PatchMatch Stereo - Stereo Matching with Slanted Support Windows

Michael Bleyer[1]
bleyer@ims.tuwien.ac.at
Christoph Rhemann[1]
rhemann@ims.tuwien.ac.at
Carsten Rother[2]
carrot@microsoft.com

[1] Institute of Software Technology
Vienna University of Technology
Vienna, Austria

[2] Microsoft Research Cambridge
Cambridge, UK

Abstract

Common local stereo methods match support windows at integer-valued disparities. The implicit assumption that pixels within the support region have constant disparity does not hold for slanted surfaces and leads to a bias towards reconstructing fronto-parallel surfaces. This work overcomes this bias by estimating an individual 3D plane at each pixel onto which the support region is projected. The major challenge of this approach is to find a pixel's optimal 3D plane among all possible planes whose number is infinite. We show that an ideal algorithm to solve this problem is PatchMatch [1] that we extend to find an approximate nearest neighbor according to a plane. In addition to PatchMatch's spatial propagation scheme, we propose (1) view propagation where planes are propagated among left and right views of the stereo pair and (2) temporal propagation where planes are propagated from preceding and consecutive frames of a video when doing temporal stereo. Adaptive support weights are used in matching cost aggregation to improve results at disparity borders. We also show that our slanted support windows can be used to compute a cost volume for global stereo methods, which allows for explicit treatment of occlusions and can handle large untextured regions. In the results we demonstrate that our method reconstructs highly slanted surfaces and achieves impressive disparity details with sub-pixel precision. In the Middlebury table, our method is currently top-performer among local methods and takes rank 2 among approximately 110 competitors if sub-pixel precision is considered.

1 Introduction

In local stereo matching, a support window is centered on a pixel of the reference frame. This support window is then displaced in the second image to find the point of lowest color dissimilarity, which represents the matching point. There is an implicit assumption in this procedure, i.e., all pixels within the support window have constant disparity. In practice, this assumption is unlikely to hold for two reasons. (1) The support window contains pixels that lie on a different surface than the center pixel. (2) The window captures a surface that is slanted, i.e., not fronto-parallel. There has recently been a large number of papers that address problem (1), and it seems that the adaptive support weight strategy is the best solution to this problem [7, 9, 10, 16]. In contrast, less attention has been paid to problem (2), which we tackle.

Figure 1a illustrates the problem. Standard local algorithms [7, 9, 10, 16] apply fronto-parallel windows at discrete disparities to reconstruct the four points of Figure 1a. In our

BMVC 2011 http://dx.doi.org/10.5244/C.25.14

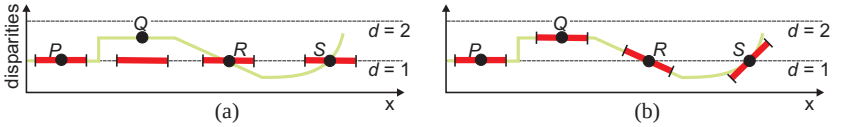

Figure 1: Support Regions (in 1D). The points of the green surface shall be reconstructed. Support regions are shown by red bars. (a) Fronto-parallel windows at integer disparities as used in standard methods. (b) Our support regions. We estimate a 3D plane at each point.

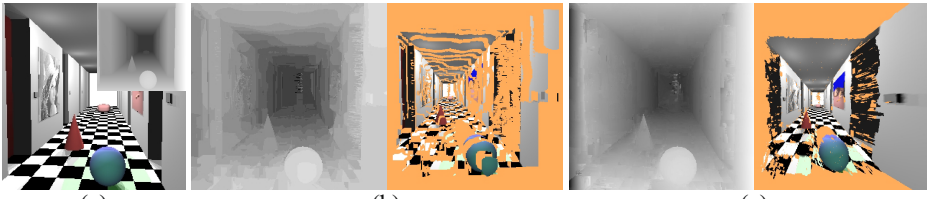

Figure 2: Advantage of our method. (a) Left image and ground truth disparities of the Corridor pair. (b) The disparity map computed with fronto-parallel windows approximates the slanted surfaces via many fronto-parallel ones. (c) Our slanted support windows correctly reconstruct the scene as a collection of slanted planar surfaces.

