPU version. The best multi-GPU configuration tested achieves an average speedup of 35.09 times for the same comparison conditions. With the obtained throughputs we are able to reduce significantly the execution time for entire 3D reconstruction procedure.

REFERENCES

[1] C. Raposo, M. Antunes, and J. P. Barreto, “Piecewise-Planar StereoScan: Structure and Motion from Plane Primitives”, European Conf. on Computer Vision (ECCV’14), 2014.

[2] Podlozhnyuk, V., Harris, M., Young, E.: “NVIDIA CUDA C programming guide”. NVIDIA Corporation (2012).

[3] Michel Antunes and J. P. Barreto, “SymStereo: Stereo Matching using Induced Symmetry”, International Journal of Computer Vision, pp. 1–21, Sep. 2014.

[4] Sinha, S., Steedly, D., Szeliski, R.: Piecewise planar stereo for image-based rendering. In: Computer Vision, 2009 IEEE 12th International Conference on. pp.1881-1888

[5] Gallup, D., Frahm, J.M., Pollefeys, M.: Piecewise planar and non-planar stereo for urban scene reconstruction. In: Computer Vision and Pattern Recognition(CVPR), 2010 IEEE Conference on. pp. 1418-1425

[6] Snavely, N., Seitz, S.M., Szeliski, R.: Modeling the world from internet photo collections. In: International J. Computer Vision 80(2), 189-210 (Nov 2008), http://dx.doi.org/10.1007/s11263-007-0107-3

[7] Vineet, V.; Narayanan, P.J., "CUDA cuts: Fast graph cuts on the GPU," in Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on , vol., no., pp.1-8, 23-28 June 2008 doi: 10.1109/CVPRW.2008.4563095

[8] Yuancheng Luo; Duraiswami, R., "Canny edge detection on NVIDIA CUDA," in Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on , vol., no., pp.1-8, 23-28 June 2008 doi: 10.1109/CVPRW.2008.4563088

[9] Xing Mei; Xun Sun; Mingcai Zhou; shaohui Jiao; Haitao Wang; Xiaopeng Zhang, "On building an accurate stereo matching system on graphics hardware," in Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on , vol., no., pp.467-474, 6-13 Nov. 2011 doi: 10.1109/ICCVW.2011.6130280

[10] Zhiyi Yang; Yating Zhu; Yong Pu, "Parallel Image Processing Based on CUDA," in Computer Science and Software Engineering, 2008 International Conference on , vol.3, no., pp.198-201, 12-14 Dec. 2008 doi: 10.1109/CSSE.2008.1448

[11] Izadi, Shahram, et al. "KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera." Proceedings of the 24th annual ACM symposium on User interface software and technology. ACM, 2011.

[12] Pollefeys, Marc, et al. "Detailed real-time urban 3d reconstruction from video."International Journal of Computer Vision 78.2-3 (2008): 143-167.

[13] Ladikos, A.; Benhimane, Selim; Navab, N., "Efficient visual hull computation for real-time 3D reconstruction using CUDA," in Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on , vol., no., pp.1-8, 23-28 June 2008 doi: 10.1109/CVPRW.2008.4563098