ITERATIVE REFINEMENT FOR REAL-TIME LOCAL STEREO MATCHING

Maarten Dumont, Patrik Goorts, Steven Maesen, Donald Degraen, Philippe Bekaert, Gauthier Lafruit

Hasselt University - tUL - iMinds
Expertise Centre for Digital Media
Wetenschapspark 2
3590 Diepenbeek, Belgium
firstname.lastname@uhasselt.be

Université Libre de Bruxelles / Brussels University
LISA department
Av. F.D. Roosevelt 50 CP165/57
1050 Brussels, Belgium
gauthier.lafruit@ulb.ac.be

ABSTRACT

We present a novel iterative refinement process to apply to any stereo matching algorithm. The quality of its disparity map output is increased using four rigorously defined refinement modules, which can be iterated multiple times: a disparity cross check, bitwise fast voting, invalid disparity handling, and median filtering. We apply our refinement process to our recently developed aggregation window method for stereo matching that combines two adaptive windows per pixel region [2]; one following the horizontal edges in the image, the other the vertical edges. Their combination defines the final aggregation window shape that closely follows all object edges and thereby achieves increased hypothesis confidence. We demonstrate that the iterative disparity refinement has a large effect on the overall quality, especially around occluded areas, and tends to converge to a final solution. We perform a quantitative evaluation on various Middlebury datasets. Our whole disparity estimation process supports efficient GPU implementation to facilitate scalability and real-time performance.

***Index Terms*—** Local Stereo Matching, Disparity Estimation, Iterative Disparity Refinement, Adaptive Aggregation Windows, Real-Time

1. INTRODUCTION

Stereo matching uses a pair of images to estimate the apparent movement of each pixel from one image to the next. This apparent movement is more commonly know as the parallax effect, and is expressed in a disparity map for the image under consideration. Local disparity estimation algorithms commonly consist of four stages as defined by Scharstein and Szeliski [6], namely cost calculation, cost aggregation, disparity selection, and refinement.

This paper's main contribution is situated in the refinement stage. In section 4 we present an iterative refinement method to significantly improve the quality of an initially estimated disparity map. One iteration of the refinement consists itself again of four strictly defined stages: a disparity cross-check (section 4.1), bitwise fast voting (section 4.2), invalid disparity handling (section 4.3), and median filtering (section 4.4).

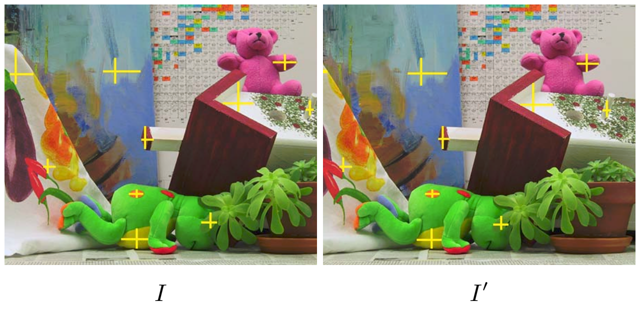

Fig. 1: The left (I) and right (I') images of the Middlebury Teddy scene. Some axis quadruplets from Eq. 1 are drawn as yellow crosses of their horizontal and vertical axes.

Our refinement (and more specifically the bitwise fast voting) depends heavily on local support windows that we first define in section 2. Unlike the method of Zhang et al. [11], we propose two edge-aware windows, one for the horizontal and one for the vertical edge directions. The windows are constructed to cover image patches of similar color, where we assume that pixels with similar color belong to the same object, and therefore should have the same disparity value.

Although applicable to a disparity map that was computed using any disparity estimation algorithm, the ultimate success of our refinement still depends on the quality of the initial disparity map. We therefore use our previously developed approach to estimate an initial disparity map [2]. It is summarized in section 3 and covers the other three stages previously mentioned: cost calculation, cost aggregation (which relies again on the support windows from section 2), and disparity selection.

We demonstrate the strength of our method on various standard Middlebury datasets [6] in section 5, where we quantitatively and qualitatively compare our iteratively refined disparity maps with their ground truth.

Because pixel-wise algorithms map very well to parallel hardware, our implementation in CUDA achieves real-time performance. This is harder to achieve when using global estimation methods such as graph cuts (and similar energy minimization techniques) [4, 8, 9], segmented patches [13], and spatiotemporal consistency approaches [1].

The full algorithmic chain is quite extensive, but the left I and right I' images of the Middlebury Teddy [6] scene (see Fig. 1) serve as a running example throughout this paper. More details can also be found in [2].

2. LOCAL SUPPORT WINDOWS

We first describe how to construct two suitable local support windows that will be used to both estimate the initial disparity map in section 3, and during the iterative refinement in section 4.

For every pixel p of the left image I, we first determine a horizontal axis $\mathbb{H}(p)$ and vertical axis $\mathbb{V}(p)$ crossing in p. These two axes can be represented as a quadruplet $\mathbb{A}(p)$:

$$\mathbb{A}(p) = (h_p^-, h_p^+, v_p^-, v_p^+) \tag{1}$$

where the component h_p^- represents how many pixels the horizontal axis extends to the left of p, v_p^+ represents how many pixels the vertical axis extends above p, and so forth. Also see Fig. 1.