example, an optimal support can only be found for point P where the surface segment coincides with the fronto-parallel plane at the whole-valued disparity 1. Note that this case is very unlikely in practice. For all other cases, this method fails in finding an optimal support, because the surface segment lies at a sub-pixel disparity (Q), the point lies on a slanted planar surface (R) or on a rounded one (S). Our remedy is to compute an individual 3D plane at each pixel onto which the support region is projected. Figure 1b shows that this leads to considerably improved support regions. For example, we can model the optimal support for P, Q and R. At point S our planar model represents an oversimplification of the real rounded surface shape. However, we will show in experiments that even in this case the planar approximation works very well (e.g., see ball in Figure 2c). Figure 2 illustrates the advantage of our algorithm using the Corridor pair that consists of several highly slanted surfaces. In figure 2b, we run our algorithm allowing fronto-parallel windows only. The resulting 3D reconstruction is relatively poor, because a single slanted surface is approximated by several fronto-parallel surfaces (see floor, ceiling and walls). In contrast, our slanted support windows correctly reconstruct these slanted surfaces with sub-pixel precision (Figure 2c). [1]

There are several algorithms that compute disparity maps with sub-pixel precision. Often, sub-pixel information is derived in post-processing by fitting a parabola in the cost volume (e.g., [14]). The more sophisticated way (that we follow) is to account for sub-pixel precision directly in the matching. In the simplest form, this is accomplished by extending the label space, i.e., some fractional disparity values (half- or quarter-pixel) are considered in addition to the whole-valued ones (e.g., [6]).[2] Instead of adding additional fronto-parallel planes to the label space, one can add slanted ones [5]. However, this requires to first extract those planes that make up the scene, which is a non-trivial task considering that the number of candidate planes is infinite. For example, the very popular segmentation-based methods (e.g., [2, 17]) extract several planes using an initial disparity map in the first step. In the

[1] In our algorithm, it would be relatively straight-forward to replace our planar model with, e.g., a B-spline model [3]. However, regularization of this spline to avoid overfitting the data would require additional parameters.

[2] This typically goes along with an increased runtime. For example, when implementing this strategy in a standard local algorithm runtime is doubled or quadrupled when using half- and quarter-pixel precision, respectively.

matching step, these planes are checked to find the best-suited one for each pixel/segment. The problem is that if the correct plane has been missed in step 1, the matching step will fail. This is also the limitation of the local algorithm in [17] that we consider as closest related work. In the plane extraction step in [17], a disparity map using fronto-parallel windows is computed. This disparity map serves to extract a plane orientation at each pixel via plane fitting. In the matching step, cost aggregation is performed along the estimated plane orientations, i.e., slanted support windows are used. However, [17] cannot reconstruct highly slanted surfaces (e.g., the ground plane in the Teddy set of Figure 4), because the results of matching with fronto-parallel windows are too poor to allow inferring the correct plane orientation. Furthermore, quantization of disparity values is required.

In this work we propose an algorithm based on PatchMatch [1] to effectively solve the problem of finding a "good" slanted support plane at each pixel. In contrast to other local algorithms (e.g., [7, 9, 10, 16, 17]), we do not construct the full stereo cost volume, which is impossible in our case due to an infinite label space consisting of all 3D planes. Instead, we smartly traverse *parts* of it. This enables a one-shot optimization where planes and assignments of pixels to planes are estimated jointly, which effectively circumvents the problem of missing correct planes (see above). We can even reconstruct rounded surfaces where many slightly different support planes are needed. PatchMatch [1] itself is an approximate dense nearest neighbor algorithm, i.e., for each patch of one image an integer-valued (x,y)-vector to a similar colored patch in a second image is computed. Here, we use the PatchMatch idea of random search and propagation to find the nearest neighbor on the epipolar line *according to a plane*. This enables handling of slanted surfaces and sub-pixel precision. Our slanted support windows can also handle the well-known window foreshortening problem, which is specifically important for a large baseline stereo setup.

2 Algorithm

2.1 Model

For each pixel p of both frames, we search a plane f_p. We compute both left and right disparity maps, because, as most other local methods, we perform occlusion handling via left/right consistency checking. Once f_p has been found, we can compute p's disparity as

$$d_p = a_{f_p} p_x + b_{f_p} p_y + c_{f_p} \tag{1}$$

where a_{f_p}, b_{f_p} and c_{f_p} are the three parameters of plane f_p and p_x and p_y denote p's x- and y-coordinates. The plane f_p we desire to find is the one of minimum aggregated matching costs among all possible planes:

$$f_p = \underset{f \in \mathscr{F}}{\operatorname{argmin}}\, m(p,f), \tag{2}$$

where $\mathscr{F}$ denotes the set of all planes whose size is infinite. Note that this infinite label space prevents us from simply checking all possible labels, as one would do in standard local stereo matching where labels correspond to discrete disparity values. The aggregated costs for matching pixel p according to plane f are computed as

$$m(p,f) = \sum_{q \in W_p} w(p,q) \cdot \rho(q, q-(a_f q_x + b_f q_y + c_f)). \tag{3}$$

