A Near Real-Time Color Stereo Matching Method for GPU

Naiyu Zhang, Hongjian Wang, Jean-Charles Créput, Julien Moreau and Yassine Ruichek
IRTES-SeT, Université de Technologie de Belfort-Montbéliard, Belfort, France
Email: {naiyu.zhang, hongjian.wang, jean-charles.creput, julien.moreau, yassine.ruichek}@utbm.fr

***Abstract*—This paper presents a near real-time stereo matching method with acceptable matching results. This method consists of three important steps: SAD-ALD cost measure, cost aggregation in adaptive window in cross-based support regions and a refinement step. These three steps are well organized to be adopted by the GPU's parallel architecture. The parallelism brought by GPU and CUDA implementations provides significant acceleration in running time. This method is tested on six pairs of images from Middlebury dataset, each possibly declined within different sizes. For each pair of images it can generate acceptable matching results in roughly less than 100 milliseconds. The method is also compared with three GPU-based methods and one CPU-based method on increasing size image pairs.**

Keywords - GPU; Real-Time Stereovision; SAD-ALD; Adaptive Window; CUDA.

I. INTRODUCTION

Stereo matching is one of the most extensively studied problems in computer vision. In years, people take their time into stereo matching algorithm designing for new achievements in the matching accuracy and the processing efficiency. Even with many algorithms introduced every year, the two concerns of accuracy and speed still tend to be contradictory in reported results: accurate stereo methods are usually time consuming. Some algorithms use large support windows for robust cost aggregation [1]–[3], and the disparity computation step is formulated as an energy minimization problem and solved with slow-converging optimizers [4]; In other studies, segmented image regions are used extensively as matching units [5], surface constraints [6], [7] or post-processing patches [8]. These techniques significantly improve the matching quality by paying considerable computation costs. As a result, people tend to look for new possibilities on Graphics Processing Unit (GPU) platforms. However, current accurate stereo algorithms employ some key techniques, which seem not to be suitable for parallel GPU architecture. As proved in [9]–[12], it will be tricky and cumbersome to directly take these techniques into GPU applications: large aggregation windows require extensive iterations over each pixel; some optimization, segmentation and post-processing methods require complex data structures and sequential processing. In spite of these difficulties, GPU is sill promising to fast stereo matching applications because of its powerful parallel computing ability. In fact, there already exist three GPU-based methods, namely, CostFilter [13], PlaneFitBP [12] and ADCensus [14] in the top 20 of Middlebury repository [15]. They all have near real-time performance. Trying to outperform them, we propose a brand new method, which also has near real-time performance, but simple computing components. It can naturally scale according to the image sizes. Our method is a correlation-based technique, which falls into the class of local dense stereo matching approaches. It includes the following key characters:

- SAD-ALD cost measure combining the adapted Sum of Absolute Differences (SAD) measure and the Arm-Length-Differences (ALD) measure. The usage of ALD is inspired by the similarity of the pixel's support region and that of its homologous pixel. This combined measure provides more accurate matching results than common aggregation methods.
- Improved cross-based regions for efficient cost aggregation. Proposed by Zhang *et al.* [16], the support regions are based on cross skeletons with accurate cross construction and cost aggregation by reusing middle-ranking disparity data.
- A simple refinement process with support region voting. This simple process proves to be quite effective with little time consuming.
- Efficient system implementation on GPU with Compute Unified Device Architecture (CUDA).

This paper is organised as follows. Section II presents the stereo-matching steps of the proposed method. Section III details the experiment carried out on a set of increasing size images from Middlebury dataset, with regard to both GPU and CPU methods. Section IV concludes the paper.

II. STEREO MATCHING STEPS

A. Adapted Matching Cost

The method is a local dense stereo matching method. It respects the very assumption that color information of the neighbors of a left pixel should be close to those of the same neighbors of its homologous right pixel in the right image. So, the matching costs are defined between the left pixel and the candidate right pixels in the corresponding line (epipolar line) in the right image. The cost is shifted over all possible pixels so that a matching cost between the left pixel and each candidate in the right image is obtained. By the aggregation of matching cost and the winner-takes-all method, the final disparity estimation is realized by selecting the candidate pixel with the lowest matching cost.

