Near Real-Time Stereo With Adaptive Support Weight Approaches

Asmaa Hosni, Michael Bleyer and Margrit Gelautz
Institute for Software Technology and Interactive Systems, Vienna University of Technology
Favoritenstr. 9-11/188/2, A-1040 Vienna, Austria
asmaa,bleyer,gelautz@ims.tuwien.ac.at

Abstract

Algorithms based on the adaptive support weight strategy currently represent the state-of-the-art in local stereo matching. Unfortunately, their good-quality results come at the price of high computation times: As opposed to standard local algorithms, incremental computation via sliding windows is not applicable for adaptive support weight windows. This paper presents a method for considerably speeding up computation times of these methods. The key idea is to exploit the adaptive support weight windows for generating an explicit over-segmentation of the reference image in a fast way. Having this explicit segmentation, we can take advantage of a modified "segmentation-based" sliding window technique, which makes run time independent of the window size. In particular, we demonstrate our transformation scheme for the geodesic stereo matcher of [11] that has recently produced excellent results. Our unoptimized GPU-based implementation processes 320 $\times$ 240 pixel images with 26 allowed disparities at 10 frames per second and achieves rank 32 out of 74 methods in the Middlebury online benchmark.

1. Introduction

Stereo research is commonly divided into two different branches [15], *i.e.* *local* and *global* methods.

Global methods express their model of the stereo problem as an energy function. This energy is then subject to optimization, which can, for example, be accomplished via dynamic programming, graph-cuts or message passing. Global methods are known to produce good-quality results, but lose attractiveness due to the high computational demands of the energy optimization step. For several stereo applications such as robotics, teleconferencing, virtual reality or security applications, slow run time is a knock-out criterion. It is therefore not surprising that virtually all commercial stereo products (*e.g.* Point Grey, Videre Design or TYZX) do not follow the global, but the local approach.

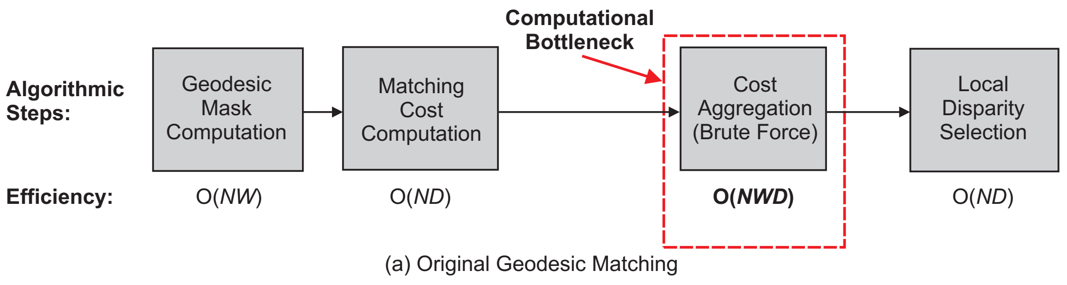
Local approaches rely on support windows. For each pixel of the left image, they center a (typically square) window, which is then shifted in the right image to determine the matching point, *i.e.* the point of highest correspondence. In a naive implementation, this matching procedure is not necessarily a fast one. However, it can effectively be speeded up by elimination of redundant computations via a *sliding window technique* [5, 14]. The sliding window technique makes run time independent of the window size, which even enables real-time stereo systems.

The inherent problem is the difficulty of choosing "good" windows. There are two contradicting requirements that a window should fullfill: (1) It should be large to capture enough intensity/color variation for handling regions of poor texture. (2) It should be small to avoid overlapping disparity boundaries (edge fattening effect). Finding good trade-offs between these requirements has dominated research on local methods for decades (*e.g.* [6, 9, 12]), but has not overcome this problem. Therefore, the results of local methods have traditionally been considerably inferior in comparison to global methods. This has changed recently.

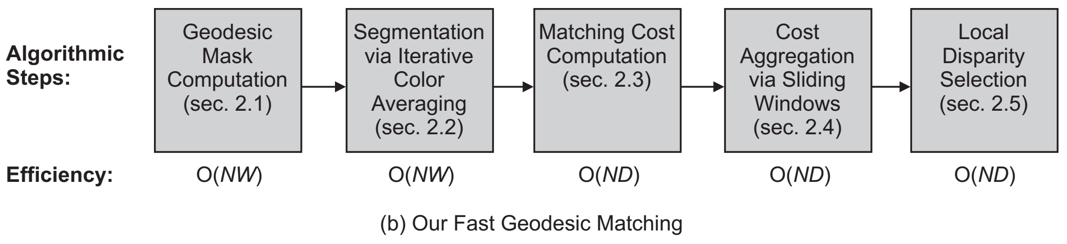
Exploitation of the concept of self similarity has represented an important progress in the field of local stereo matching. The simple idea is that if neighboring or close-by pixels show similar appearance (typically in their colors), they are likely to have the same disparity value. Hence, in a local approach, the support region for a specific pixel p should be formed by pixels that are close to p in terms of colors and spatial positions. The adaptive support weight approach of [19] has been the first local method to implement this concept.[1] Numerous variations of [19] have appeared since then (*e.g.* [7, 11, 13, 17]). A detailed discussion and evaluation of such methods is found in [8, 18].