Here, W_p denotes a square window centered on pixel p. We will also apply our method for temporal stereo. In this case, W_p is no longer a 2D, but a 3D window where the third dimension is formed by pixels of preceding and consecutive frames of the video sequence,

as has recently been proposed in [10]. The weight function $w(p,q)$ is used to overcome the edge-fattening problem and implements the adaptive support weight idea [16]. It computes the likelihood for p and q lying on the same plane by looking at the pixels' colors, i.e., it returns high values if colors are similar:

$$w(p,q) = e^{-\frac{||I_p - I_q||}{\gamma}}. \tag{4}$$

Here, γ is a user-defined parameter and $||I_p - I_q||$ computes the L_1-distance of p and q's colors in RGB space. Our function can be seen as a simplified form of the weighting function proposed in [16].[3] Note that there is nothing that speaks against using other support weight functions (e.g., [7, 9]), which might potentially further improve results.[4] Let us now focus on the second part of equation (3). Here, we first compute q's disparity according to plane f and derive q's matching point in the other view q' by subtracting this disparity from q's x-coordinate.[5] The function $\rho(q,q')$ now computes the pixel dissimilarity between q and q':

$$\rho(q,q') = (1-\alpha)\cdot\min(||I_q - I_{q'}||, \tau_{col}) + \alpha\cdot\min(||\nabla I_q - \nabla I_{q'}||, \tau_{grad}), \tag{5}$$

where $||\nabla I_q - \nabla I_{q'}||$ denotes the absolute difference of gray-value gradients computed at q and q'. Since the x-coordinate of q' lies in the continuous domain, we derive its color and gradient values by linear interpolation. The user-defined parameter α balances the influence of color and gradient term. Parameters τ_{col} and τ_{grad} truncate costs for robustness in occlusion regions. This match measure has recently been applied in [9] and has the advantage of handling radiometric differences in the input images due to using the gradient.

2.2 Inference via PatchMatch

Let us now focus on the problem of finding a 3D plane at each pixel in both views that minimizes the cost given in equation (3). Our algorithm is based on PatchMatch [1]. Its basic idea is that in natural stereo pairs relatively large regions of pixels can be modeled by approximately the same plane. We find the plane for a region by initializing each pixel with a random plane. The hope is that after this random initialization at least one pixel of the region carries a plane that is close to the correct one. Note that this is very likely because we have many guesses, i.e., each pixel of the region represents one. Having a single "good" guess is already enough for the algorithm to work, since there is a propagation step that passes this plane on to the other pixels of the region. Apart from propagating the plane among spatial neighboring pixels as in [1], we introduce two new propagation steps, i.e., view propagation and temporal propagation. Finally, there is also a plane refinement step where we alter plane parameters to get closer to the optimal plane. We now explain details for these steps.

Random Initialization We assign each pixel of both views to a random plane. In principle, one can obtain a random plane by directly assigning random values to the three parameters a_f, b_f and c_f of a plane f. However, this strategy will not sample the space of all allowed planes evenly. In our method for computing a random plane at pixel (x_0, y_0), we first select a random disparity z_0 that lies in the range of allowed continuous disparity values. This gives us a point $P = (x_0, y_0, z_0)$ on our random plane. We now compute the plane's normal vector as a random unit vector $\vec{n} = (n_x, n_y, n_z)$. We then convert to the plane representation of equation

[3] We have removed the spatial term of [16] that compares pixel positions. In our experiments, improvement due to this term has been too small in order to justify another parameter.

[4] One can also use a symmetrical function that computes support weights in both images for increased robustness near occlusions [16]. In our experience, the practical benefit of this strategy is relatively low.

[5] If q resides in the right image, the disparity value is added.

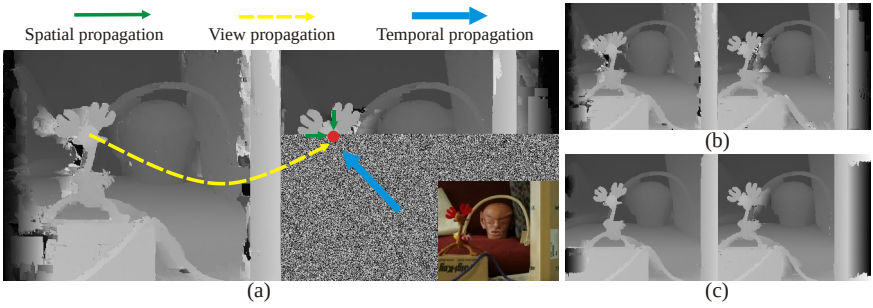

Figure 3: Different steps of the algorithm. (a) Left and right disparity maps at an intermediate step of the first iteration. All pixels up to the marked red one have already been processed. The other ones are still assigned to planes found in random initialization. We use three types of propagation illustrated by arrows. (b) Results after iteration 3. (c) Results after post-processing. We perform left/right consistency checking and occlusion filling.