 ISBN: 978-1-61208-310-0

The SAD matching cost is adapted as (1).

$$SAD(x_l, y, d) = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix} \times \begin{bmatrix} |I_{lR}(x_l, y) - I_{rR}(x_l - d, y)| \\ |I_{lG}(x_l, y) - I_{rG}(x_l - d, y)| \\ |I_{lB}(x_l, y) - I_{rB}(x_l - d, y)| \end{bmatrix} \quad (1)$$

In the formula, $|\cdot|$ is the absolute value, d is the spatial shift along the horizontal epipolar line (or we call it the disparity of the two pixels), and $|I_{li}(x_l, y) - I_{ri}(x_l - d, y)|_{(i=R,G,B)}$ are the absolute difference (AD) of three color components in the two chosen pixels. For the coefficient matrix of the three color components, we choose the $\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$ according to the Bayer Filter Mosaic [17], which uses twice as many green elements as red or blue to mimic the physiology of the human eye.

Since these pixels of horizontal lines with the same parity in left and right color images are characterized by the same three color components, we can reasonably assume that the color points of two homologous pixels are similar. Because the matching cost compares the color points of left and right pixels located on the same horizontal lines, they reach an extremum when the shift is equal to the disparity.

An assumption for matching cost aggregation is that a pixel and its homologous pixel should have the similar support region arm-length in vertical direction as shown in Fig. 1. This implies that the arm-length data can be used to enhance the matching results in most regions of the image pairs. So, we update the matching cost equation as (2),

$$MatchingCost(x_l, y, d) = SAD(x_l, y, d) + K \times \sum_j |AL_{lj}(x_l, y) - AL_{rj}(x_l - d, y)| \quad (2)$$

where $j = (\textit{Up arm, Bottom arm})$

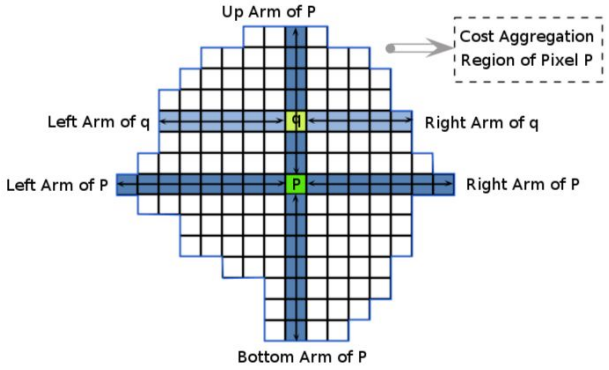
In the formula, $|AL_{lj}(x_l, y) - AL_{rj}(x_l - d, y)|$ is the difference of their arm-length (AL) in vertical direction, and the parameter K is a empirical preset value. The arm-length (AL) is equivalent to absolute difference of their coordinates in the same direction ($|x_p - x_{p'}|$ or $|y_p - y_{p'}|$ for pixel p and one of its endpoint pixel p', detail in Section *B*).

We find that this enhancement can evidently reduce the errors in most regions (the advantage degrades in those areas where both the color and the shape repeat).

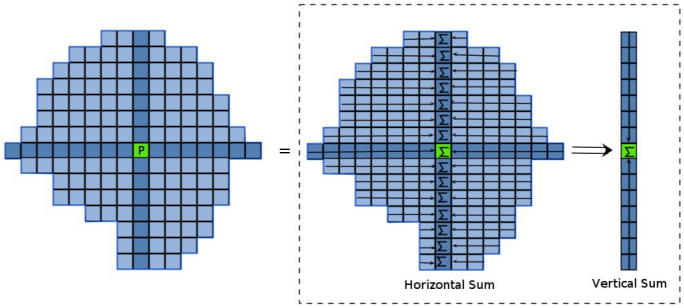
B. Cross-based Cost Aggregation

In this step, each pixel's matching cost over its support region is aggregated, to reduce the matching ambiguities and noise, in the initial cost volume in order to pick out the best candidate pixel. As it is mentioned in the previous section, the matching method respects the very assumption that the color information of neighbors of a left pixel is close to those of the same neighbors of its homologous right pixel in the right image. Meanwhile, for matching cost aggregation, there is also two simple but effective assumptions, the first is the one that a pixel and its homologous pixel should have the similar support region arm-length in vertical direction, as shown in Fig. 1. For updating the matching cost equation, the second one is that neighboring pixels with similar colors should have similar disparities. Many well known cost aggregation methods have adopted this assumption such as segment support [6], adaptive weight [3], geodesic weight [1] and those proposed for GPU systems such as simplified adaptive weight techniques with 1D aggregation [11], [18] and color averaging [9], [12], either require too many segmentation operations, expensive iterations or lead to matching quality decrease owing to maladjustment of GPU's parallel architecture. Some other researchers have formulated the matching cost step as a cost filtering problem [13] and made the matching quality be well guaranteed by smoothing each cost slice with a guided filter.

