REAL-TIME LOCAL STEREO MATCHING USING GUIDED IMAGE FILTERING

Asmaa Hosni[1], *Michael Bleyer*[1], *Christoph Rhemann*[1], *Margrit Gelautz*[1] *and Carsten Rother*[2]

[1]Vienna University of Technology, Vienna, Austria [2]Microsoft Research Cambridge, Cambridge, UK

ABSTRACT

Adaptive support weight algorithms represent the state-of-the-art in local stereo matching. Their limitation is a high computational demand, which makes them unattractive for many (real-time) applications. To our knowledge, the algorithm proposed in this paper is the first local method which is both fast (real-time) and produces results comparable to global algorithms. A key insight is that the aggregation step of adaptive support weight algorithms is equivalent to smoothing the stereo cost volume with an edge-preserving filter. From this perspective, the original adaptive support weight algorithm [1] applies bilateral filtering on cost volume slices, and the reason for its poor computational behavior is that bilateral filtering is a relatively slow process. We suggest to use the recently proposed guided filter [2] to overcome this limitation. Analogously to the bilateral filter, this filter has edge-preserving properties, but can be implemented in a very fast way, which makes our stereo algorithm independent of the size of the match window. The GPU implementation of our stereo algorithm can process stereo images with a resolution of 640 × 480 pixels and a disparity range of 26 pixels at 25 fps. According to the Middlebury on-line ranking, our algorithm achieves rank 14 out of over 100 submissions and is not only the best performing local stereo matching method, but also the best performing real-time method.

Index Terms— Real-time stereo matching, Local stereo, Adaptive support weights, Guided image filter.

1. INTRODUCTION

Real-time stereo matching algorithms are important in many applications such as virtual reality, on-line robotics navigation and teleconferencing. Most existing real-time algorithms are local matching algorithms.

Standard local techniques are basically block matching algorithms. These methods work as follows: First, each pixel's color dissimilarity is computed at each disparity level. The color dissimilarities measure the correlation between the left and right images. Color dissimilarities are then aggregated inside a support window centered at a given pixel [3, 4]. Using standard block-based aggregation, each pixel inside the support window has the same influence in the aggregation process. This leads to bad results when the support window overlaps a disparity discontinuity (edge fattening problem).

To overcome this problem, adaptive support weight approaches have been introduced. Such methods were pioneered by Yoon and Kweon [1], who performed color segmentation inside the support window. The idea is that pixels having a color similar to the center pixel are likely to lie on the same object, and therefore have similar depth (disparity). A pixel inside the support window receives a high support weight if it is close in both color and spatial distance to the central pixel of the window. Thus, pixels inside the support window have different influence in the aggregation process. This strategy considerably reduces the edge fattening problem when using large window sizes and accordingly leads to better quality results. This is well reflected in the ranking of the Middlebury benchmark [5].[1]

Many variations of the Yoon and Kweon method have been introduced since then. Some authors focus on the improvement of the color segmentation within the support window. In [6], for example, the authors use the geodesic distance transform to compute the support weights. In their method, a pixel inside the support window receives a high weight if it is connected to the central pixel by a path of approximately constant color. This allows implementing the concept of connectivity in the segmentation step. Other variations of the adaptive support aggregation method have been published in [7, 8]. For more details, discussion and evaluation, the reader is referred to [9] and [10].

Despite the good quality results, adaptive support weight methods cannot be used in real-time applications. This is due to the large amount of time needed in both the pixel-wise support weight computation and the aggregation processes. Unlike standard block matching methods, adaptive weight methods can usually not be speeded up by using the "sliding window" technique described in [11] and [12]. Thus the speed of adaptive support weight methods is directly dependent on the support window size; and to achieve good quality results in such methods, large window sizes should be used.[2]

In recent years, accelerated versions of adaptive support weight aggregation methods have been proposed. In [13], for

Michael Bleyer and Christoph Rhemann received financial support from the Vienna Science and Technology Fund (WWTF) under project ICT08-019.

[1] Many adaptive support weight methods now lie within the top quarter of the ranking table.

[2] The window size is reported to be 33 × 33 in [1] and 31 × 31 in [6].