(1) by $a_f := -\frac{n_x}{n_z}$, $b_f := -\frac{n_y}{n_z}$ and $c_f := \frac{n_x x_0 + n_y y_0 + n_z z_0}{n_z}$. Note that we can enforce fronto-parallel windows by setting $\vec{n} := (0,0,1)$. In addition, we can switch off sub-pixel precision by enforcing an integer-valued disparity for z_0.[6] We will do this in our experiments.

Iteration In an iteration, each pixel runs through four stages, i.e., (1) spatial propagation, (2) view propagation, (3) temporal propagation and (4) plane refinement. We first process all pixels of the left frame and then all pixels of the right image. In even iteration, we start with the top left pixel and traverse pixels in row-major order until we reach the bottom right pixel. In odd iterations, we reverse the order, i.e., we start with the right bottom pixel and stop at the top left one. In our experiment, we run three iterations. Figure 3 plots results obtained in different iterations.

Spatial Propagation The idea behind this form of propagation is that spatial neighboring pixels are likely to have similar planes. Let p denote the current pixel and f_p its plane. We evaluate whether assigning p to the plane f_q of a spatial neighbor q improves the costs of equation (3), i.e., we check the condition $m(p,f_q) < m(p,f_p)$. If this is the case, we accept f_q as p's new plane, i.e., $f_p := f_q$. In even iterations we consider the left and upper neighbors, whereas in odd iterations the right and lower neighbors are checked.

View Propagation Here, we exploit the strong coherency that exists between left and right disparity maps, i.e., a pixel and its matching point in the other view are likely to have similar planes. We check all pixels of the second view that have our current pixel p as a matching point according to their current plane.[7] Let p' be such a pixel and $f_{p'}$ denote its plane transformed to the first view. If $m(p,f_{p'}) < m(p,f_p)$, we set $f_p := f_{p'}$.

Temporal Propagation This form of propagation can only be used when working on stereo video sequences. The assumption is that a pixel p of the current video frame and a pixel p' at the same coordinates in the preceding or consecutive image are likely to have similar planes. Obviously, this assumption is more likely to hold if there is little motion in the sequence, which is oftentimes the case. We check the condition $m(p,f_{p'}) < m(p,f_p)$. If it is fulfilled, we set $f_p := f_{p'}$.

[6] The same adjustments need to be done in the plane refinement step to prevent this step from generating slanted support windows and sub-pixel disparities, respectively.

[7] When computing the matching point we obtain a continuous x-coordinate, which does not make sense here. Hence, we round to the closest whole-valued x-coordinate. Note this rounding procedure is the reason why planes of corresponding points are, in general, not exactly the same, but only similar. The other reason for different planes is the occlusion problem.

Plane Refinement The goal of this step is to refine the parameters of the plane f_p at pixel $p = (x_0, y_0)$ in order to further reduce the costs of equation (3). We convert f_p to the point plus normal vector representation. We have two parameters, i.e., $\Delta_{z_0}^{max}$ that defines the maximum allowed change of the 3D point's z-coordinate z_0 and Δ_n^{max} that sets a limit on the allowed change of components of the normal vector $\vec{n}$. We now estimate Δ_{z_0} as a random value that lies in the interval $[-\Delta_{z_0}^{max}, \Delta_{z_0}^{max}]$ and compute $z_0' := z_0 + \Delta_{z_0}$, which gives us a new 3D point $P' = (x_0, y_0, z_0')$. Analogously, we estimate three random values of the interval $[-\Delta_n^{max}, \Delta_n^{max}]$, which form the components of the vector $\vec{\Delta}_n$. We now estimate the modified normal vector as $\vec{n'} := u(\vec{n} + \vec{\Delta}_n)$ where $u()$ computes the unit vector. Finally, we convert the plane defined by P' and $\vec{n'}$ to the representation of equation (1), which gives as the modified plane f_p'. If $m(p, f_p') < m(p, f_p)$, we accept the plane, i.e., $f_p := f_p'$.