Zhang *et al.* [16] have proposed a cross-based matching cost aggregation method. This method can be adopted to G-PU's parallel computation architecture and produce aggregated results comparable to the adaptive weight method but with less computation time. Moreover, this method constructs a support region for every pixel, which can provide supplemental information for later processing steps.



(a) Cross Construction



(b) Cost Aggregation

Fig. 1: Cross-based aggregation.

As it is shown in Fig. 1(a), cross-based aggregation is carried out by a two-step-process. In the first step, an upright cross in constructed for every pixel. The support region of a given pixel, such as p in the Fig. 1(a), is modeled by merging the horizontal arms of the pixels (such as the pixel q in Fig. 1(a)) lying on the vertical arms of pixel p. Generally, every pixel has four arms and the length of the arms is set by an endpoint pixel p' in the same direction that does not obey both

 ISBN: 978-1-61208-310-0

the two following rules:

1) $D_c(p,p') < \sigma_{dc}$ and $D_c(p',p'') < \sigma_{dc}$, where $D_c(p,p') = max|I_i(p) - I_i(p')|_{(i=R,G,B)}$ is the color difference between the pixel p and the pixel p' , and p'' is the predecessor of p' lying between p and p' while σ_{dc} is a empirical preset threshold value.
2) $D_s(p,p') < \sigma_{ds}$, where $D_s(p,p') = |p-p'|$ is the spatial distance, which is equivalent to absolute difference of their coordinates in the same direction ($|x_p - x_{p'}|$ or $|y_p - y_{p'}|$) while the σ_{ds} is a empirical preset maximum length measured in pixels.

These two rules provide constraints in the four arm directions both on color similarity and arm length with parameter σ_{dc} and σ_{ds}. After the cross construction step, the support region for pixel p is modeled by merging the horizontal arms of all the pixels lying on p's vertical arms (as done for q for example in Fig. 1(a)). In the second step, shown in Fig. 1(b), the aggregated costs over all pixels are computed by firstly summing up the matching costs horizontally and secondly summing up these horizontal sum results vertically to get the final costs.

In some too textured regions, the color and the shape both repeat, which leads to degradation in matching results. We find that the reason for this degradation lies in the shape of aggregation support region. As seen in Fig. 1(a), we take the pixel at the end of pixel p's right arm as example, for this pixel, its up arm and bottom arm could be very short (less than 2 pixels), so, in the aggregation of its matching cost, there will be not enough information to achieve a unique minimum in the Winner-Takes-All processing, which leads to marching errors at this pixel. As a solution to this problem, we artificially enlarge the arms of a pixel to two pixels if its support region is too small, to make sure that the matching cost aggregation processing can have enough information for stereo matching. This operation provides a slight improvement in matching results in paying no computation time cost.

C. Simple Refinement

After the previous step, the disparity results of both the left image and the right image contain some outliers in certain regions that should be corrected by further operations. A simple refinement is carried out after detecting these outliers.

The outliers in the left image are detected with left-right consistency check: for a given pixel p, it is classified into outliers if this equation does not hold true: $\hat{d}_{Lp} = \hat{d}_{R(p-\hat{d}_{Lp})}$ where $\hat{d}_{Lp}$ is the estimated disparity for pixel p in the left image and $\hat{d}_{Rp}$ is the estimated disparity for pixel p in the right image.

These detected outliers are these errors that should be corrected. The most current accurate stereo matching algorithms use segmented regions for outlier handing [7], [8], which are not suitable for GPU architecture. Here, what we use is a simple voting refinement in reusing the support region information. We still take the pixel p in the left image as example, all the reliable disparities lying in its cross-based support region are sorted by their disparity values. The disparity value, which repeats the most (has the most votes), is denoted as $\hat{d'}_{Lp}$, its repeating frequency is denoted as $F_p(\hat{d'}_{Lp})$, the number of reliable pixels are denoted as S_p^r and the total number of pixels in its support region are denoted as S_p. The disparity value of outlier pixel p is then replaced with $\hat{d'}_{Lp}$ if these inequations hold true: $\frac{F_p(\hat{d'}_{Lp})}{S_p^r} > \sigma_F$, $\frac{S_p^r}{S_p} > \sigma_S$. If not, the p's disparity will be updated with nearest reliable disparity [19] in its support region.