The main reason why [19] and related approaches have become popular in the stereo community is their good-quality results, *i.e.* the performance is very close to that of global methods. Unfortunately, this advantage does not come "for free". It is known that the sliding window

[1]It is interesting that the importance of self similar pixels has been realized considerably earlier in global stereo matching [16] and has led to a long track of publications on segmentation-based stereo (*e.g.* [2, 10]).



Computational Bottleneck
Algorithmic Steps:
Geodesic Mask Computation
Matching Cost Computation
Cost Aggregation (Brute Force)
Local Disparity Selection
Efficiency:
O(NW)
O(ND)
O(NWD)
O(ND)
(a) Original Geodesic Matching



Algorithmic Steps:
Geodesic Mask Computation (sec. 2.1)
Segmentation via Iterative Color Averaging (sec. 2.2)
Matching Cost Computation (sec. 2.3)
Cost Aggregation via Sliding Windows (sec. 2.4)
Local Disparity Selection (sec. 2.5)
Efficiency:
O(NW)
O(NW)
O(ND)
O(ND)
O(ND)
(b) Our Fast Geodesic Matching

Figure 1. Fast versus original geodesic stereo matching. We show the individual steps of the algorithms via block diagrams and plot corresponding complexity estimations. (Here, N denotes the number of pixels, W the number of pixels within the match window and D the number of allowed disparities.) (a) The original geodesic algorithm. The cost aggregation step represents a computational bottleneck and cannot be speeded up by the sliding window technique. (b) Our fast geodesic algorithm. We eliminate the computational bottleneck by using the geodesic masks to generate an explicit color segmentation. This explicit segmentation enables application of "segmentation-based" sliding windows in the cost aggregation step, which makes run time independent of the window size. Our fast approximation of the original geodesic algorithm does not lead to considerable performance degradations in terms of quality of results.

method cannot be applied to adaptive support weight windows. Therefore, cost aggregation has to be performed in a "brute-force" manner so that the run time is directly dependent on the size of the match window. This is particularly bad considering that windows should be large to get good-quality results (33×33 pixels in [19]). Consequently, [19] has reported run times that easily exceed a minute. This is very crucial, since it cancels out the biggest advantage over global methods, *i.e*. fast computation time.

In this work, we present a simple technique for transforming adaptive support weight approaches into *fast* stereo matchers. In particular, we approximate the geodesic support weight approach of [11]. (Note that our method would also work for [19], but we have decided for the geodesic strategy, because it has produced higher-quality results in our experiments.) The main idea is to use the adaptive support weight windows for generating an explicit over-segmentation, *i.e*. we divide the reference image into disjoint regions of homogeneous color. Note that our segmentation method is fast and hence avoids that the segmentation step becomes the new computational bottleneck in our method. The advantage of an explicit segmentation is that it enables a modified "segmentation-based" sliding window technique [7]. Thus our stereo method is no longer dependent on the size of the match window as in [11, 13, 17, 19]. Using this transformation scheme, we can considerably improve the computational performance of [11], while keeping approximately the same level of quality. Our method is very close to real-time performance.

Figure 1 shows the pipelines of the original geodesic approach [11] and our "transformed" algorithm. The major difference is that we remove the computational bottleneck (*Cost Aggregation (Brute Force)*) of figure 1a and replace it by two low-cost steps (*Segmentation via Iterative Color Averaging* and *Cost Aggregation via Sliding Windows*) in figure 1b.

2. Algorithm

In the following, we go through the individual steps visualized in the block diagram of figure 1b.

2.1. Geodesic Mask Computation

In the adaptive support weight approach, a square window $\mathcal{W}_c$ of predefined size is centered at each pixel c of the reference image. For each pixel $p \in \mathcal{W}_c$, a function $w(p, c)$ computes a weight representing the likelihood to which pixel p lies on the same surface with pixel c. We refer to the set of values $\{w(p, c) : p \in \mathcal{W}_c\}$ as the mask of c. (Since in our case, masks are determined using the geodesic approach of [11], we will also use the expression geodesic mask.) Note that in [7, 11, 13, 17, 19] such a mask is exploited to regulate the influence of each pixel $p \in \mathcal{W}_c$ in the aggregation process, *e.g*. if $w(p, c) = 0$,