To determine each component, we keep extending an axis until the difference between p and the outermost pixel q becomes too large:

$$\max_{c \in \{r,g,b\}} |I_c(p) - I_c(q)| \leq \tau \tag{2}$$

where $I_c(p)$ is the red, green or blue color channel of pixel p, and τ is the threshold for color consistency. We also stop extending if the size exceeds a maximum predefined length.

Using these four components, we define two local support windows for pixel p, referred to respectively as the horizontal local support window $W^H(p)$ and the vertical local support window $W^V(p)$ (illustrated in Fig. 2).

To construct the horizontal window $W^H(p)$, we first need to create its vertical axis based on the values of v_p^- and v_p^+. We call this the primary vertical axis $\mathbb{V}(p)$. Next, we consider the values of h_q^-, and h_q^+ for each pixel q on the primary vertical axis. These define a horizontal axis per pixel q on the primary vertical axis. These axes are called the subordinate horizontal axes $\mathcal{H}(q)$. In short, this results in the orthogonal decomposition:

$$W^H(p) = \bigcup_{q \in \mathbb{V}(p)} \mathcal{H}(q) \tag{3}$$

Completely analogous, but in the other direction, we define the vertical local support window $W^V(p)$ by creating a

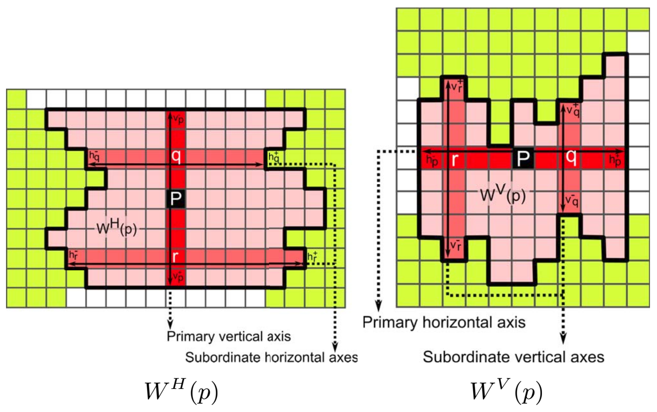

Fig. 2: Derivation of the horizontal ($W^H(p)$, Eq. 3) and vertical ($W^V(p)$, Eq. 4) local support windows for pixel p, using its axis-defining quadruplet $(h_p^-, h_p^+, v_p^-, v_p^+)$ of Eq. 1.

primary horizontal axis $\mathbb{H}(p)$ using h_p^-, and h_p^+, and on this axis create the subordinate vertical axes $\mathcal{V}(q)$:

$$W^V(p) = \bigcup_{q \in \mathbb{H}(p)} \mathcal{V}(q) \tag{4}$$

By requiring only the single quadruplet $(h_p^-, h_p^+, v_p^-, v_p^+)$ to define both windows, we reduce memory usage and access, which is a serious consideration when using GPU computing.

Constructed this way, our windows are sensitive to edges in the image. The horizontal window $W^H(p)$ will fold nicely around vertical edges, because the width of each subordinate horizontal axis is variable. Horizontal edges are not followed as accurately, as the height of the window is fixed and only determined by its primary vertical axis. This situation, however, is reversed for the vertical window $W^V(p)$.

The notation $W'^H(p')$ and $W'^V(p')$ represents the local support windows for each pixel p' in the right image I'.

3. INITIAL DISPARITY ESTIMATION

In this section, our goal is to first estimate an initial disparity map [2] that will serve as input to our iterative refinement process in section 4. First, we consider each disparity and calculate (in section 3.1) for each pixel in the left image the difference (i.e. matching cost) between that pixel and the corresponding pixel in the right image, based on the disparity under consideration. Next, the costs of neighboring pixels are aggregated (in section 3.2) to obtain a more confident matching cost. Typically for this stage, variable window sizes [3, 7] or variable weights inside the windows [5, 10] are used to increase aggregation quality. Instead, we combine the horizontal and vertical local support windows from section 2 into a global support window. Opposite to the method of Zhang et al. [11], which uses only a horizontal window, we combine two directions, so that vertical edges are not favored. Once the costs are aggregated per pixel and per disparity value, the most suitable disparity with the lowest cost is selected (in section 3.3).

3.1. Per-Pixel Matching Cost

For a disparity hypothesis $d \in [d_{min}, d_{max}]$ and pixel p of the left image I, consider the raw per-pixel matching cost $E_d(p)$, defined as the *Sum of Absolute Differences* (SAD):

$$E_d(p) = \frac{\sum_{c \in \{r,g,b\}} |I_c(p) - I'_c(p')|}{e_{max}} \tag{5}$$

where pixel p in the left image I is compared with pixel p' in the right image I', and the coordinates of $p = (x_p, y_p)$ and $p' = (x_{p'}, y_{p'})$ relate to the disparity hypothesis d as $x_{p'} = x_p - d$, $y_{p'} = y_p$. The constant e_{max} normalizes the cost $E_d(p)$ to the floating point range $[0, 1]$.

We calculate $E_d(p)$ for each pixel p and refer to E_d as the per-pixel left confidence (or cost) map for disparity d. Similarly the per-pixel right confidence map E'_d can be constructed by calculating $E'_d(p')$ for each pixel p' analogously to Eq. 5, with the x-coordinates of p and p' now related as $x_p = x_{p'} + d$.