978-1-61284-350-6/11/$26.00 ©2011 IEEE

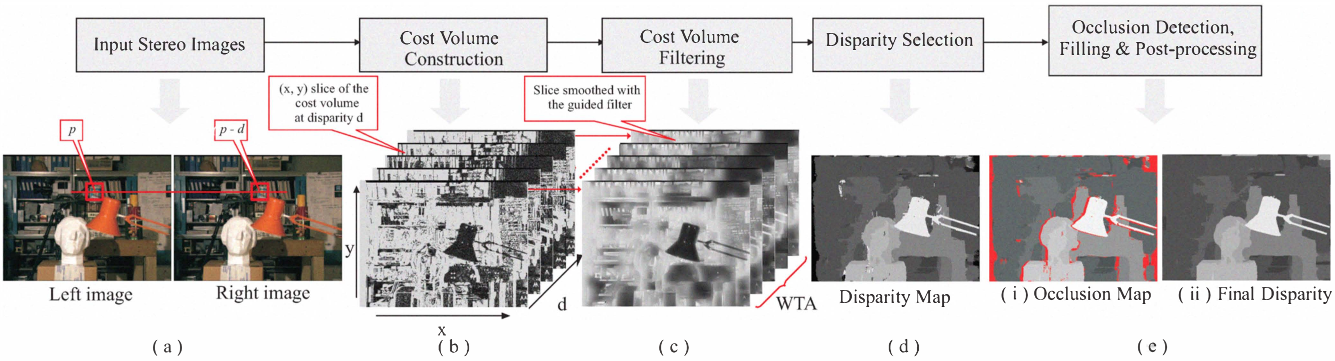


Fig. 1. Outline of the proposed algorithm. (a) Input stereo images. (b) Cost volume construction. Each (x, y) slice holds the color dissimilarity costs at a given disparity value d. Note that the per-pixel costs are very noisy. (c) Cost volume filtering. Filtering is applied on each of the (x, y) slices. Previous methods use filters that either blur disparity borders (mean filter) or are computationally slow [1]. We propose to use the guided filter [2], which enables high-quality real-time stereo matching. (d) Disparity selection. The disparity is computed by applying the Winner-Takes-All strategy, i.e. for each pixel, we select the disparity of lowest costs in the smoothed volume of (c). (e) Occlusion detection, filling and post-processing. Occluded pixels are marked by red pixels in the occlusion map.

example, the authors apply explicit (mean-shift) segmentation to the reference image of the stereo pair. Then they use a modified version of the "sliding window" technique. In this case the aggregation process is independent of the support window size, but the color segmentation of the reference image becomes the new bottleneck of their algorithm. Another attempt to speed up adaptive support aggregation algorithms is presented in [14]. This method removes the bottleneck step of [13] by using the computed support weights for obtaining an explicit color segmentation in a very fast manner. [14] reports near-real-time performance, but the quality of resulting disparity maps is inferior to the state-of-the-art.[3]

There exists a different way for looking at local methods. A key insight used in this work is that we can interpret the aggregation step as smoothing of the stereo cost volume. Here, the cost volume is derived by computing the color dissimilarity for each pixel (x, y) at each allowed disparity d. The resulting three-dimensional structure is also known as disparity space image (DSI) in stereo literature [5]. Smoothing is then applied on two-dimensional (x,y) slices of the cost volume at a fixed disparity value (also see figure 1b and c), and aggregation methods basically differ by the filter kernel that is used to accomplish this smoothing. Let us repeat the above discussion from this point of view.

Standard block-based aggregation corresponds to applying a mean filter on the cost volume.[4] Mean filtering is very fast (due to sliding windows). However, it fails in preserving object borders, which leads to edge-fattening when used in stereo. A better choice in smoothing the cost volume is to apply an edge preserving filter. In this context, the original adaptive support weight method [1] uses the bilateral filter for accomplishing this task. (The authors do not explicitly state this in the corresponding paper.) The advantage is that, via using the bilateral filter, disparity borders are now well preserved. However, the bilateral filter cannot be implemented in a fast way so that run-time is independent of the window size. This is also the reason why [1] can most likely not be implemented efficiently (at least not in an exact, non-approximative way). Obviously, if we had a filter which gives an edge-preserving effect at the same quality as the bilateral filter, but is fast, this would help adaptive support weight stereo methods.

Fortunately, the recently proposed "guided image filter" [2] fulfills these two conditions. The contribution of this paper is to integrate this filter into adaptive support weight stereo. This leads to a powerful algorithm that produces high-quality results at real-time frame rates.