This refinement procedure is iterated. We start by setting $\Delta_{z_0}^{max} := maxdisp/2$, where $maxdisp$ is the maximum allowed disparity, and $\Delta_n^{max} := 1$. After each refinement, we set $\Delta_{z_0}^{max} := \Delta_{z_0}^{max}/2$ and $\Delta_n^{max} := \Delta_n^{max}/2$, which exponentially reduces the search scope. We stop if $\Delta_{z_0}^{max} < 0.1$. The idea is to allow large changes in the first iterations, which makes sense if the current plane is completely wrong. In later iterations, we sample planes that are very close to our current one, which allows capturing disparity details, e.g., at rounded surfaces.[8]

2.3 Post-Processing

We now apply occlusion treatment via left/right consistency checking. For each pixel p, we compute its matching point p' in the other view.[9] We now check the condition $|d_p - d_{p'}| \leq 1$. If this condition is false, pixel p is invalidated. This consistency check typically fails for occluded pixels, but also for mismatched ones.

We now fill in the disparity for invalidated pixels. For an invalidated pixel p, we search its closest valid pixel to the left and to the right. The planes f^l and f^r of both points are recorded. We now compute the disparities when assigning p to f^l and f^r (equation (1)) and select the lower of the two as p's filled-in disparity. Selecting the lower disparity is motivated by the fact that occlusion occurs at the background. Note that this filling scheme extrapolates planes, instead of replicating constant disparities as is commonly done (e.g., [7, 9]). Hence we can also correctly treat slanted surfaces at this stage. However, the obvious problem is that this strategy generates horizontal streaks in the disparity map. To weaken this problem, we apply a weighted median filter on the filled-in disparities (also see [9]). The weight mask for the median filter is computed by equation (4). We use the same setting for γ and the window size that we have used in the matching process. (This also means that we apply a 3D median filter when doing temporal stereo.) Note that all pixels that have survived left/right checking are not affected by this operation. Disparity maps before and after post-processing are shown in Figures 3b and 3c, respectively.

2.4 Building a Data Term for Global Methods

Window-based matching costs allow global algorithms to inherit the ability to precisely capture depth discontinuities from adaptive support weight approaches [15]. Hence, it is no longer necessary to compute an explicit color segmentation, which is currently needed by many state-of-the-art methods (e.g., [3, 13]).[10] The problem is that one also inherits a disadvantage of window-based matching, i.e., the bias towards fronto-parallel surfaces. Note that

[8] As an alternative to the current refinement step, we could do gradient descent.

[9] The continuous x-coordinate of the matching point is rounded to the next whole-valued x-coordinate.

[10] The second argument for using window-based correlation in global matching is that one can use robust radiometric-insensitive matching costs such as NCC or Census that only work on a window basis.

this problem is not present in standard global data terms that compute the match measure on a pixel basis (1×1 match window). The data term proposed in the following allows global methods to take advantage of adaptive support weight windows without fronto-parallel bias.

We now construct a cost volume that stores the costs for matching each pixel at each allowed whole-valued disparity. (The discretization of disparity is the obvious downside of this approach.) Let us now compute the costs for matching all pixels at a fixed disparity d. This is accomplished by setting a minimum allowed disparity $mindisp := d - 0.5$ and a maximum allowed disparity $maxdisp := d + 0.5 - eps$ where eps is a very small value. We now run the PatchMatch algorithm of Section 2.2.[11] The costs for matching pixel p at disparity d are now derived by looking up p's plane f_p and evaluating the costs $m(p, f_p)$.

We embed the above data costs in a global algorithm that optimizes an energy function consisting of data and smoothness terms. The data term is thereby taken from [8] and enables symmetrical occlusion handling. As a smoothness term, we use the second-order term of [13], which overcomes the fronto-parallel bias of competing terms (e.g., truncated linear model). The exact form of this energy is found in the supplementary material. We perform energy minimization using the α-expansion algorithm [4].

3 Results

Let us first focus on the local method. We apply a window of 35×35 pixels and set the other parameters to $\{\gamma, \alpha, \tau_{col}, \tau_{grad}\} := \{10, 0.9, 10, 2\}$. We implement two competitors in our PatchMatch framework: (1) fronto-parallel windows matched at whole-valued disparities, (2) fronto-parallel windows matched at continuous sub-pixel positions (see Section 2.2). Also these competitors can be seen as contributions of this paper. Competitor (1) should be regarded in the context of techniques that speed up the adaptive support weight approach. Previous methods [9, 10] use special adaptive support weight functions to make the runtime independent of the match window size. In contrast, our method keeps the dependency on the window size, but removes dependency on the disparity search range.[12] In contrast to [9, 10], this strategy works for arbitrary adaptive support weighting functions (e.g., [7]). Speaking of runtime, PatchMatch Stereo consumes approximately one minute on a Middlebury pair.[13] Competitor (2) overcomes the need to quantize disparities for sub-pixel matching, as required in previous work [6, 17]. Note that already this method is very successful in sub-pixel estimation when compared against the state-of-the-art (see Table 1 - error threshold 0.5).