3.2. Cost Aggregation over Global Support Windows

To reliably aggregate costs, we must simultaneously consider both local support windows $W(p)$ for pixel p in the left image and $W'(p')$ for pixel p' in the right image. If we only consider the local support window $W(p)$, the matching cost aggregation will be polluted by outliers in the right image, and vice versa. Therefore, while processing for disparity hypothesis d, the two local support windows are combined into what we call a global support window $U_d(p)$. Distinguishing again between horizontal and vertical support windows, they are defined as:

$$U_d^H(p) = W^H(p) \cap W'^H(p') \tag{6}$$

$$U_d^V(p) = W^V(p) \cap W'^V(p') \tag{7}$$

where the coordinates of $p = (x_p, y_p)$ and $p' = (x_{p'}, y_{p'})$ are again related to the disparity hypothesis d as $x_{p'} = x_p - d$, $y_{p'} = y_p$. In practice, this simplifies beautifully to taking the component-wise minimum of their axis quadruplets from Eq. 1:

$$\mathbb{A}_d(p) = \min\left(\mathbb{A}(p), \mathbb{A}'(p')\right) \tag{8}$$

Two more confident matching costs $\varepsilon_d^H(p)$ and $\varepsilon_d^V(p)$ can now be aggregated over each pixel s of the horizontal and vertical global support windows $U_d^H(p)$ and $U_d^V(p)$ respectively:

$$\varepsilon_d^H(p) = \frac{1}{\|U_d^H(p)\|} \sum_{s \in U_d^H(p)} E_d(s) \tag{9}$$

$$\varepsilon_d^V(p) = \frac{1}{\|U_d^V(p)\|} \sum_{s \in U_d^V(p)} E_d(s) \tag{10}$$

where the number of pixels $\|U_d(p)\|$ in the support window acts as a normalizer.

To combine $\varepsilon_d^H(p)$ and $\varepsilon_d^V(p)$ into the final aggregated cost $\varepsilon_d(p)$, we use a weighted sum:

$$\varepsilon_d(p) = \alpha\, \varepsilon_d^H(p) + (1 - \alpha)\, \varepsilon_d^V(p) \tag{11}$$

where α is a weighting parameter between 0 and 1 to steer the algorithm between horizontal and vertical windows.

The aggregation is repeated over the right image, which means computing $\mathbb{A}'_d(p') = \min\left(\mathbb{A}(p), \mathbb{A}'(p')\right)$, with p and p' now related as $x_p = x_{p'} + d$ and from there setting up an analogous reasoning to end up at the right aggregated confidence map ε'_d.

The global support windows $U_d^H(p)$ and $U_d^V(p)$ can be orthogonally decomposed analogously to Eq. 3 and Eq. 4, which is key to a fast and efficient implementation [2].

3.3. Disparity Selection

After the left and right aggregated confidence maps have been computed for every disparity $d \in [d_{min}, d_{max}]$, the best disparity per pixel (i.e. the one with lowest cost $\varepsilon_d(p)$) is selected using a *Winner-Takes-All* approach:

$$D_W(p) = \underset{d \in [d_{min}, d_{max}]}{\arg\min}\ \varepsilon_d(p) \tag{12}$$

which results in the disparity maps D_W for the left image and D'_W for the right image, both shown in Fig. 3. These disparity maps will serve as input to the iterative refinement process described next in section 4.

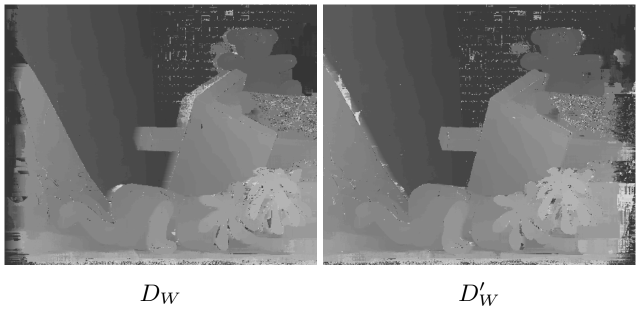

Fig. 3: Left and right Winner-Takes-All disparity maps, as determined by Eq. 12.

4. ITERATIVE DISPARITY REFINEMENT

We now iteratively refine the two initial disparity maps D_W and D'_W. One iteration consists of four stages. First we cross-check the disparities for consistency between the two disparity maps in section 4.1. Next, the local support windows as described in section 2 are employed again, to update a pixel's disparity with the disparity that appears most inside its windows. This method is the most powerful and is detailed in section 4.2. Any invalid disparities that remain after this are handled in section 4.3. In the last stage in section 4.4 the disparity map is median filtered to remove any remaining speckle noise. Finally, we initialize for the next iteration in section 4.5.