The work that we see as most closely related to ours is [15]. Here, the authors speed-up the original adaptive support weight method [1] by using a fast approximation of the bilateral filter. However, this approximation goes along with reduced quality of filtering results. Hence, the authors could not achieve state-of-the-art stereo results. Besides, due to the approximation, big amounts of memory are required if the method is applied to color images. Therefore this method is usually applicable only for gray-scale stereo images, which results in poor quality disparity maps especially at disparity discontinuities.[5] Although a proposed alternative (in which

[3] The authors achieved rank 32 out of 74 methods on the Middlebury ranking system at the time of their evaluation.

[4] In standard block matching, aggregation is performed by summing up dissimilarities within a window. Similarly, the mean filter applied on (x,y) slices of the cost volume sums up the dissimilarities within the window, but then divides the sum by the window size. This division does not change the optimum in winner-takes-all disparity selection. Hence, block matching and mean filtering on the cost volume are equivalent.

[5] This is clearly reflected in the Middlebury ranking where the method ranks at position 84 out of 100 methods.

two color channels are used) is given in [15], this solution is reported to be 13 times slower than the gray-scale method. In this case, no real-time performance was achieved.

2. ALGORITHM

Figure 1 shows the outline of our algorithm. In the following sections we will go through each step of this diagram in detail.

2.1. Cost Volume Construction

The cost value $C(p,d)$ for pixel p at disparity d is derived by measuring the dissimilarity in appearance of pixel p of the left image and pixel $p-d$ of the right view. In our implementation, we choose the truncated absolute difference of colors and gradients as a dissimilarity measure. This model is borrowed from the optical flow literature. It has been shown to be robust to illumination changes and is commonly used in optical flow estimation [16, 17].

The absolute difference $M()$ of colors is computed as

$$M(p,d) = \sum_{i=1}^{i=3} \left| {I^i}_{left}(p) - {I^i}_{right}(p-d) \right| . \qquad (1)$$

Here, $I^i(p)$ denotes the value of the ith color channel (in RGB space) at pixel p. The absolute difference $G()$ of gradients is expressed as

$$G(p,d) = |\nabla_x(I_{left}(p)) - \nabla_x(I_{right}(p-d))| \qquad (2)$$

where $\nabla_x(I(p))$ denotes the gradient in x direction computed at pixel p. Finally, our overall cost function $C()$ is derived as

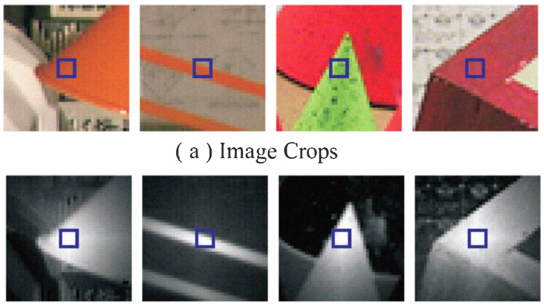
$$\begin{aligned} C(p,d) = \alpha \cdot \min\left(T_c, M(p,d)\right) + \\ (1-\alpha) \cdot \min\left(T_g, G(p,d)\right) . \end{aligned} \qquad (3)$$

Here, α balances the color and gradient terms and T_c, T_g are color and gradient truncation values.

2.2. Cost Volume Filtering

After generating the cost volume, we use the guided image filter [2] to filter each (x,y) slice of the cost volume. The reasons why we choose this filter are:

- It has been shown to have an edge-preserving property.
- It can be implemented in a non-approximate manner which results in good quality results.
- It has a linear running time which is independent of the filter size and thus only depends on the number of image pixels.
- Based on a box filter, it can be efficiently implemented on the GPU architecture thus achieving real-time performance.


(a) Image Crops

(b) Guided Filter Weights

Fig. 2. Examples of computed weights. (a) Reference windows. The center pixels are marked by blue rectangles. (b) Guided filter weights computed for marked pixels in (a).

By using this type of filtering, interactive frame rates can be achieved while keeping the good quality of the results.

The output of the filtering process at pixel p and disparity d is a weighted average of all pixels in the same slice:

$$C'(p,d) = \sum_q W_{p,q}(I) C(q,d). \qquad (4)$$

Here, $C'(p,d)$ denotes the filtered cost value at pixel p and slice d. The filter weights $W_{p,q}$ of the guided filter depend on image I, which is the reference image. In the following, these weights are defined.