Figure 4 plots results of our two competitors (Figures 4a and 4b) and our slanted window algorithm (Figure 4c) on the Middlebury set [11]. The most obvious difference is found in the Teddy test set where, in contrast to its competitors, our method correctly reconstructs the highly slanted ground plane. Let us now look at Figure 4d where we show 3D reconstructions of objects from the Middlebury images. The problem of competitor (1) is that it reconstructs a single slanted surface via various fronto-parallel disparity segments so that, e.g., the 3D reconstruction of the slanted plane from the Venus set shown top-left in Figure 4d looks like a staircase. Allowing sub-pixel matching (competitor (2)) only weakens this fronto-parallel bias. In contrast, our slanted windows can reconstruct the Venus plane as what it is - a planar

[11] We enforce minimum and maximum allowed disparity by modifying equation (3) so that $m(p, f)$ returns infinite costs if p's disparity according to plane f lies outside the interval $[mindisp, maxdisp]$. We also modify the random initialization step so that it only computes planes that lead to disparities inside this interval.

[12] This is also why our algorithm should be efficient when computing optical flow, which is future work.

[13] We have an unoptimized C implementation. An obvious idea is to run the algorithm on the GPU, which we have not yet done. For the special case of using integer disparities, running our plane refinement step and three overall iterations might be too excessive. One or two iterations and turning off plane refinement may already be sufficient to get good results and would considerably reduce runtime.