These parameters are given in Table I, which will be kept constant in all the following experiments.

TABLE I: EXPERIMENT PARAMETERS.

K	σ_{dc}	σ_{ds}	σ_F	σ_S
1.12	12	10	0.4	0.55

III. EXPERIMENTS

A. Platform and CUDA Employments

Our experiments are carried out both on CPU and GPU. For the CPU, it is a Intel(R) Core(TM)2 Duo CPU E8400, 3.00GHz, each of the two cores with a cache of 6144KB. The GPU used in the experiments is a GPU GeForce GTX 570 of NVIDIA. It has 15 multiprocessors of 32 cores, the total amount of global memory is 1280 Mb (constant memory 65536 Kb, shared memory per block 49152 Kb). The system, on which the experiment is evaluated, is Ubuntu 11.04, 32 bits.

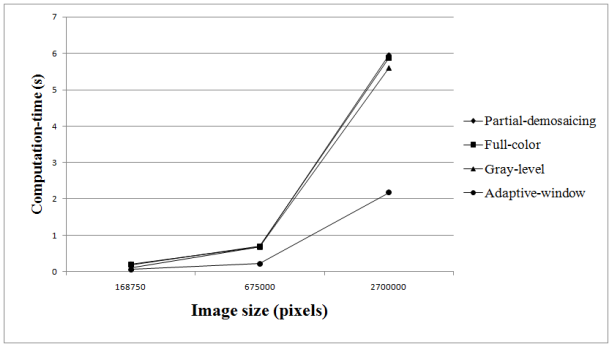
The programming interface we used for parallel computation on GPU is the Compute Unified Device Architecture (CUDA). The parallel computation work is realized by a *kernel* function which is executed concurrently by multiple threads on data elements. All these threads are organized into a two level concepts: *grid* and *block*. A *kernel* has one *grid* which contains multiple *blocks*. Every *block* is formed of multiple threads. The dimension of *grid* and of *blocks* can be one-dimension, two-dimension or three-dimension. The performance of GPU with CUDA is closely related to thread organization and memory accesses, which should attract much attention according to various computation works and GPU platform. Based on our experimental platform, given an image of size $W \times H$, we briefly lay out the CUDA settings and the parameter values mentioned in our algorithm.

In our experiments, we use two-dimension *blocks* of size 16×16. Every thread takes care of one pixel at the three steps: matching cost computation, aggregation of the matching cost and refinement. So, for the $W \times H$ image, there are three grid each containing $W \times H$ threads distributed to the three matching steps and the size of *grid* is obtained by $\frac{W+16-1}{16} \times \frac{H+16-1}{16}$. In the cost computation step, a *grid* is created with $W \times H$ threads, every thread takes care of one pixel for the matching cost value computation at a set of given disparities. A logical 3D memory space will be employed to store matching costs obtained by this grid for the following steps.

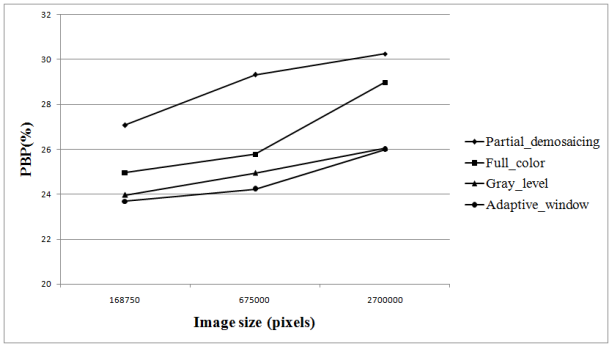
 ISBN: 978-1-61208-310-0

TABLE II: COMPARATIVE EVALUATION ON THE FOUR MIDDLEBURY DATA SETS OF THREE DIFFERENT SIZES BY DIFFERENT GPU PROGRAMMING VERSIONS. 'COMPUTATION TIME (CT)' AND 'PERCENTAGE OF BAD PIXELS (PBP)' .