4.1. Disparity Cross-check

A left-to-right cross-check means that for each of the pixels p of the left disparity map D_W, the corresponding pixel p' is determined in the right image based on the disparity value $D_W(p)$, and the disparity value $D'_W(p')$ in the right disparity map is compared with $D_W(p)$. If they differ, the cross-check fails and the disparity is marked as invalid. Introducing the superscript $i \geq 1$ to denote the current refinement iteration, this is expressed as:

$$D_C^i(p) = \begin{cases} D_W^{i-1}(p) & \text{if } D_W^{i-1}(p) = D_W'^{i-1}(p') \\ \textit{INVALID} & \text{otherwise} \end{cases} \tag{13}$$

with $D_W^0 = D_W$ and $D_W'^0 = D'_W$, and with p now related to p' as $x_{p'} = x_p - D_W^{i-1}(p)$, $y_{p'} = y_p$. The process is then reversed for a right-to-left cross-check of the disparity map $D_W'^{i-1}$, which leaves us with the left and right cross-checked disparity maps D_C^i and $D_C'^i$.

Invalid disparities are most likely to occur around edges in the image, where occlusions are present in the scene. In Fig. 4 we show these occluded regions as pure black (marked as invalid) pixels.

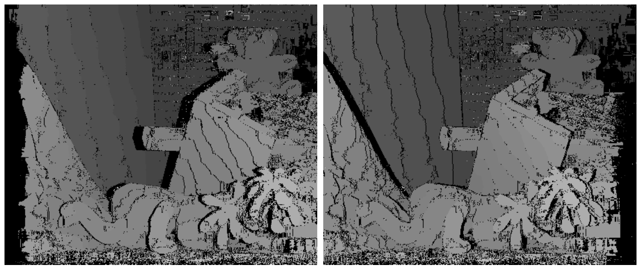

Fig. 4: Left-to-right and right-to-left cross-checked disparity maps, as determined by Eq. 13.

4.2. Bitwise Fast Voting over Local Support Windows

This second stage updates a pixel's disparity with the disparity that is most present inside its local support windows $W^H(p)$ and $W^V(p)$ as defined in section 2. We may say that this is valid, because pixels in the same window have similar colors by definition, and therefore with high probability belong to the same object and should have the same disparity. Confining the search to the local support windows also ensures that we greatly reduce the risk of edge fattening artifacts.

To efficiently determine the most frequent disparity value within a support window, we apply a technique called *bitwise fast voting* [12] and adapt it to handle both horizontally and vertically oriented support windows. At the core of the bitwise fast voting technique lies a procedure that derives each bit of the most frequent disparity independently from its other bits.

First consider a pixel p with local support window $W(p)$. We sum the k^{th} bit $b_k(s)$ (either 0 or 1) of the disparity value $D_C^i(s)$ of all pixels s in the support window, and call the result $B_k(p)$ (for clarity, we drop the superscript i for a moment). Furthermore distinguishing again between horizontal and vertical support windows, this gives:

$$B_k^H(p) = \sum_{s \in W^H(p)} b_k(s) \tag{14}$$

$$B_k^V(p) = \sum_{s \in W^V(p)} b_k(s) \tag{15}$$

The k^{th} bit $D_B^{i|k}(p)$ of the final disparity value $D_B^i(p)$ is then decided as:

$$D_B^{i|k}(p) = \begin{cases} 1 & \text{if } B_k(p) > \beta \times N(p) \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

where $\beta \in [0, 1]$ is a sensitivity factor that we will come back to below, and $B_k(p)$ and $N(p)$ are determined by a weighted sum:

$$B_k(p) = \alpha\ B_k^H(p) + (1-\alpha)\ B_k^V(p) \tag{17}$$

$$N(p) = \alpha\ \left\|W^H(p)\right\| + (1-\alpha)\ \left\|W^V(p)\right\| \tag{18}$$

where α is as before in Eq. 11.

Intuitively, for a pixel p, Eq. 16 states that the k^{th} bit of its final disparity value is 1 if the k^{th} bit appears as 1 in most of the disparity values under its local support window. The number of actual appearances of 1 are counted in $B_k(p)$, whereas the maximum possible appearances of 1 is represented by the window size $N(p)$. The sensitivity factor β controls how many appearances of 1 are required to confidently vote the result and is best set to 0.5.

Certain disparities might be invalid due to the cross-check of $D_C^i(p)$ in section 4.1. While counting bit votes, we must take this into account by reducing $N(p)$ accordingly. This way the algorithm is able to update an invalid disparity by depending on votes from valid neighbors only, and thereby reliably fill in occlusions and handle part of the image borders. Black patches in Fig. 5 are remaining invalid disparities that the bitwise vast voting was not able to fill in. Lastly, the bit votes can be counted very efficiently by orthogonally separating Eq. 14 and Eq. 15 [2].

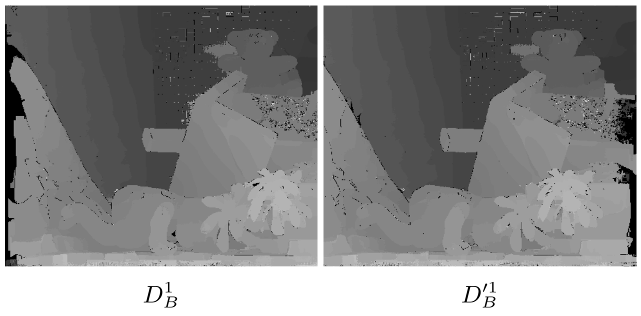

Fig. 5: Left and right disparity maps after bitwise fast voting, as determined by Eq. 16.