In [2], a *guidance* image I is used to filter a *guided* image f. In our case the *guidance* image is the left color image and the *guided* image is an (x,y) slice of the cost volume (as, for example, shown in figure 1b). For color guidance images, the filter weights $W_{i,j}$ are defined as follows:

$$W_{i,j} = \frac{1}{|\omega|^2} \sum_{k:(i,j)\in\omega_k} (1 + (I_i - \mu_k)^T (\Sigma_k + \epsilon U)^{-1} (I_j - \mu_k)). \qquad (5)$$

Here, Σ_k and μ_k are the covariance and mean matrices of I in the window ω_k with dimensions $r \times r$, centered at pixel k.[6] The number of pixels in this window is $|\omega|$ and ϵ is a smoothness parameter. I_i, I_j and μ_k are 3×1 (color) vectors, and the covariance matrix Σ_k and identity matrix U are of size 3×3. The guided filter weights do not have to be computed explicitly. Instead, some linear operations are done:

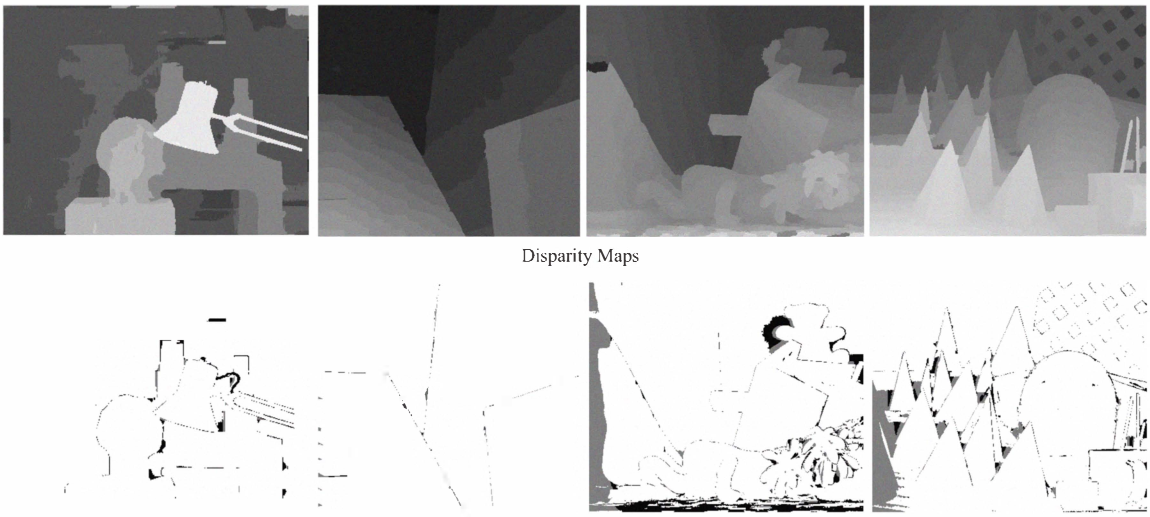
$$a_k = (\Sigma_k + \epsilon U)^{-1} \left(\frac{1}{|\omega|} \sum_{i\in\omega_k} I_i f_i - \mu_k \bar{f}_k\right). \qquad (6)$$

$$b_k = \bar{f}_k - a_k^T \mu_k. \qquad (7)$$

$$q_i = \bar{a}_i^T I_i + \bar{b}_i. \qquad (8)$$

Here, f_i is the pixel i in the guided image f, $\bar{f}_k = \frac{1}{|\omega|} \sum_{i\in\omega_k} f_i$ is the mean of f in ω_k and q_i is the filtered output

[6] The sum in eq. (5) is defined over all windows which include pixel indexes i and j. So the actual size of the filter kernel is $(4r+1)^2$.


Disparity Maps

Error Maps

Fig. 3. Our results using Middlebury images. All results are generated using constant parameter settings. 1^{st} row shows the disparity maps computed by our algorithm. 2^{nd} row shows a comparison against the ground truth maps. Disparity errors larger than one pixel are plotted as black pixels.

with $\bar{a}_i = \frac{1}{|\omega|} \sum_{i \in \omega_k} a_k$ and $\bar{b}_i = \frac{1}{|\omega|} \sum_{i \in \omega_k} b_k$. All summations are box filters and can be computed in $O(N)$ time, where N is the number of image pixels. For more details about the guided filtering see [2].