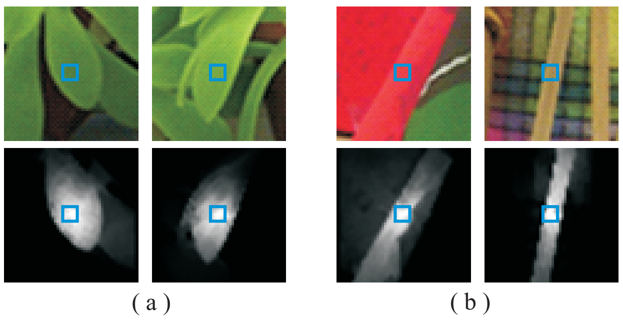


Figure 2. Example masks computed using the geodesic support weight approach of [11]. The blue rectangle marks the center pixel for which the geodesic mask (bottom images) is computed. In the mask images, bright pixels represent high-weight pixels that are likely to lie on the same surface with the center pixel. (a) Masks computed for the Teddy image of the Middlebury set. By using the concept of connectivity we can separate the center leaf from the other leaves that have very similar colors and spatial positions, but lie on a different depth surface. (b) Geodesic masks computed for the Cones set.

pixel p is not considered to be part of c's support region. As will be discussed in section 2.2, this paper proposes a different approach, *i.e.* we use the masks for generating an explicit over-segmentation of the reference image.

Let us now focus on the most challenging question, namely how to implement the function $w(p,c)$. This is where the concept of self similarity comes into play, *i.e.* two pixels are likely to reside on the same surface, if they are similar in their appearance. Adaptive weight approaches basically only differ in the way how they express this concept. In [19], $w(p,c)$ is computed as the pixels' difference in spatial positions and color values. [11] is more advanced in this respect, since the authors implement an additional cue for self similarity, namely connectivity. This means that $w(p,c)$ only returns high values, if there is a path connecting p with c along which the color does not vary considerably.

Let us explain the advantage of this connectivity property using the example masks of figure 2. In figure 2a, there are several leaves that all have similar colors and similar spatial positions. When estimating the support region of the center leaf, the approach of [19] would therefore erroneously also include pixels of other leaves although they lie on a different depth surface. In contrast to this, [11] can avoid this wrong segmentation: Every path connecting two pixels of different leaves involves a color edge where the color will vary considerably. Hence, these pixels do not fulfill the connectivity property.

Let us now formulate the function $w(p,c)$ according to the geodesic approach [11]. The weight of p is inversely proportional to its geodesic distance to the center pixel c, which we implement by

$$w(p,c) = \exp\left(-\frac{D(p,c)}{\gamma}\right) \tag{1}$$

where γ is a user-defined parameter that defines the strength of the resulting segmentation. $D(p,c)$ denotes the geodesic distance defined as the costs of the cheapest path between p and c:

$$D(p,c) = \min_{P \in \mathcal{P}_{p,c}} \rho(P). \tag{2}$$

Here, $\mathcal{P}_{p,c}$ denotes the set of all paths between p and c. A path $P = \{p_1, p_2, \cdots, p_n\} \in \mathcal{P}_{p,c}$ represents a sequence of spatially neighboring points (in 8-connectivity) so that $p_1 = p$ and $p_n = c$. We define the costs of this path as

$$\rho(P) = \sum_{i=2}^{i=n} \sigma(p_i, p_{i-1}) \tag{3}$$

with $\sigma()$ being a function that determines the color difference. This function is implemented by

$$\sigma(p,q) = \sqrt{\sum_{i=1}^{i=3} \left(f_i(p) - f_i(q)\right)^2}. \tag{4}$$

Here, $f_i(p)$ denotes the value of the ith color channel at pixel p. In our implementation, we represent color using the RGB system.

Evaluation of equation (2) can easily become computationally expensive if done in an exact way. However, in our method, we only determine an approximation of the geodesic distances via Borgefors' algorithm [3]. This approximation algorithm is computationally very efficient, *i.e.* the computational complexity for computing *all* geodesic masks for the reference image is $O(NW)$ where N is the number of the reference image's pixels and W denotes the number of pixels inside a window.

It is important to understand that the computational bottleneck in [11] stems from applying the precomputed geodesic masks at each pixel and *disparity* in the aggregation step, which leads to a complexity of $O(NWD)$ with D being the number of allowed disparities. In this paper, we eliminate this computational bottleneck by using the precomputed geodesic masks for generating an explicit over-segmentation, which is discussed next.

2.2. Segmentation via Iterative Color Averaging

In the following, we exploit the geodesic masks of section 2.1 to compute an explicit color over-segmentation in a fast way. It is important to note that computational efficiency of the segmentation step is vital for the transformation scheme proposed in this paper. If one applies a computational demanding segmentation algorithm such as *e.g.* the