4.3. Invalid Disparity Handling

The bitwise fast voting removes many invalid disparities by replacing them with the most occurring valid value inside their windows. It will fail however if the window does not contain any valid values, or in other words, if $N(p) = 0$ in Eq. 16. This occurs mostly near the borders of the disparity maps, but also can manifest itself anywhere in the image where the occlusions are large enough.

For each remaining pixel with an invalid disparity, we search to the left and to the right on its scanline for the closest valid disparity, and store it in the corrected disparity map D_I^i (shown in Fig. 6). Unlike the bitwise fast voting, this scanline search is necessarily not confined to image patches of similar colors.

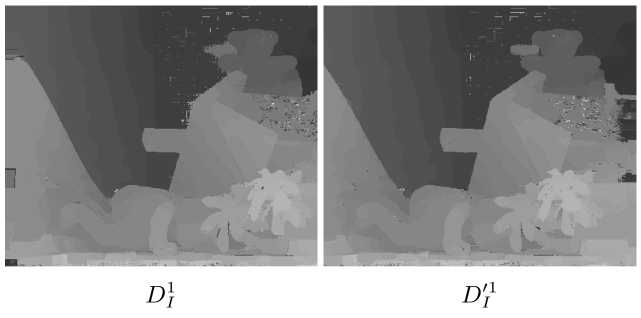

Fig. 6: Left and right disparity maps after remaining invalid disparities have been filled in, as described in section 4.3.

4.4. Median Filter

In the last refinement step, small disparity outliers are filtered using a 3×3 median filter, resulting in the final disparity maps (for the current iteration) D_M^i and $D_M'^i$ shown in Fig. 7. A median filter has the property of removing speckle noise, in this case caused by disparity mismatches, while returning a sharp signal (unlike an averaging filter).

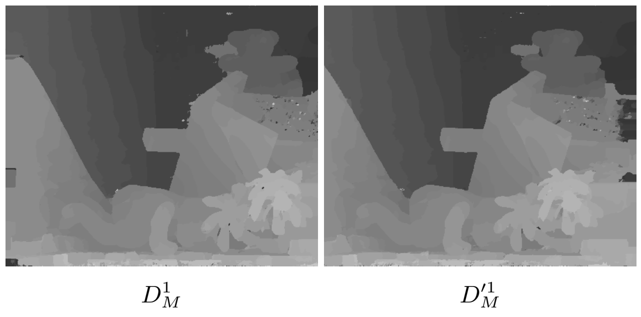

Fig. 7: Left and right disparity maps after application of a 3×3 median filter, as described in section 4.4.

4.5. The Next Iteration

This completes one iteration of the disparity refinement. The next iteration $i + 1$ immediately starts again with the disparity cross-check of Eq. 13, by setting $D_W^i = D_M^i$ and $D_W'^i = D_M'^i$.With each iteration the disparity map is considerably improved. In practice three to five iterations ($3 \leq i \leq 5$) suffice more often than not, at which point the refinement tends to converge to its final solution.

5. RESULTS

We demonstrate the effectiveness of our method on the left viewpoint of various Middlebury datasets [6]. All quantitative measurements are expressed in dB PSNR compared with the respective scene's ground truth. Black patches in the ground truth disparity maps indicate invalid pixels (missing data) and are therefore not taken into account.

For the Teddy scene the effect is clear in Fig. 9. Without any refinement (D_W^0) the result remains noisy, with a PSNR of 19.40 dB. Furthermore, the left border can not be reliably matched and remains ambiguous, because this information is missing in the right image. One refinement iteration (D_M^1) already increases the quality with 8 dB to 27.53 dB, yet some substantial noise overall and errors in the left border remain. A second iteration (D_M^2, 29.56 dB) resolves these issues for the most part, and adds another 2 dB in PSNR. The third iteration (D_M^3, 29.57 dB) takes care of the last visually noticeable artifacts (e.g. the black erroneous patch in the lower left corner) and delineates the objects' edges slightly better, but quantitatively its effect is already negligible. Performing any

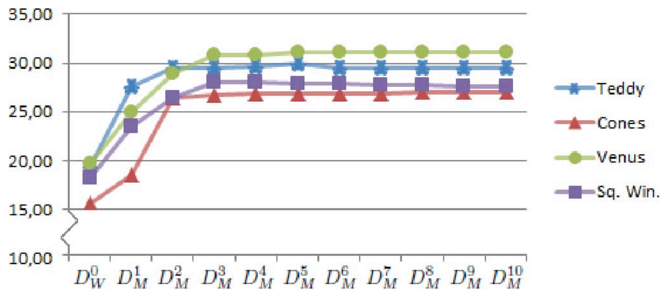

Fig. 8: From this plot of the PSNR measurements of Table 2, it is clear that the iterative refinement quickly reaches its peak quality level and then stabilizes.

more iterations (e.g. D_M^5 and D_M^{10}) barely has any effect at all, and the algorithm stabilizes on a final solution. One obvious erroneous patch remains next to the pink teddy's right ear. However, we postulate that this is due to the limited accuracy of the color consistency check that determines the local support windows (Eq. 2), rather than a limitation of the refinement as a whole.