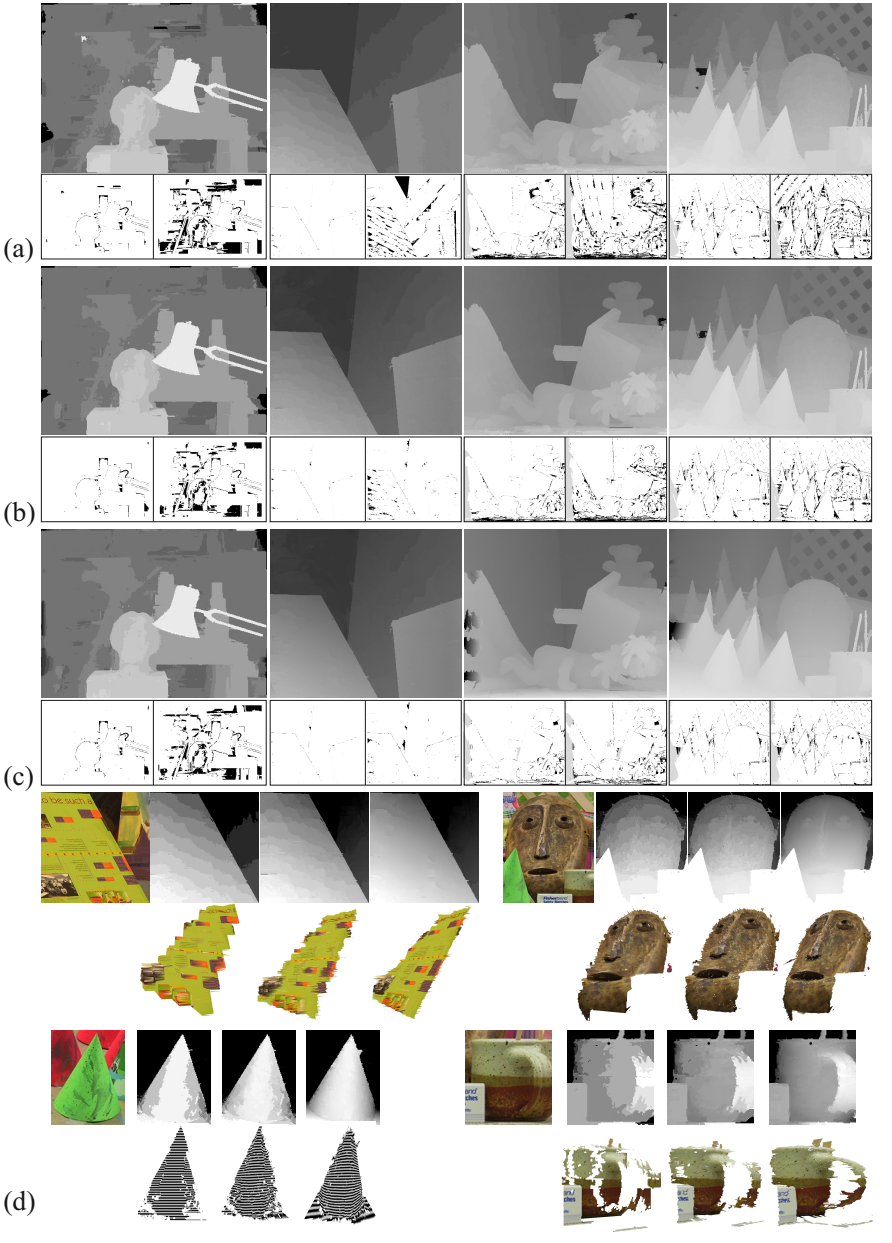

Figure 4: Middlebury results. (a) Fronto-parallel windows matched at integer disparities. We show disparity map (top), disparity errors > 1 pixel (bottom left) and disparity errors > 0.5 pixels (bottom right). (b) Fronto-parallel windows matched at continuous disparities. (c) Our slanted support windows. Note, e.g., the correctly reconstructed slanted ground plane in the Teddy set. For Teddy, our method is the new Middlebury top-performer (error > 1 pixel measured in unoccluded regions). (d) Crops of above disparity maps with different scaling to highlight disparity details. Using fronto-parallel windows (left and middle disparity maps) results in a bias towards reconstructing fronto-parallel disparity segments. Our slanted windows (right disparity maps) overcome this bias (see corresponding 3D reconstructions).

Algorithm	thresh.	Rank	Tsukuba			Venus			Teddy			Cones		
			nocc	all	disc	nocc	all	disc	nocc	all	disc	nocc	all	disc
Slanted Support	1.0	11	2.09_{57}	2.33_{43}	9.31_{54}	0.21_{20}	0.39_{15}	2.62_{26}	$\mathbf{2.99}_{1}$	8.16_{8}	9.62_{2}	2.47_{4}	7.80_{8}	7.11_{6}
Fronto-Par. Sub-Pixel	1.0	22	2.13_{57}	2.32_{43}	9.92_{60}	0.25_{27}	0.42_{16}	2.76_{27}	5.52_{17}	10.5_{17}	14.2_{15}	3.23_{27}	8.33_{17}	8.20_{18}
Fronto-Par. Integer	1.0	32	2.23_{58}	2.44_{44}	9.18_{53}	0.25_{25}	0.41_{16}	2.21_{16}	6.73_{34}	12.2_{34}	16.9_{38}	3.71_{35}	8.90_{27}	9.31_{35}
Slanted Support	0.5	2	15.0_{37}	15.4_{36}	20.3_{45}	1.00_{4}	1.34_{4}	7.75_{7}	$\mathbf{5.66}_{1}$	11.8_{2}	$\mathbf{16.5}_{1}$	$\mathbf{3.80}_{1}$	$\mathbf{10.2}_{1}$	$\mathbf{10.2}_{1}$
Fronto-Par. Sub-Pixel	0.5	4	14.1_{36}	14.4_{32}	19.2_{35}	1.73_{7}	2.15_{7}	7.84_{7}	9.63_{6}	16.2_{8}	22.6_{8}	7.08_{24}	12.6_{16}	13.2_{11}
Fronto-Par. Integer	0.5	41	19.1_{51}	19.4_{46}	21.2_{53}	7.57_{56}	8.09_{56}	13.4_{38}	14.2_{36}	20.7_{39}	29.1_{37}	11.6_{53}	16.9_{42}	18.3_{41}

Table 1: Quantitative Middlebury results for the disparity maps of Figure 4. Subscripts denote rankings in the table. We use error threshold 1.0 (Middlebury default threshold) and threshold 0.5 to measure sub-pixel performance. Our method (*Slanted Support*) performs particularly well for the more complex scenes of the benchmark, i.e., Teddy and Cones.

surface. Also note that our slanted support windows are effective in reconstructing rounded surfaces (e.g., the handle of the cup in Figure 4d). Table 1 gives quantitative results that are taken from the Middlebury table. When using the Middlebury default error threshold of one pixel, our method takes rank 11 among approximately 110 algorithms and is the best-performing local method in this ranking. Note that for Teddy, our slanted window method is the top-performer according to the error percentage in unoccluded regions, which is because most other algorithms run into problems when reconstructing the slanted ground plane. When using an error threshold of 0.5 to assess sub-pixel performance, our method takes rank 2 in the table and is the top-performer on the complex Teddy and Cones images.[14] We have also applied our method on stereo videos (Figure 5). Here, we have employed a 3D window of size $71 \times 71 \times 3$ pixels. We use a larger spatial window to account for the high resolution of these sequences (up to 1024×576 pixels). Note we can handle high-resolution images, since our algorithm is memory-efficient, i.e., one only needs to hold the current plane parameters and corresponding aggregated costs at each pixel in memory.