Image	Size	Partial_Demosaiced		Full_Color		Gray_level		GPU_Ada	
		PBP	CT (s)	PBP	CT (s)	PBP	CT (s)	PBP	CT (s)
Rocks1	Small size	28.76	0.214	25.81	0.191	24.58	0.146	24.28	0.068
	Half size	32.24	0.679	27.87	0.715	25.82	0.913	24.91	0.239
	Full size	37.06	5.811	31.20	5.717	28.93	8.235	28.40	2.275
Aloe	Small size	27.08	0.214	24.96	0.192	23.96	0.115	23.69	0.063
	Half size	29.33	0.698	25.78	0.697	24.94	0.68	24.22	0.217
	Full size	30.26	5.955	28.98	5.875	26.04	5.595	25.99	2.180
Cones	Small size	39.18	0.234	29.77	0.229	28.76	0.276	18.90	0.079
	Half size	46.98	2.123	37.36	1.548	36.91	3.024	35.26	0.812
	Full size	52.14	20.569	44.85	11.502	45.16	19.881	44.18	6.159
Teddy	Small size	45.27	0.23	32.23	0.23	29.86	0.284	23.67	0.112
	Half size	53.69	2.12	38.38	1.55	37.77	3.246	36.09	0.999
	Full size	57.36	27.09	46.58	11.51	47.37	24.922	45.98	6.783
Avg.		39.95	5.495	32.82	3.330	31.68	5.610	28.84	1.666



(a) Comparison on computation-time (s).



(b) PBP(%) comparison results.

Fig. 2: Comparison of the methods on the 'Aloe' image pair.

In the cost aggregation step, a second *grid* of size $W \times H$ is created, so that each pixel has one thread to take care of its matching cost aggregation and then the Winner-Takes-All processing aiming at a winner pixel from a set of candidate pixels, which will be the estimated disparity at the pixel. Here, data reuse with shared memory is considered in this step to reduce the accesses into the global memory space for saving time.

For the simple refinement, the platform does the executions concurrently on the estimated disparity images, a third *grid* of size $W \times H$ is employed to make sure that each pixel has one thread for its refinement processing.

B. Experimental Results

We test our method on the standard image pairs from the Middlebury datasets. Firstly, we take into the test these four pairs of three different sizes (measured in pixels):

- Small size: Rocks1: 425×370; Aloe: 427×370; Cones: 450×375; Teddy: 450×375
- Half size: Rocks1: 638×555; Aloe: 641×555; Cones: 900×750; Teddy: 900×750
- Full size: Rocks1: 1276×1110; Aloe: 1282×1110; Cones: 1800×1500; Teddy: 1800×1500

We first compare our method with three other methods that we implement also on GPU. These methods are the partial demosaicing matching method originally proposed by Halawana [20] and the classic fixed windows matching method, treating full color image and gray-level image separately. The results are shown in Table II. The PBP column reports the percentage of bad pixels, whereas the Computation Time (CT) column reports the computation time in seconds. We can verify that our adaptive method competes with the other ones on all these four pairs both in matching quality and in computation time, as it is shown in Fig. 2 more intuitively. Better performance comes from a new support region and different structure of computation, in comparing with the other methods. The other three methods use square window for cost aggregation. They always have the problem of adjusting the window size: small windows do not contain enough information to allow a correct matching or for a unique minimum in the matching cost, while at the opposite, too large aggregation windows may cover image regions containing pixels with different disparities, which violates the assumption of constant disparity inside the aggregation window. The upright cross support region used in this method has no such weakness. Here, the cost aggregation window could be well adjusted to make sure that only those useful pixels are covered, and it could be big enough to have sufficient information for a good