The Cones scene of Fig. 10 is another challenging dataset that our iterative refinement is able to handle very well. Without refinement (D_W^0, 15.62 dB) the disparity map naturally remains noisy, and one refinement iteration (D_M^1, 18.55 dB) is not able to improve the quality satisfactorily. A considerable amount of noise and errors remain, e.g. on the white cone in the background, on the little white box in the foreground, and the left border. A second iteration is required to increase the quality with nearly 8 dB to 26.44 dB. The PSNR continues to slowly rise hereafter, until by the ninth iteration (D_M^9, 26.97 dB) even the mismatches on the wooden framework in the background are fully resolved. All cones are well discernible. Note in particular the green cone with blue base in front of the red cone. The cones in the left border however disappear, because this information is again not available in the right image.

Venus in Fig. 11 consists of three to four slanted planes with large homogeneously textured regions interspersed with rapidly changing fine – but similar – detail that may easily throw off most local window-based cost aggregation. After five iterations (D_M^5, 31.06 dB) however, the algorithm succeeds to comprehend the slanting of the planes, although the result will always lack the smooth gradual change in grayscale luminance of its ground truth (D_{GT}).

A great strength of our iterative refinement is that it can be applied to any local stereo matching algorithm, as long as the initial disparity map is of sufficient quality. To demonstrate this in the extreme case, we applied it to a disparity map that was computed using a conventional 17×17 square cost aggregation window. As shown in Fig. 12, the result improves dramatically with nearly 10 dB: from 18.21 dB for D_W^0 to 28.10 dB for D_M^3 after only three iterations. However, all artifacts from a naive stereo matching algorithm cannot be eliminated, not even by applying many more iterations.

Module	GT 640		GTX Titan	
	ms	%	ms	%
Local Support Windows	2.905	1.08	0.510	1.16
Per-Pixel Matching Cost	4.684	1.74	0.748	1.70
Cost Aggregation	213.141	79.30	36.716	83.20
Disparity Cross-check	0.100	0.04	0.014	0.03
Bitwise Fast Voting	47.693	17.74	6.100	13.82
Invalid Disparity Handling	0.067	0.03	0.010	0.02
Median Filter	0.193	0.07	0.031	0.07
Total	268.783	100.00	44.129	100.00

Table 1: Absolute and percentage-wise execution time measurements on an NVIDIA GeForce GT 640 and GTX Titan.

All PSNR measurements are listed in Table 2. From their plot in Fig. 8, it is clear that the iterative refinement reaches its peak quality level after no more than three to five iterations, after which it remains stable.

With regard to execution time, Table 1 lists the measurements to compute the left disparity map of the 450×375 resolution Teddy scene, for a disparity range of $[d_{min}, d_{max}] = [12, 53]$ (42 disparities). To complete the pipeline up to and including one refinement iteration (i.e. to compute D_M^1), our algorithm takes 44 ms on an NVIDIA GeForce GTX TITAN. Of these 44 ms, about 38 ms (or 86%) is taken by the initial disparity estimation, whereas one refinement iteration only requires about 6 ms (or 14%). Adding four more iterations of the refinement totals $44 + 4 \times 6 = 68$ ms, thus still providing a real-time solution at about 14 FPS.

6. CONCLUSION

We proposed a novel iterative refinement process to apply to any stereo matching algorithm. It performs in real-time and increases the quality of a disparity map with several dB PSNR.

Its overall success is in large part attributable to the repeated interaction between four rigorously defined modules, and especially between the disparity cross-check and the bitwise fast voting. Starting from a seed disparity map, the bitwise fast voting smooths out its disparities over patches of similar color, which are assumed to belong to the same surface and therefore should possess the same disparity (or depth in the scene). The disparity cross-check subsequently removes again all disparities that were incorrectly estimated. In between this, the invalid disparity handling helps to fill in invalid pixels that the bitwise fast voting cannot reach, and the median filter removes speckle noise. It would be expected that an indefinite repetition would eventually have a detrimental effect on the quality of the disparity map. However, we observed that the interaction between the four modules prevents this from happening and instead the process tends to converge to a final solution.