Let us now focus on evaluating the data term described in Section 2.4 using the Teddy set. We first construct the cost volume via pixel-wise correlation (no aggregation performed) using the match measure of equation (5). (Pixel-wise correlation is what standard global methods commonly do.) The corresponding disparity map of Figure 6a shows that disparity borders cannot be well-preserved when using this data term.[15] We now construct the cost volume using adaptive support weight windows, but only allow fronto-parallel windows. Figure 6b shows that results at disparity borders are improved, but the algorithm fails to capture the slanted ground plane. In contrast, our data term with slanted windows correctly reconstructs object boundaries *and* the slanted ground plane.

Let us give our general opinion. Local adaptive support weight methods are starting to outperform global methods on Middlebury. Also the global algorithm we have used here cannot compete with our local results on the Middlebury images. However, we believe that this is only because the Middlebury images are ideal for local methods, i.e., almost no untextured regions. Global methods still make sense, because they allow occlusion handling directly in the matching process and can treat large untextured regions. As an example, we plot the Middlebury Plastic set that contains large untextured regions. As seen from Figure 6d our local method fails, whereas the global method (used with our slanted window data term) can correctly reconstruct disparity (Figure 6e).

[14] The sub-pixel top-performer is, to our knowledge, not published yet. Also note it hardly makes sense to measure sub-pixel performance on Tsukuba, because the ground truth does not have sub-pixel information.

[15] Also the ground plane could not be reconstructed, although there is no fronto-parallel bias in this cost volume. We found that this is, because optimization did not work very well, i.e., QPBO leaves a large percentage of pixels unlabelled. This has not been a problem when using the adaptive support weight cost volumes, which is most likely because they contain less ambiguity.

Figure 5: Temporal stereo. We have applied our local algorithm on two stereo videos that are found in the supplementary material.

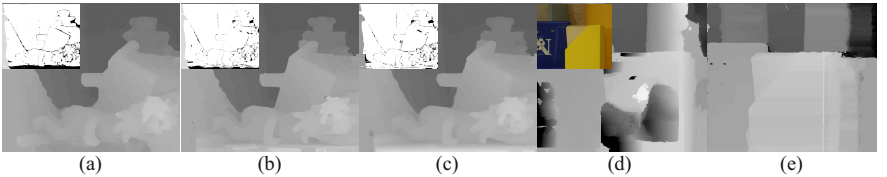

Figure 6: Global matching. (a) Disparities and errors > 1 pixel when using pixel-wise correlation in data term computation. (b) Fronto-parallel adaptive support weight windows. (c) Our slanted windows used in data term computation allow preserving depth discontinuities and reconstructing the slanted ground plane. (d) Our local algorithm fails on the untextured Plastic set. (e) The global algorithm succeeds.

4 Conclusions

We have proposed a local algorithm that computes a 3D plane at each pixel onto which the support region is projected. The ideal algorithm to solve the challenging task of finding these planes is PatchMatch. Our results show impressive sub-pixel results and rank excellently in the Middlebury benchmark. We have also demonstrated that our slanted windows can serve as a data term for global methods. In future work, we will extend this algorithm to compute optical flow and perform a GPU implementation that might lead to real-time performance.

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

References

[1] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (SIGGRAPH)*, 2009.

[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] M. Bleyer, C. Rother, and P. Kohli. Surface stereo with soft segmentation. In *CVPR*, 2010.

[4] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *PAMI*, 23(11):1222–1239, 2001.

[5] D. Gallup, J. Frahm, P. Mordohai, Q. Yang, and M. Pollefeys. Real-time plane-sweeping stereo with multiple sweeping directions. In *CVPR*, 2007.

[6] S. Gehrig and U. Franke. Improving sub-pixel accuracy for long range stereo. In *ICCV VRML workshop*, 2007.

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

[8] V. Kolmogorov and R. Zabih. Multi-camera scene reconstruction via graph cuts. In *ECCV*, 2002.

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

[10] 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.

[11] 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://vision.middlebury.edu/stereo/.

[12] H. Tao, H. Sawhney, and R. Kumar. A global matching framework for stereo computation. In *ICCV*, pages 532–539, 2001.

[13] O. Woodford, P. Torr, I. Reid, and A. Fitzgibbon. Global stereo reconstruction under second order smoothness priors. In *CVPR*, 2008.

[14] Q. Yang, R. Yang, J. Davis, and D. Nister. Spatial-depth super resolution for range images. In *CVPR*, 2007.

[15] Q. Yang, L. Wang, R. Yang, H. Stewenius, and D. Nister. Stereo matching with color-weighted correlation, hierarchical belief propagation and occlusion handling. *PAMI*, 2009.

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

[17] Y. Zhang, M. Gong, and Y. Yang. Local stereo matching with 3D adaptive cost aggregation for slanted surface modeling and sub-pixel accuracy. In *ICPR*, 2008.