 ISBN: 978-1-61208-310-0

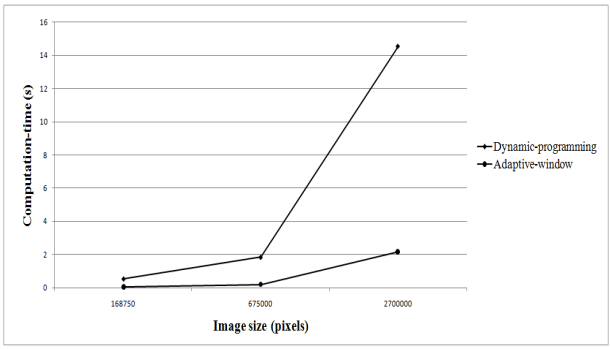
stereo matching result, which offers important contribution to matching quality. Meanwhile, the cost computation, the cost aggregation are organized into two different CUDA *girds*, each does a part of the stereovision work. This partition leads to less memory coalescence but better profit from GPU's massive mathematical capacity, feeding back a short running time. It is worth noting that near-real time computation is achieved for the small size images, since computing time may reduce to about less than 100 ms. Our method puts most of the data on the global memory space and carefully treats the coalescence of memory access with proper programming structures and adapted usage of cached memory space of GPU such as shared memory and texture memory for intermediary data, which makes our method be very extensible and scalable for large image pairs.

TABLE III: COMPARATIVE EVALUATION ON THE FOUR MIDDLEBURY DATA SETS OF THREE DIFFERENT SIZES BY DP AND OUR METHOD. 'COMPUTATION TIME (CT) ' AND 'PERCENTAGE OF BAD PIXELS (PBP)' .

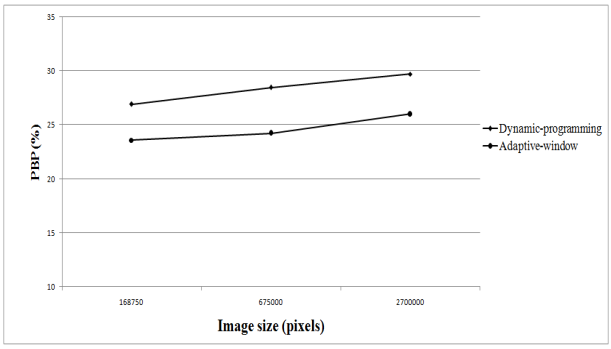
Image	Size	CPU_DP		GPU_Ada	
		PBP	CT (s)	PBP	CT (s)
Rocks1	Small size	25.49	0.509	24.28	0.068
	Half size	27.71	1.811	24.91	0.239
	Full size	27.85	14.194	28.40	2.275
Aloe	Small size	26.90	0.544	23.69	0.063
	Half size	28.46	1.842	24.22	0.217
	Full size	29.72	14.534	25.99	2.180
Cones	Small size	21.57	0.583	18.90	0.079
	Half size	28.89	4.903	35.26	0.812
	Full size	35.32	37.736	44.18	6.159
Teddy	Small size	20.80	0.590	23.67	0.112
	Half size	27.95	4.881	36.09	0.999
	Full size	33.51	37.704	45.98	6.783
Avg.		27.85	9.986	28.84	1.666

We also compare our method with a standard Dynamic Programming (DP) matching method that we implement on CPU, and the results are presented in Table III, and in Fig. 3 more intuitively. These two methods can achieve similar matching quality but our method outperforms the dynamic-programming method in computation time with about five to ten times acceleration. On the four small size image pairs, the dynamic-programming method can finish its work in less than half a second, however, our system does have work done in 100 milliseconds.

Finally, some disparity results are presented in Fig. 4. These results concern the four images allowed in the Middlebury database for general comparison and ranking. These images are the small size images *Tsukuba*, *Venus*, *Teddy* and *Cones*. The ranking evaluations is shown in Fig. 5. Our method gives back the best results for the *Venus* image pair among the four image pairs. Generally speaking, the matching quality of the method is not very competitive in comparing to some other very sophisticated methods on CPU, especially for the *Tsukuba* image pair, in some regions of this image pair, near the shoulder or the lamp for example, the color is too dark and



(a) Comparison on computation-time (s).



(b) PBP(%) comparison results.

Fig. 3: Comparison to Dynamic Programming on 'Aloe' image pair.

the color components' values are far out of the ordinary, which become as the noises to the matching method. Different from the methods on CPU, which take at least half a second to do the stereo matching, our method requires only 0.017 seconds for *Tsukuba* pair, 0.053 seconds for *Venus* pair, 0.079 seconds for *Cones* pair and 0.112 seconds for *Teddy* image pair.

IV. CONCLUSION

This paper presented a stereo matching method suitable to GPU's parallel architecture with good performance when looking at the trade-off between accuracy and computation time. The method is formed of three steps: SAD-ALD cost measure, cost aggregation in adaptive window in cross-based support regions and a refinement step to reduce the matching errors in the disparity results. Every step is well organized so that this method can be adopted efficiently by the GPU's parallel architecture. Experiment results show the accuracy and the efficiency of this method: this method can handle some pairs of images from Middlebury database within roughly 100 milliseconds with acceptable matching quality both in non-occluded regions and depth discontinuities. Furthermore, the approach scales well as the image size increases.