Currently, we consider the weakest link to be the way the

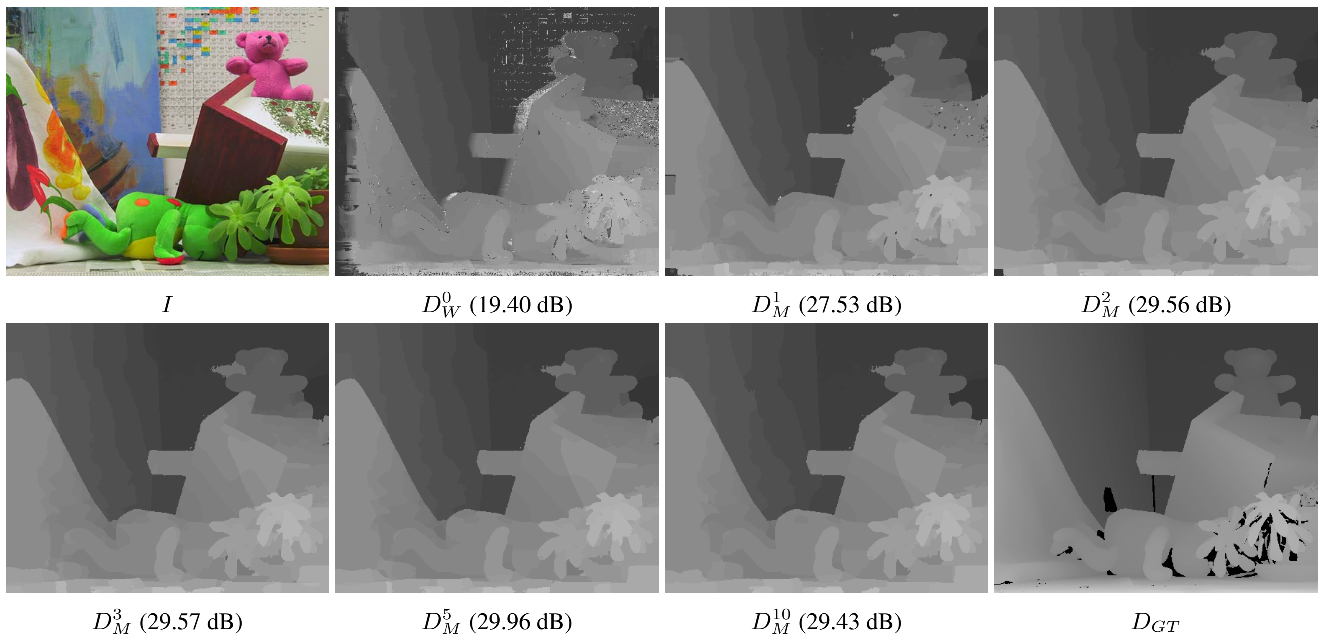

Fig. 9: Teddy: (I) left image, (D_W^0) initial disparity map, (D_M^1) – (D_M^{10}) 1 to 10 refinement iterations, (D_{GT}) ground truth.

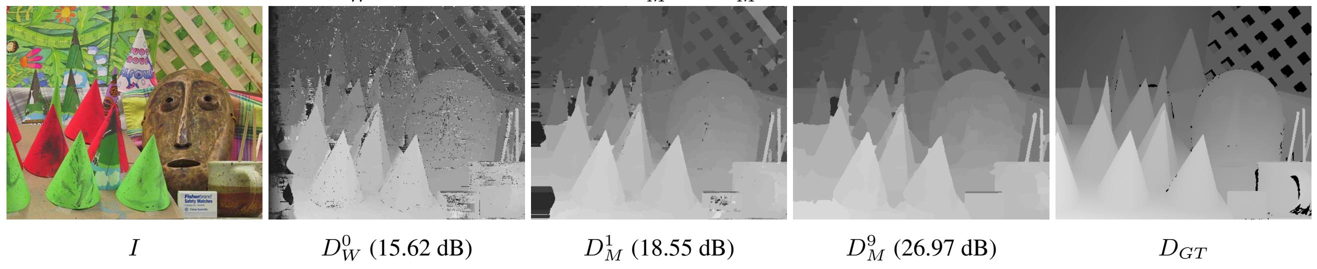

Fig. 10: Cones: (I) left image, (D_W^0) initial disparity map, (D_M^1) – (D_M^9) 1 to 9 refinement iterations, (D_{GT}) ground truth.

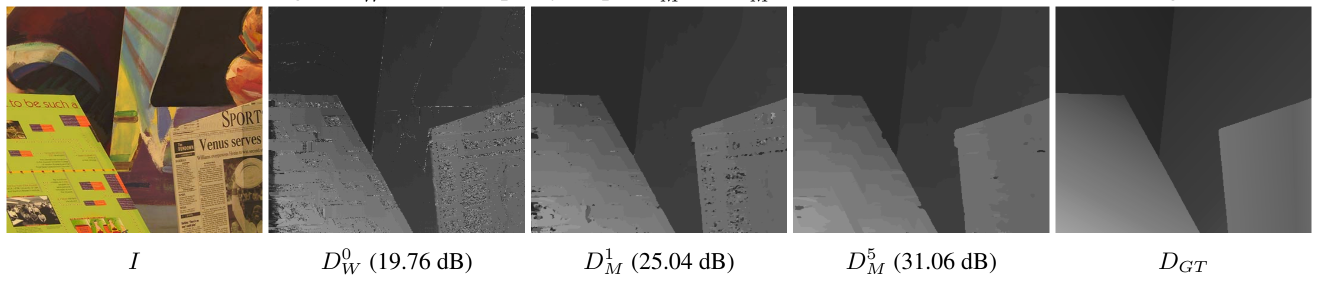

Fig. 11: Venus: (I) left image, (D_W^0) initial disparity map, (D_M^1) – (D_M^5) 1 to 5 refinement iterations, (D_{GT}) ground truth.

