

Stereo Disparity through Cost Aggregation with Guided Filter

Pauline Tan, Pascal Monasse

▶ **To cite this version:**

Pauline Tan, Pascal Monasse. Stereo Disparity through Cost Aggregation with Guided Filter. Image Processing On Line, 2014, pp.252-275. 10.5201/ipol.2014.78 . hal-01086731

HAL Id: hal-01086731

https://enpc.hal.science/hal-01086731

Submitted on 24 Nov 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

2014/07/01 v0.5 IPOL article class

Published in Image Processing On Line on 2014–10–23.
Submitted on 2013–03–01, accepted on 2014–06–12.
ISSN 2105–1232 © 2014 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
http://dx.doi.org/10.5201/ipol.2014.78

Stereo Disparity through Cost Aggregation with Guided Filter

Pauline Tan[1], Pascal Monasse[2]

[1] CMLA, ENS Cachan, France (pauline.tan@ens-cachan.fr)
[2] Université Paris-Est, LIGM (UMR CNRS 8049), ENPC, F-77455 Marne-la-Vallée (monasse@imagine.enpc.fr)

Abstract

Estimating the depth, or equivalently the disparity, of a stereo scene is a challenging problem in computer vision. The method proposed by Rhemann et al. in 2011 is based on a filtering of the *cost volume*, which gives for each pixel and for each hypothesized disparity a cost derived from pixel-by-pixel comparison. The filtering is performed by the *guided filter* proposed by He et al. in 2010. It computes a weighted local average of the costs. The weights are such that similar pixels tend to have similar costs. Eventually, a *winner-take-all* strategy selects the disparity with the minimal cost for each pixel. Non-consistent labels according to left-right consistency are rejected; a densification step can then be launched to fill the disparity map. The method can be used to solve other labeling problems (optical flow, segmentation) but this article focuses on the stereo matching problem.

Source Code

A software written in C++ is available on the IPOL web page of this article[1], which is the code used in the online demo. This gives similar results to the original authors' Matlab implementation[2]. The program needs several parameters (see Section 4 for more detailed explanations). By default they are tuned as suggested in the original article, but one can adapt them to get better results.

Supplementary Material

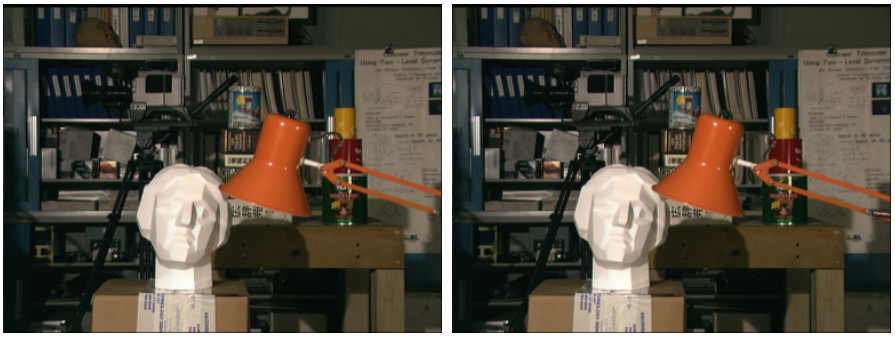
In the demo, an optional rectification step can be launched before running the algorithm. The source code for this preprocessing step (not reviewed) can be found at the IPOL web page of this article[3].

Keywords: stereo-matching; cost volume; guided filter

[1] http://dx.doi.org/10.5201/ipol.2014.78
[2] https://www.ims.tuwien.ac.at/publications/tuw-202088
[3] http://dx.doi.org/10.5201/ipol.2014.78

1 Introduction

Stereo matching algorithms aim at estimating the depth of a scene given two photographs taken from different points of view (Figure 1). Depth estimation is done by estimating the *disparity* of each pixel, namely its apparent displacement from the first (reference) image to the second (target) image. This disparity permits to recover the 3D position of the point that was photographed in this pixel, if the stereo setup has been previously calibrated.



(a) Left (reference) image (b) Right (target) image

Figure 1: Tsukuba stereo pair

We assume that the left image is the reference image. Let M be a 3D-point. We denote by O_L (resp. O_R) the principal point of the left (resp. right) camera, that is the orthogonal projection of the optical center on the image plane. The point p (resp. q) denotes the image of M in the left image (resp. in the right image). Then, the disparity of M is given by $d(M) = \overrightarrow{O_Rq} - \overrightarrow{O_Lp}$. Assume the camera makes a so-called fronto-parallel move, meaning that the focal plane is the same for all positions of the camera and that its motion is parallel to the x-axis of the camera CCD array. This is the case for instance in the Middlebury benchmark dataset[4]. Then, the disparity is scalar and its absolute value is inversely proportional to the distance from the observer. More precisely, if the camera moves from the left to the right, any point has a negative disparity (see Figure 2). Hence, the lower the (signed) disparity, the lower the depth.

In more general cases, the situation is slightly different. After stereo-rectification, some information is lost, such as the position of the camera principal points. The coordinates of pixels in the images are then not given with respect to the optical center, but to an arbitrary point (denoted O'_L and O'_R). Without any supplementary information, the disparity of a point M is then given by $d'(M) = \overrightarrow{O'_Rq} - \overrightarrow{O'_Lp}$. We see that we can go back to the previous case by writing $d'(M) = \overrightarrow{O'_RO_R} + \overrightarrow{O_Rq} - \overrightarrow{O_Lp} - \overrightarrow{O'_LO_L} = d(M) + C$ where C does not depend on the point M. Then, to recover the depth from the disparity of any point, a constant has to be added to the computed disparity, but lower disparity still means lower depth.