The guided filter weights for some specified image pixels are shown in figure 2. From this figure we can see that the weights are high in regions which have similar color in comparison with the central pixel of the window, and low otherwise. The edges of the guidance image are well-preserved in the output weights.

2.3. Disparity Selection

Once the cost volume slices are filtered, the Winner-Takes-All strategy (WTA) is applied to choose the best disparity value for each pixel p:

$$d_p = \underset{d \in \mathcal{D}}{\operatorname{argmin}}\, C'(p, d) \tag{9}$$

where $\mathcal{D}$ represents the set of all allowed disparities.

2.4. Occlusion Detection, Filling and Post-processing

We apply left/right consistency checking. Hence, we do not only compute the disparity map for the left image, but also compute a second disparity map with the right image chosen as the reference view. We mark a pixel in the left disparity map as inconsistent, if the disparity value of its matching pixel differs by a value larger than one pixel. This consistency check typically fails for occluded and mismatched pixels (see figure 1e(i)). Inconsistent pixels are then filled by the disparity of the closest consistent pixel.[7]

Our simple filling method produces streak-like artifacts in the output disparity map. To reduce these artifacts, we apply a post-processing step. We use a weighted median filter to smooth the filled regions. (Note that pixels that have not been invalidated by the left/right check are not affected by this operation.) Ideally, we would like to use the guided filter weights defined in eq. (5). However, making these weights explicit is computationally intractable. Hence we choose the bilateral filter weights [18]:

$$W_{i,j}^{BL} = \frac{1}{K_i} \exp(-\frac{|i-j|^2}{\sigma_s^2}) \exp(-\frac{|I_i - I_j|^2}{\sigma_c^2}) \tag{10}$$

where K_i is a normalization factor and σ_s, σ_c are constant parameters which adjust the spatial and color dissimilarity, respectively.

3. EXPERIMENTAL RESULTS

Our real-time stereo algorithm is implemented and tested using an Intel Core 2 Quad, 2.4 GHZ PC. The GPU is a GeForce GTX480 graphics card with 1.5GB of memory from NVIDIA. We used the CUDA technique of NVIDIA Corporation [19] for implementation on the GPU. Throughout our test runs, the algorithm's parameters were set to constant values: $\{r, \epsilon, \alpha, \sigma_s, \sigma_c, T_c, T_g\}$ = {9, 0.0001, 0.9, 9, 0.1, 0.028, 0.008}. The window size of the bilateral median filter of eq.

[7] In practice, we record the disparity d_l of the closest consistent pixel to the left of the inconsistent pixel. We also record the disparity d_r of the closest consistent pixel to the right. The inconsistent pixel is then assigned to disparity $\min(d_l, d_r)$.

Image	Resolution	Disparities	With LR Check		Without LR Check	
			Time(sec.)	FPS	Time(sec.)	FPS
Tsukuba	384 x 288	16	0.019	52	0.010	100
Venus	434 x 383	20	0.038	26	0.019	52
Teddy	450 x 375	60	0.098	11	0.048	21
Cones	450 x 375	60	0.098	11	0.048	21

Table 1. Speed of our algorithm. LR check means left-right consistency checking.

Algorithm	Rank	Avg. Error (%)	Avg. Runtime (ms)
Ours	**14**	**5.55**	**65**
GeoSup [6]	18	5.80	16000
Ours using AdaptWeight [1]	20	5.86	15000
AdaptWeight [1]	40	6.67	8550
DCBGrid [15]	84	10.9	95.4*

Table 2. Rankings and run times for selected local stereo methods. Run times in the table are averaged over the four Middlebury test images. *The run time in [15] was reported before left-right consistency check in the corresponding paper. Hence, for fairness, we have multiplied the reported time by a factor of 2.

Algorithm	Rank	Avg. Error [%]	Error non-occluded pixels [%]				MDE/s
			Tsukuba	Venus	Teddy	Cones	
Ours	**14**	**5.55**	**1.51**	**0.20**	**6.16**	**2.71**	**199.68**
PlaneFitBP	17	5.78	0.97	0.17	6.65	4.17	9.43
RealTimeABW	49	7.9	1.26	0.33	10.7	4.81	4.42
RealtimeBFV	53	7.65	1.71	0.55	9.90	6.66	100.85
RealtimeBP	54	7.69	1.49	0.77	8.72	4.61	20.89
FastAggreg	59	8.24	1.16	4.03	9.04	5.37	20.25
RealtimeVar	68	9.05	3.33	1.15	6.18	4.66	3.80
RTCensus	73	9.73	5.08	1.58	7.96	4.10	1152*
RealTimeGPU	74	9.82	2.05	1.92	7.23	6.41	53.43
RealiabilityDP	77	10.7	1.36	2.35	9.82	12.9	42.11
DCBGrid	84	10.9	5.90	1.35	10.5	5.34	4.71

Table 3. Rankings for real time algorithms in the Middlebury on-line database. Currently, our algorithm is the bestperforming real time algorithm.