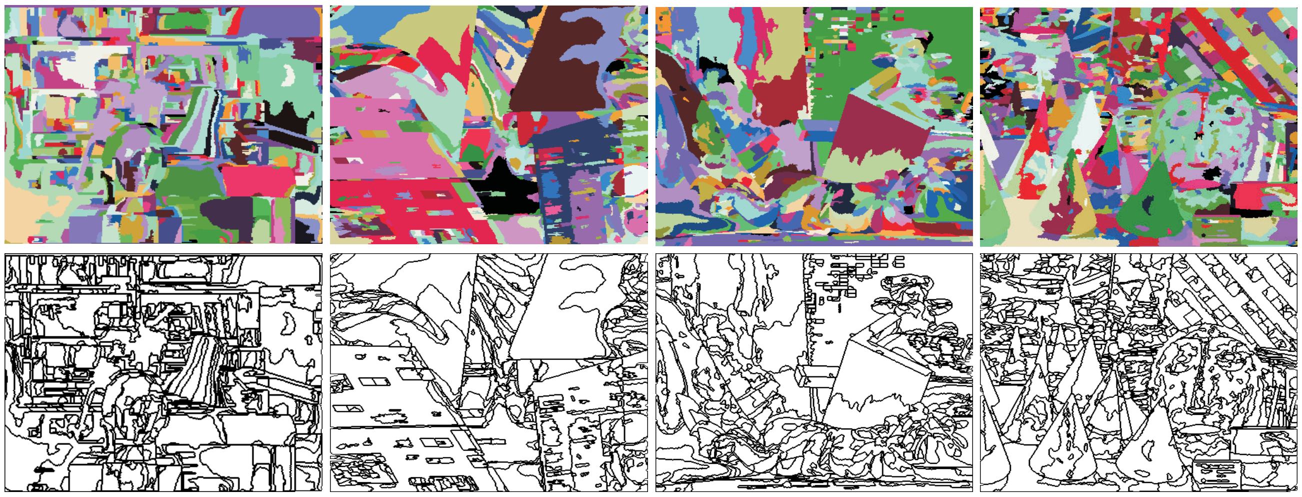


Figure 3. Our segmentation results for the four images currently used in the Middlebury benchmark. The top row shows the extracted color segments by assigning all pixels of the same segment to the same color. The bottom row shows the same results, but uses a different representation. Here, we show the borders of segments.

commonly-used meanshift algorithm [4], the segmentation step can easily become the new computational bottleneck. Hence, the overall computational performance of the stereo matching algorithm might not necessarily be faster than that of the "untransformed" algorithm.

Our fast segmentation method works as follows. We take the reference color image as an input and iteratively apply filtering on it. At each pixel c of the reference image, our filter replaces c's color value with the mean color computed over all pixels inside a small window centered on c. The idea is that if this filter is applied a few times, then pixels belonging to the same color surface will have almost identical color values in the filtered image. The crucial point is that this segmentation strategy can only work if we compute the mean color solely over pixels that belong to the same color surface. Otherwise, artifacts at segment discontinuities, where colors of different surfaces would be mixed, are inevitable. Note that our geodesic masks hold the information required for this step, *i.e.* for a pixel c, its geodesic mask provides a set of close-by pixels that are likely to lie on the same surface with c.[2]

We now describe this filtering procedure in a more formal way. Let F and F' be the color values of the original reference image and the filtered image, respectively. F' is computed from F by applying the following operation on each pixel c of the left image:

$$f'_i(c) = \frac{\sum_{p \in \mathcal{W}_c} w(p,c) \cdot f_i(p)}{\sum_{p \in \mathcal{W}_c} w(p,c)} \qquad (5)$$

with $f()$ and $f'()$ denoting color values in F and F'.

[2]A segmentation algorithm that applies color averaging has also been proposed in [20]. In contrast to our more sophisticated large geodesic filter masks, [20] uses a set of 8 predefined filters in a 3×3 pixel window.

As stated above, we iterate this procedure, *i.e.* after computing F' we compute F'' using equation (5) and so on. In our implementation, we use three iterations. The result of the iterative filtering is an image of approximately piecewise constant colors. We interpret groups of neighboring pixels that share the same color as our final segments. In a postprocessing step, we reduce the number of segments by deleting segments of very small size. A small segment is thereby merged with its most similar neighboring segment, *i.e.* the one that has the most similar color.

Note that we can regulate the strength of the segmentation via the setting of parameter γ in equation (1). In general, it is advisable to apply a strong over-segmentation to avoid situations in which a color segment overlaps a disparity discontinuity. Figure 3 shows over-segmentations produced by our fast segmentation algorithm on the Middlebury images. It can be observed that our algorithm works well in preserving the object edges in all four images.

2.3. Matching Cost Computation

Let us now compute the matching costs. Given a pixel p of the left and a pixel q of the right image, the matching costs determine the likelihood to which p and q correspond to each other. To compute the matching costs, we use the absolute difference of color values (in RGB space) as a dissimilarity function. Formally, the function $m(p,d)$ computes the costs for matching pixel p at disparity d by

$$m(p,d) = \sum_{i=1}^{i=3} |f_{i,left}(p) - f_{i,right}(p-d)|\,. \qquad (6)$$

In the following, we will aggregate these matching costs.