Although the running time is short, the implementation in real time is still a great challenge. As the cost aggregation step takes the biggest proportion of running time, looking for a more efficient way to further accelerate cost aggregation and finding out a set of robust experiment parameters to improve matching quality can be interesting topics for future studies.

 ISBN: 978-1-61208-310-0

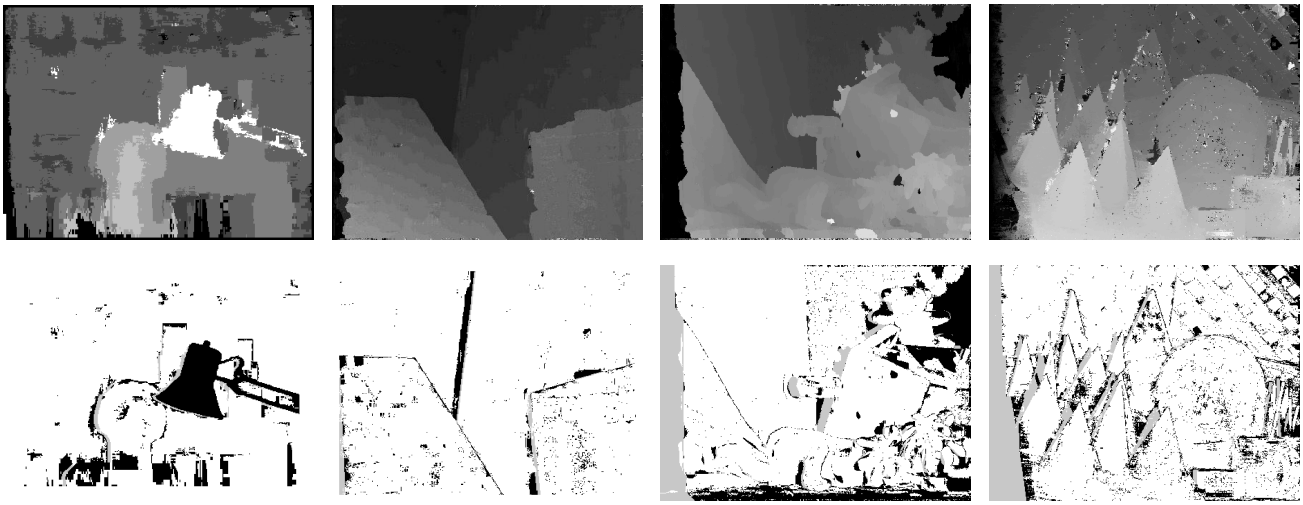


Fig. 4: Matching results for the four basic Middlebury image pairs: Estimated disparity map (first row) and disparity matching error maps (second row) with threshold 1 where the errors in unoccluded and occluded regions are marked in black and gray respectively.

Algorithm	Avg. Rank	Tsukuba ground truth nonocc	Tsukuba all	Tsukuba disc	Venus ground truth nonocc	Venus all	Venus disc	Teddy ground truth nonocc	Teddy all	Teddy disc	Cones ground truth nonocc	Cones all	Cones disc	Average Percent Bad Pixels
FW-DLR [129]	141.4	4.87 139	5.89 133	22.9 148	2.50 130	3.22 128	18.3 132	18.2 149	18.7 130	37.2 150	24.2 153	27.9 152	42.1 153	18.8
SO [1c]	142.9	5.08 141	7.22 145	12.2 113	9.44 150	10.9 150	21.9 139	19.9 151	28.2 153	26.3 135	13.0 147	22.8 150	22.3 141	16.6
MI-nonpara [85]	145.2	5.59 145	7.54 147	18.8 139	7.50 146	8.99 146	35.0 150	17.4 146	25.7 149	36.9 149	10.2 140	19.9 143	22.6 143	18.0
OUR METHOD	145.8	14.4 153	15.5 153	38.2 153	4.60 141	5.95 142	28.3 146	16.3 142	23.9 142	30.8 143	12.1 144	21.7 147	22.9 144	19.6
PhaseDiff [23]	146.2	4.89 140	7.11 143	16.3 131	8.34 149	9.76 149	26.0 144	20.0 152	28.0 152	29.0 141	19.8 152	28.5 153	27.5 149	18.8
STICA [16]	146.4	7.70 151	9.63 152	27.8 150	8.19 148	9.58 147	40.3 153	15.8 140	23.2 141	37.7 151	9.80 137	17.8 136	28.7 151	19.7
Rank+ASW [84]	146.5	6.51 149	8.43 149	19.7 141	10.5 152	12.0 152	32.7 148	15.7 139	24.1 144	32.8 145	14.1 148	23.1 151	21.7 140	18.4
LCDM+AdaptWgt [75]	146.8	5.98 148	7.84 148	22.2 146	14.5 153	15.4 153	35.9 151	20.8 153	27.3 151	38.3 152	8.90 134	17.2 135	20.0 137	19.5
Infection [10]	148.1	7.95 152	9.54 151	28.9 152	4.41 140	5.53 140	31.7 147	17.7 147	25.1 148	44.4 153	14.3 149	21.3 146	38.0 152	20.7