(10) is set to 19 × 19 pixels. These parameters have been found empirically.

To evaluate our approach, we performed two experiments. The first experiment is conducted on the images from the Middlebury benchmark [5]. We show the qualitative results in figure 3 and the quantitative results in tables 1, 2 and 3. The second experiment is carried out on a dynamic scene in order to assess the performance of our method when used with live scenarios.

Figure 3 shows the disparity maps computed for the test images and the corresponding error maps. From figure 3 we can see that our approach generates high quality results. Our method shows good performance in regions of poor texture and close to objects boundaries. According to the Middlebury ranking system, our approach takes rank 14 out of more than 100 submissions. Even more importantly, we are the best performing local stereo method in the on-line table. The algorithm even outperforms the original implementation of the adaptive support weight approach of [1] (rank 40).

To investigate why our method performs better than [1], we used their weights in our algorithm. For fair comparison, we used the same matching costs and occlusion handling as in our approach. We tuned the parameters of that approach to yield the best possible results. We found that the approach of [1] is about 230 times slower than ours, but ranks closely behind it (rank 20). From this test we deduce that the guided filter gives slightly better results for stereo matching than the bilateral filter. However, more importantly, our algorithm runs in real-time. Table 1 plots the run time of our algorithm. It takes approximately 10ms seconds to generate the disparity map for the Tsukuba image pair (100 fps) and 48ms seconds for the Teddy and Cones image pairs (21 fps). In table 2 we show the average run time of our method in comparison to other local stereo algorithms. We report run times of the full method including left-right consistency checking and post-processing. The run time for [1] is taken from [15]. Although these times were measured on different platforms, they still give a reasonable indication of the computational complexity. Table 3 shows the ranking of our algorithm compared to other real-time algorithms in the Middlebury on-line ranking table. The last column in the table shows the MDE/s computed for all algorithms.[8] From this table we can observe that there is only one method (RTCensus [20]) which has MDE/s value larger than ours (hence, is faster than our method). However, there is a large gap in the rank between this method and our algorithm. Nevertheless, our method is currently the best performing real-time method in the Middlebury table, showing an excellent trade-off between speed and quality of results.

The second experiment is done by capturing live videos using a bumblebee camera manufactured by Point Grey Research. Our algorithm runs at 17 fps when handling stereo images of resolution 640 × 480 pixels and 40 disparity levels (excluding the overhead for rendering and rectification). The disparity maps generated for two frames captured by our live system are shown in figure 4. From this figure one can notice that our approach is able to produce detailed and accurate disparity maps along with clean object boundaries. Note that there is only little flickering in the disparity sequences (see video in the supplementary material). This is an indicator for the high robustness of our method.

4. CONCLUSIONS

This paper has proposed a high-quality real-time approach for stereo matching. The key idea is to filter the cost volume with an edge-preserving filter that can be implemented in a very fast way, i.e. the guided filter [2]. This has led to a new adaptive support weight approach whose run time in the aggregation step is independent of the match window size. According to the Middlebury results, our approach is the top performer

[8] MDE/s (Million Disparity Estimations per second) = width * height * disparities * fps.

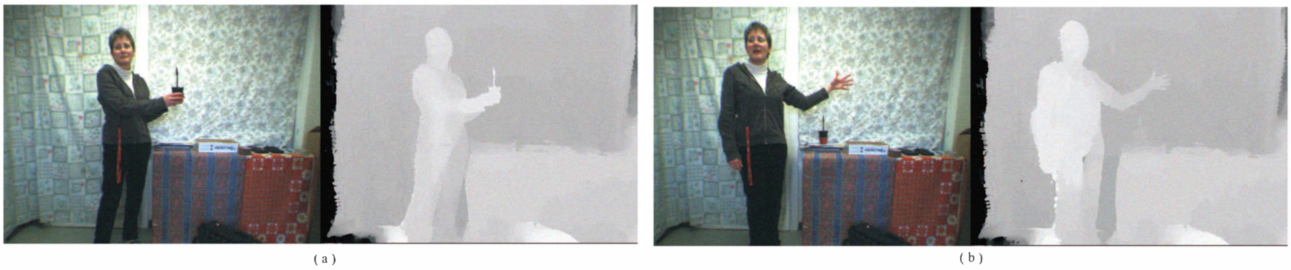


Fig. 4. Two sample frames captured by our live system and their corresponding disparity maps. (a) Shot 1 (b) Shot 2.