2.4. Cost Aggregation via Sliding Windows

The explicit color segmentation computed in section 2.2 enables the use of a segmentation-based sliding window technique [7] in the aggregation step. Note that a key argument for our transformation scheme is that by bringing adaptive support weight approaches into a form where sliding windows are applicable, we can reduce the computational complexity of the aggregation step from $O(NWD)$ to $O(ND)$. Hence, we can eliminate the computational bottleneck.

We start by formulating the aggregation function. The idea is that the support region of a pixel c is solely formed by pixels that lie in c's color segment. Let $\mathcal{S}_c$ denote all pixels that reside in the same segment as c itself and $\mathcal{W}'_c$ be a square window centered at c. We build c's support region $\mathcal{L}_c$ by intersection:

$$\mathcal{L}_c = \mathcal{S}_c \cap \mathcal{W}'_c. \tag{7}$$

Knowing the support region $\mathcal{L}_c$, we can compute the aggregated matching scores $a(c, d)$ of pixel c at disparity d by

$$a(c, d) = \sum_{p \in \mathcal{L}_c} m(p, d). \tag{8}$$

Let us now focus on the task of computing $a(c, d)$ for each pixel c and each allowed disparity d. The main advantage of the aggregation function in equation (8) is that there is significant computational redundancy when calculating $a()$ for spatially neighboring pixels. More precisely, let pixels p and p' be spatial neighbors lying in the same segment. If one writes a list of values that need to be summed-up in order to compute $a(p, d)$ and another list for computing $a(p', d)$, both list will be identical with a few exceptions.[3] This motivates an incremental computation scheme, *i.e.* we can start with the value of $a(p, d)$ and focus on the few exceptions to compute $a(p', d)$.

The idea of removing these redundant calculations to make the computational complexity for calculating $a()$ independent of the window size is implemented in the segmentation-based sliding window method of [7]. This algorithm is listed in figure 4 using pseudo-code. Here, the basic difference to the standard sliding window approach is that instead of just having a single "running sum", an individual "running sum" T_s is maintained for each segment s of the reference image.[4]

[3]Note that this is not valid for "untransformed" adaptive support weight approaches where the matching scores are multiplied with a support mask that is different for each pixel.

[4]In fact, the standard sliding window algorithm can be considered as a special case of the segmentation-based algorithm where there is just one segment that contains all pixels of the reference view.

Segmentation-based sliding windows

```
1. For each row:
    (a) For each segment $s$: $T_s = 0$
    (b) For each pixel $x$ in row $y$:
        i.   $T_{s_{x+w/2,y}} = T_{s_{x+w/2,y}} + M_{x+w/2,y}$
        ii.  $T_{s_{x-w/2,y}} = T_{s_{x-w/2,y}} - M_{x-w/2,y}$
        iii. $A^*_{x,y} = T_{s_{x,y}}$
2. For each column:
    (a) For each segment $s$: $T_s = 0$
    (b) For each pixel $y$ in column $x$:
        i.   $T_{s_{x,y+w/2}} = T_{s_{x,y+w/2}} + A^*_{x,y+w/2}$
        ii.  $T_{s_{x,y-w/2}} = T_{s_{x,y-w/2}} - A^*_{x,y-w/2}$
        iii. $A_{x,y} = T_{s_{x,y}}$
```

Figure 4. The segmentation-based sliding window algorithm proposed in [7]. The algorithm computes the values of $a()$ in equation (8) for all pixels at a single, fixed disparity. M denotes the precomputed matching costs of equation (6) and w is the size of the aggregation window. A holds the aggregated costs that represent the algorithm's output.

2.5. Local Disparity Selection

We use the aggregated costs to determine a disparity value d_p for each pixel p of the reference view. For the sake of high computational efficiency, we do not incorporate global optimization algorithms, but use the local Winner-Takes-All strategy. Hence, the disparity is computed by