Fig. 5: The rankings in the Middlebury datasets with the error percentages in different regions.

REFERENCES

[1] A. Hosni, M. Gelautz, and C. Rheman, "Local stereo matching using geodesic support weights," *Proc ICIP*, pp. 2093–2096, 2009.

[2] F. Tombari, S. Mattoccia, and L. D. Stefano, "Segmentation-based adaptive support for accurate stereo correspondence," *Proc PSIVT*, pp. 427–438, 2007.

[3] K. J. Yoon and I. S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE TPAMI*, pp. 650–656, 2006.

[4] R. Szeliski and et al., "A comparative study of energy minimization methods for markov random fields with smoothness-based priors," *IEEE TPAMI*, pp. 1068–1080, 2008.

[5] Z. Wang and Z. Zheng, "A region based stereo matching algorithm using cooperative optimization," *Proc. CVPR*, pp. 1–8, 2008.

[6] A. Klaus, M. Sormann, and K. Karner, "Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure," *ICPR*, pp. 15–18, 2006.

[7] Q. Yang, L. Wang, R. Yang, H. Stewenius, and D. Nister, "Stereo matching with color-weighted correlation, hierarchical belief propagation and occlusion handling," *IEEE TPAMI*, pp. 492–504, 2009.

[8] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE TPAMI*, pp. 328–341, 2008.

[9] A. Hosni, M. Bleyer, and M. Gelautz, "Near real-time stereo with adaptive support weight approaches," *Proc. 3DPVT*, 2010.

[10] J. Liu and J. Sun, "Parallel graph-cuts by adaptive bottom-up merging," *Proc. CVPR*, pp. 2181 – 2188, 2010.

[11] Y. Wei, C. Tsuhan, F. Franz, and C. H. James, "High performance stereo vision designed for massively data parallel platforms," *IEEE TCSVT*, pp. 1–11, 2010.

[12] Q. Yang, C. Engels, and A. Akbarzadeh, "Near real-time stereo for weakly-textured scenes," *Proc. BMVC*, pp. 80–87, 2008.

[13] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," *Proc. CVPR*, 2011.

[14] X. X. Mei, " On building an accurate stereo matching system on graphics hardware," *GPUCV*, 2011.

[15] S. A. N. Brad Hiebert-Treuer and D. Scharstei, "2006 Stereo Datasets," *http://vision.middlebury.edu/stereo/data/scenes2006/*, [accessed: 2013-10-14].

[16] K. Zhang, J. Lu, and G. Lafruit, "Cross-based local stereo matching using orthogonal integral images," *IEEE TCSVT*, pp. 1073–1079, 2009.

[17] "Bayer filter," *http://en.wikipedia.org/wiki/Bayer_filter*, March 2012.

[18] X. Sun, X. Mei, S. Jiao, M. Zhou, and H. Wang, "Stereo matching with reliable disparity propagation," *Proc. 3DIMPVT*, pp. 132–139, 2011.

[19] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two- frame stereo correspondence algorithms," *International Journal of Computer Vision*, pp. 7–42, 2002.

[20] H. Halawana, "Partial demosaicing of CFA images for stereo matching," Ph.D. dissertation, Université de Lille 1, 2010.

 ISBN: 978-1-61208-310-0