among stereo methods that rely on local optimization and also among competing real-time algorithms. The key argument for our method is that it achieves real-time frame rates, while keeping the quality of results at a high level.

5. REFERENCES

[1] K.J. Yoon and I.S. Kweon, “Locally adaptive support-weight approach for visual correspondence search,” in *CVPR*, 2005.

[2] K. He, J. Sun, and X. Tang, “Guided image filtering,” in *ECCV*, 2010.

[3] M. Agrawal and L.S. Davis, “Window-based, discontinuity preserving stereo,” in *CVPR*, 2004.

[4] A. Fusiello, V. Roberto, and E. Trucco, “Efficient stereo with multiple windowing,” *CVPR*, 1997.

[5] D. Scharstein and R. Szeliski, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,” *IJCV*, vol. 47, no. 1/2/3, pp. 7–42, 2002. http://www.middlebury.edu/stereo/.

[6] A. Hosni, M. Bleyer, M. Gelautz, and C. Rhemann, “Local stereo matching using geodesic support weights,” in *ICIP*, 2009.

[7] D. Min and K. Sohn, “Cost aggregation and occlusion handling with WLS in stereo matching,” *IEEE Tansactions on Image Processing*, vol. 17, no. 8, pp. 1431 – 1442, 2008.

[8] F. Tombari, S. Mattoccia, and L. Di Stefano, “Segmentation-based adaptive support for accurate stereo correspondence,” in *PSIVT*, 2007.

[9] M. Gong, R. Yang, L. Wang, and M. Gong, “A performance study on different cost aggregation approaches used in real-time stereo matching,” *IJCV*, vol. 75, no. 2, pp. 283 – 296, 2007.

[10] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda, “Classification and evaluation of cost aggregation methods for stereo correspondence,” in *CVPR*, 2008.

[11] O. Faugeras, B. Hotz, H. Mathieu, T. Viéville, Z. Zhang, P. Fua, E. Théron, L. Moll, G. Berry, J. Vuillemin, P. Bertin, and C. Proy, “Real time correlation based stereo: algorithm implementations and applications,” Tech. Rep., RR-2013, INRIA, 1996.

[12] K. Mühlmann, D. Maier, J. Hesser, and R. Männer, “Calculating dense disparity maps from color stereo images, an efficient implementation,” *IJCV*, vol. 47, no. 1 - 3, pp. 79 – 88, 2002.

[13] M. Gerrits and P. Bekaert, “Local stereo matching with segmentation-based outlier rejection,” in *CRV*, 2006.

[14] A. Hosni, M. Bleyer, and Margrit Gelautz, “Near real-time stereo with adaptive support weight approaches,” in *3DPVT*, 2010.

[15] C. Richardt, D. Orr, I. Davies, A. Criminisi, and N. Dodgson, “Real-time spatiotemporal stereo matching using the dual-cross-bilateral grid,” in *ECCV*, 2010.

[16] A. Bruhn and J.Weickert., “Optical flow estimation on coarse-to-fine region-trees using discrete optimization,” in *ICCV*, 2005.

[17] T. Brox and J. Malik., “Large displacement optical flow: descriptor matching in variational motion estimation,” in *PAMI*, 2010.

[18] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand, “A gentle introduction to bilateral filtering and its applications,” *International Conference on Computer Graphics and Interactive Techniques*, 2008.

[19] “CUDA: Compute Unified Device Architecture programming guide,” Tech. Rep., Nvidia Corporation, 2008.

[20] M. Humenberger, C. Zinner, M. Weber, W. Kubinger, and M. Vincze, “A fast stereo matching algorithm suitable for embedded real-time systems,” *Computer Vision and Image Understanding*, vol. 114, pp. 1180 – 1202, 2010.