$$d_p = \operatorname*{argmin}_{d \in \mathcal{D}} a(p, d) \tag{9}$$

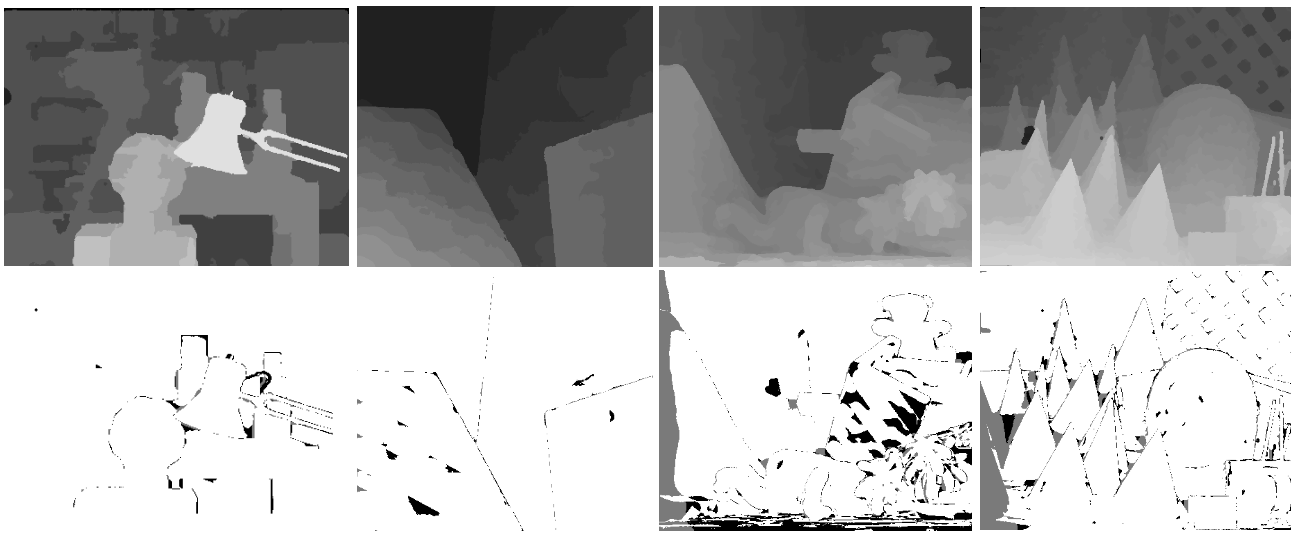
where $\mathcal{D}$ represents the set of all allowed disparities.

3. Experimental Results

We perform two tests to evaluate our method. The first test is conducted on the four test images from the Middlebury benchmark [15] and shall evaluate the quality and computational efficiency of our algorithm. The second test is carried out on a dynamic scene in order to assess the performance of our stereo matcher when used in real-time scenarios.

We run the proposed algorithm on an Intel Core 2 Quad Q6600 processor with 2.4 GHZ and use a GeForce GTS 250 graphics card with 1GB memory manufactured by NVIDIA. We apply CUDA [1] to implement our approach on the GPU.

In our test runs, the algorithm's parameters are set to constant values. The parameter γ, which is used in converting


(a)

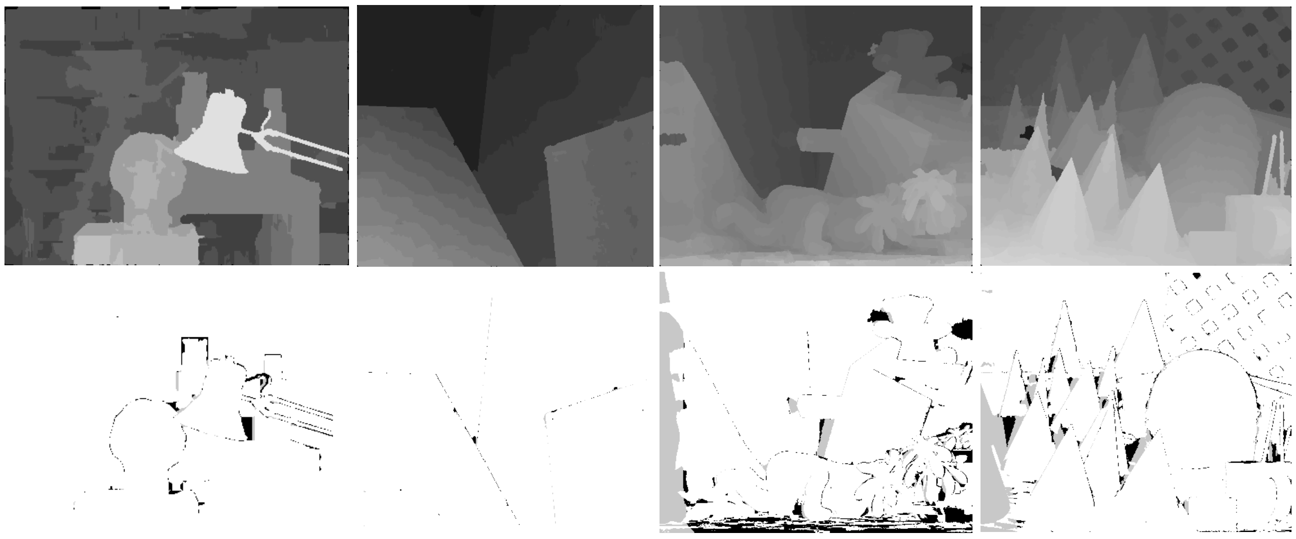

(b)

Figure 5. Results on Middlebury images generated using constant parameter settings. (a) The first row shows the results computed by our algorithm after applying left-right consistancy checking and occlusion filling. The second row shows a comparison against the ground truth by plotting disparity errors larger than one pixel. (b) Corresponding results obtained by [11].

geodesic distances to geodesic weights, is fixed to the value $\gamma = 10$. The window size $\mathcal{W}_c$ for image segmentation is chosen to be 9×9 pixels and adjusted to 31×31 pixels for $\mathcal{W}'_c$ in the cost aggregation process. These parameters have been found empirically.