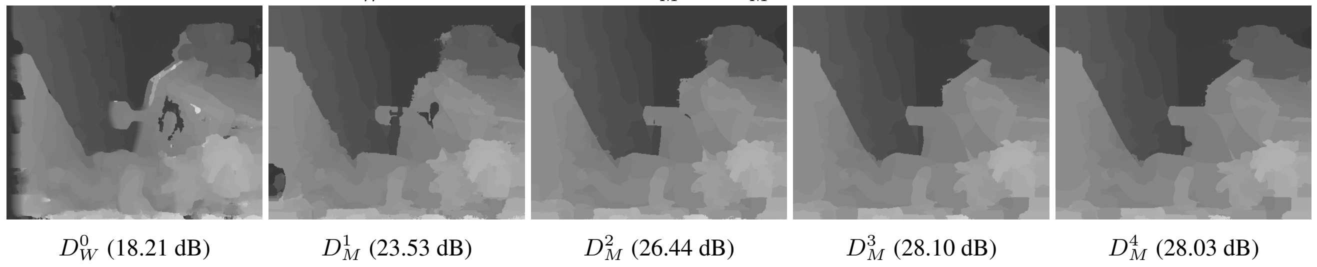

Fig. 12: Teddy: (D_W^0) initial disparity map computed using a conventional 17×17 square window, and (D_M^1) – (D_M^4) subsequently refined using our iterative refinement. Compare with Fig. 9.

Dataset	D_W^0	D_M^1	D_M^2	D_M^3	D_M^4	D_M^5	D_M^6	D_M^7	D_M^8	D_M^9	D_M^{10}
Teddy (Fig. 9)	**19.40**	**27.53**	**29.56**	**29.57**	29.72	**29.96**	29.54	29.54	29.54	29.45	**29.43**
Cones (Fig. 10)	**15.62**	**18.55**	**26.44**	26.75	26.83	26.87	26.88	26.91	26.95	**26.97**	26.95
Venus (Fig. 11)	**19.76**	**25.04**	28.97	30.83	30.88	**31.06**	31.10	31.08	31.10	31.09	31.08
Sq. Win. (Fig. 12)	**18.21**	**23.53**	**26.44**	**28.10**	**28.03**	27.94	27.83	27.74	27.69	27.65	27.62

Table 2: PSNR measurements in dB, for 1 (D_M^1) to 10 (D_M^{10}) iterations of our iterative refinement process from section 4 on the left disparity map of various Middlebury datasets. The initial disparity map (D_W^0) has been computed with our stereo matching algorithm from section 3. The bottom entry (Sq. Win.) is an exception, where the initial disparity map of the Teddy dataset was computed using a conventional 17×17 square window, and subsequently refined using our iterative refinement. (Boldfaced numbers are referenced in the text and figures.)

local support windows are determined. The pixel-wise color consistency check in section 2 is rather rudimentary. Relying on color-based image segmentation to more precisely define the local support windows has the potential to increase the matching quality considerably.

References

[1] J. Davis, D. Nehab, R. Ramamoorthi, and S. Rusinkiewicz. Spacetime stereo: a unifying framework for depth from triangulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27 (2):296–302, February 2005.

[2] Maarten Dumont, Patrik Goorts, Steven Maesen, Philippe Bekaert, and Gauthier Lafruit. Real-time Local Stereo Matching using Edge Sensitive Adaptive Windows. In *Proceedings of the 11th International Conference on Signal Processing and Multimedia Applications*, SIGMAP 2014, pages 117–126, Vienna, Austria, August 2014.

[3] Jiangbo Lu, Sammy Rogmans, Gauthier Lafruit, and Francky Catthoor. High-speed dense stereo via directional center-biased support windows on programmable graphics hardware. In *Proceedings of 3DTV-CON: The True Vision Capture, Transmission and Display of 3D Video*, Kos, Greece, May 2007.

[4] Nicolas Papadakis and Vicent Caselles. Multi-label depth estimation for graph cuts stereo problems. *Journal of Mathematical Imaging and Vision*, 38(1):70–82, 2010.

[5] Christian Richardt, Douglas Orr, Ian Davies, Antonio Criminisi, and Neil A. Dodgson. Real-time spatiotemporal stereo matching using the dual-cross-bilateral grid. In *Computer Vision ECCV 2010*, volume 6313 of *Lecture Notes in Computer Science*, pages 510–523. Springer Berlin Heidelberg, 2010.

[6] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1–3):7–42, 2002.

[7] Olga Veksler. Fast variable window for stereo correspondence using integral images. In *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'03, pages 556–561, Madison, Wisconsin, June 2003.

[8] Liang Wang, Miao Liao, Minglun Gong, Ruigang Yang, and David Nister. High-quality real-time stereo using adaptive cost aggregation and dynamic programming. In *Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission*, 3DPVT '06, pages 798–805, Washington, DC, USA, 2006.

[9] Qingxiong Yang, Liang Wang, Ruigang Yang, Shengnan Wang, Miao Liao, and David Nister. Real-time global stereo matching using hierarchical belief propagation. In *Proceedings of the British Machine Vision Conference*, volume 6, pages 989–998, 2006.

[10] Kuk-Jin Yoon and In-So Kweon. Locally adaptive support-weight approach for visual correspondence search. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR '05, pages 924–931, Washington, DC, USA, 2005.

[11] Ke Zhang, Jiangbo Lu, and Gauthier Lafruit. Cross-based local stereo matching using orthogonal integral images. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(7):1073–1079, 2009.

[12] Ke Zhang, Jiangbo Lu, Qiong Yang, Gauthier Lafruit, Rudy Lauwereins, and Luc Van Gool. Real-time and accurate stereo: A scalable approach with bitwise fast voting on cuda. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(7):867–878, July 2011.

[13] C. Lawrence Zitnick and SingBing Kang. Stereo for image-based rendering using image over-segmentation. *International Journal of Computer Vision*, 75(1):49–65, Oct 2007.