In the first test, we use the Middlebury stereo benchmark [15] to quantitatively evaluate the matching accuracy and computational efficiency of our approach on a static scene. In this test, we incorporate a simple method for occlusion detection and filling, which is the same as that used in [11]. (Using the same occlusion handling method shall ensure a fair comparison between the results of the two algorithms.) Roughly spoken, occlusion detection works by left-right consistency checking. Pixels invalidated by this check are then assigned to the background disparity and finally a smoothing filter is applied on the invalidated regions. For more details on this occlusion handling procedure, the reader is referred to [11].

Figure 5 shows the results obtained by our algorithm (figure 5a) compared against the results obtained by the "untransformed" algorithm of [11] (figure 5b). From this figure we can see that our results exhibit a small degradation in

Algorithm	Rank	Avg. Error [%]	Error non-occluded pixels [%]			
			Tsukuba	Venus	Teddy	Cones
GeoSup [11]	10	5.80	1.45	0.14	6.88	2.94
NRTGeoSup	**32**	**6.55**	**1.52**	**1.05**	**9.21**	**3.06**

Table 1. Rankings of our proposed algorithm and the "untransformed" algorithm [11] in the Middlebury online database. Our proposed algorithm is denoted by *NRTGeoSup*.

quality compared to the results of [11]. For example, some higher deviations between our results and the ground truth can be recognized from the error maps of the Venus and Teddy images in columns 2 and 3 of figure 5. The problem is that our large 31×31 pixel windows are not ideal for handling slanted surfaces where the window contains pixels that lie on slightly different disparities. This problem is less pronounced for the untransformed algorithm, since it assigns lower continuous weights to pixels that have a large distance from the center pixel, whereas our weights are essentially binary ones. However, it is important to note that our algorithm performs well in the reconstruction of disparity borders, while it also finds correct disparities for regions of low texture.

Table 1 shows quantitative results that are taken from the Middlebury online table. Our algorithm currently takes the 32th rank out of 74 submissions, while the "untransformed" algorithm of [11] takes the 10th rank. Despite this obvious gap in the ranking, it is important to note that the average error is increased by less than 1 percent. This implies that the price we pay for the significant speed-up (as discussed below) is only a relatively small degradation in quality.

Table 2 compares the execution times of our algorithm and the "untransformed" method of [11]. To allow for a fair comparison, we have performed a GPU implementation of [11]. We also use the same window size in the matching cost aggregation step for both algorithms. The measured speed up is therefore solely due to our algorithmic modifications. The table shows the overall time consumed to compute the disparity maps for the four test images of the Middlebury benchmark (which includes left-right consistency checking and occlusion filling).

From Table 2 one can see that our algorithm has a speed up factor of 61 for the Tsukuba image and an approximate speed up factor of 26 for the Teddy and Cones images. The processing time for our method is 0.176 seconds for the Tsukuba images which have a resolution of 384×288 pixels and 16 allowed disparity values. Hence, our algorithm runs at 6 fps. For the Teddy and Cones images that have a resolution of 450×375 pixels and 60 allowed disparity values, our algorithm consumes a processing time of 0.704 and 0.688 seconds, respectively. Similar performance figures are obtained for the Venus images. These results demonstrate that our approach yields a very good trade-off between accuracy and speed.

Image	Resolution	Time(sec.)		Speed-up Factor
		GeoSup [11]	**Our Alg.**	
Tsukuba	384 x 288	10.874	**0.176**	**61**
Venus	434 x 383	15.97	**0.406**	**39**
Teddy	450 x 375	18.532	**0.704**	**26**
Cones	450 x 375	18.56	**0.688**	**26**

Table 2. Computational efficiency of our algorithm compared to the "untransformed" geodesic stereo matcher [11] (denoted by *GeoSup*).

Our algorithm is also tested on live videos captured using a bumblebee camera manufactured by Point Grey Research. We found that our algorithm can achieve 10 fps when handling stereo images of 320×240 pixels and 26 disparity levels (excluding the overhead for rectification and rendering). This is equivalent to 19.97 million disparity estimations per second (MDE/s). The disparity maps generated for two frames captured by our live system are shown in figure 6. As opposed to the tests on static scenes, we do not apply any occlusion detection or filling mechanism in order to achieve maximum computational speed. Although we do not perform occlusion handling, we found that our approach is still able to produce detailed and accurate disparity maps along with clean object boundaries.

4. Conclusions

This paper has proposed a method for considerably improving the computational efficiency of adaptive support weight algorithms. We argue that standard support weight approaches are computationally slow, because they use support masks directly in the aggregation step. In contrast to this, we modify the stereo pipeline so that our aggregation step applies the support masks only in an implicit way. We add an additional step to the pipeline where a fast segmentation is computed via the support masks. The major advantage of this explicit segmentation is that it enables application of a fast segmentation-based sliding window technique in the aggregation step. This makes run time independent of the window size and hence overcomes the computational bottleneck of standard algorithms.

In the experimental results, we have shown that in comparison to an "untransformed" adaptive support weight algorithm, our method leads to a slight decrease in quality of disparity maps. However, the important point is that our method is 20-60 times faster.

In future work, we will concentrate on improving our GPU-based implementation. Currently, we only have a preliminary implementation that does not fully exploit the capabilities of modern graphics cards. We strongly believe that real-time performance can be accomplished by tuning our implementation (or even by using a more recent graphics card).

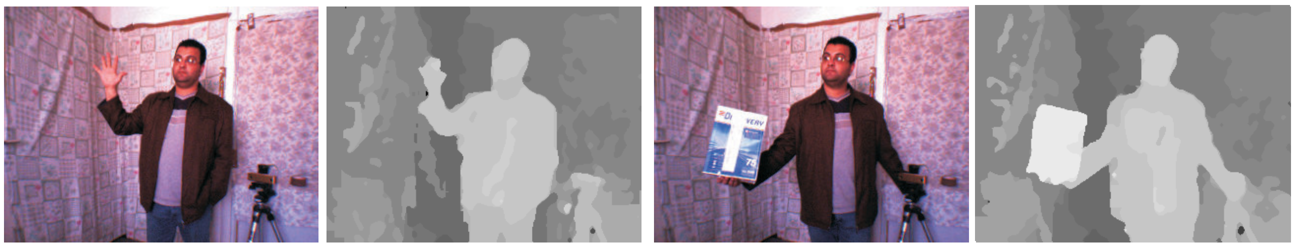


Figure 6. Two sample frames captured by our live system and their corresponding disparity maps. (a) Shot 1: (i) left image, (ii) disparity map. (b) Shot 2: (i) left image, (ii) disparity map.

Acknowledgements

Asmaa Hosni is supported by Vienna PhD School of Informatics. Michael Bleyer obtained financial support from the Austrian Science Fund (FWF) under project P19797 and the Vienna Science and Technology Fund (WWTF) under project ICT08-019.

References

[1] CUDA: Compute Unified Device Architecture programming guide. Technical report, Nvidia Corporation, 2008.

[2] M. Bleyer and M. Gelautz. A layered stereo matching algorithm using image segmentation and global visibility constraints. *ISPRS Journal*, 59(3):128–150, 2005.

[3] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34(3):344–371, 1986.

[4] C. Christoudias, B. Georgescu, and P. Meer. Synergism in low-level vision. In *International Conference on Pattern Recognition*, volume 4, pages 150–155, 2002.

[5] 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. Technical report, RR-2013, INRIA, 1996.

[6] A. Fusiello, V. Roberto, and E. Trucco. Efficient stereo with multiple windowing. In *CVPR*, pages 858–863, 1997.

[7] M. Gerrits and P. Bekaert. Local stereo matching with segmentation-based outlier rejection. In *CRV*, 2006.

[8] M. Gong, R. Yang, L. Wang, and M. Gong. A performance study on different cost aggregation approaches used in real-time stereo matching. *IJCV*, 75(2):283–296, 2007.

[9] H. Hirschmüller, P. Innocent, and J. Garibaldi. Real-time correlation-based stereo vision with reduced border errors. *IJCV*, 47:229–246, 2002.

[10] L. Hong and G. Chen. Segment-based stereo matching using graph cuts. In *CVPR*, volume 1, pages 74–81, 2004.

[11] A. Hosni, M. Bleyer, M. Gelautz, and C. Rhemann. Local stereo matching using geodesic support weights. In *ICIP*, 2009.

[12] T. Kanade and M. Okutomi. A stereo matching algorithm with an adaptive window: Theory and experiment. *PAMI*, 16(9):920–932, 1994.

[13] D. Min and K. Sohn. Cost aggregation and occlusion handling with WLS in stereo matching. *TIP*, 17(8):1431–1442, 2008.

[14] K. Mühlmann, D. Maier, J. Hesser, and R. Männer. Calculating dense disparity maps from color stereo images, an efficient implementation. *IJCV*, 47(1):79–88, 2002.

[15] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 47(1/2/3):7–42, 2002. http://www.middlebury.edu/stereo/.

[16] H. Tao and H. Sawhney. Global matching criterion and color segmentation based stereo. In *Workshop on the Application of Computer Vision*, pages 246–253, 2000.

[17] F. Tombari, S. Mattoccia, and L. D. Stefano. Segmentation-based adaptive support for accurate stereo correspondence. In *PSIVT*, pages 427–438, 2007.

[18] F. Tombari, S. Mattoccia, L. D. Stefano, and E. Addimanda. Classification and evaluation of cost aggregation methods for stereo correspondence. In *CVPR*, pages 1–8, 2008.

[19] K. Yoon and I. Kweon. Locally adaptive support-weight approach for visual correspondence search. In *CVPR*, volume 2, pages 924–931, 2005.

[20] L. Zitnick, S. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Transaction on Graphics*, 23(3):600–608, 2004.