If the camera motion is right to left, the situation is inverted and lower disparity means higher depth. This is important for the occlusion filling process, which in a first step fills in the non informed pixels with a disparity corresponding to lower depth.

[4] http://vision.middlebury.edu/stereo/

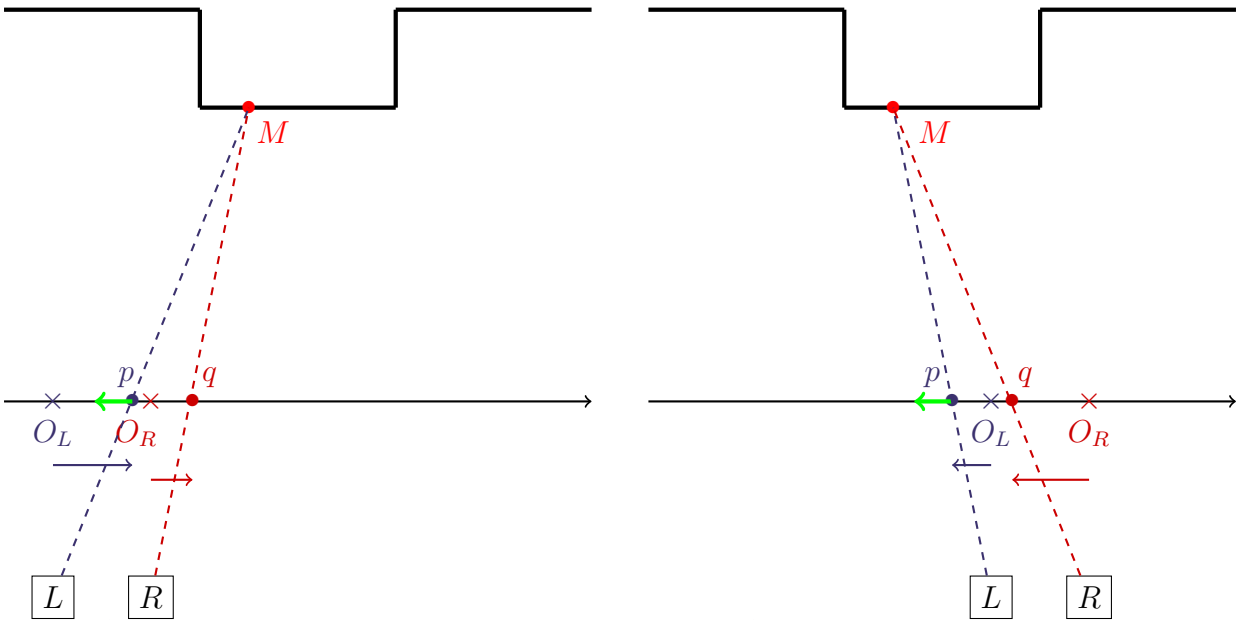


Figure 2: Disparity when the camera moves from the left to the right. The disparity of the pixel M is given by subtracting the blue vector from the red vector. The disparity (in green) is then always negative.

2 Algorithm Description

Let I_L and I_R be respectively the left and right color images of a stereo pair, of size $M \times N$. Then, I_L^r, I_L^g and I_L^b denote the color channels of the left image (I_R^r, I_R^g and I_R^b for the right image). We proceed to the gray-level conversion of the images:

$$\tilde{I}_L = 0.299 \cdot I_L^r + 0.587 \cdot I_L^g + 0.0721 \cdot I_L^b \qquad \text{and} \qquad \tilde{I}_R = 0.299 \cdot I_R^r + 0.587 \cdot I_R^g + 0.0721 \cdot I_R^b. \quad (1)$$

We assume the disparity range $[d_{\min}, d_{\max}]$ to be known.

2.1 Cost Volume

Pixel-to-pixel comparison is done by involving the truncated absolute difference of the color and the x-derivative of the grayscale images. These two terms are balanced with a parameter α.

1. Color comparison: for each pixel i in the left image and each pixel j in the right image, the color penalty is obtained by averaging the AD (Absolute Difference) on each color channel:

$$N_{\text{color}}(i,j) := \frac{1}{3}\Big(\big|I_L^r(i) - I_R^r(j)\big| + \big|I_L^g(i) - I_R^g(j)\big| + \big|I_L^b(i) - I_R^b(j)\big|\Big) \quad (2)$$

   which is then thresholded:

$$N_{\text{color}}^{\tau_1}(i,j) := \min\Big(N_{\text{color}}(i,j), \tau_1\Big). \quad (3)$$

2. x-derivative comparison: the discrete derivative along the x-axis of the gray-level image $\tilde{I}_L$ (resp. $\tilde{I}_R$), denoted $\nabla_x \tilde{I}_L$ (resp. $\nabla_x \tilde{I}_R$), is computed as follows[5]

$$\forall\, i = (i_x, i_y), \quad \nabla_x \tilde{I}_L(i) := \frac{\tilde{I}_L(i_x + 1, i_y) - \tilde{I}_L(i_x - 1, i_y)}{2}. \quad (4)$$

[5]In (4), pixel intensities outside the image are considered to be the same as at the closest pixel inside the image.

The AD is then computed

$$N_{\text{gradient}}(i,j) := \left|\nabla_x \tilde{I}_L(i) - \nabla_x \tilde{I}_R(j)\right| \tag{5}$$

and thresholded

$$N^{\tau_2}_{\text{gradient}}(i,j) := \min\Big(N_{\text{gradient}}(i,j), \tau_2\Big). \tag{6}$$

3. Eventually, the color and gradient terms are balanced:

$$N(i,j) := (1-\alpha)\cdot N^{\tau_1}_{\text{color}}(i,j) + \alpha \cdot N^{\tau_2}_{\text{gradient}}(i,j). \tag{7}$$

The *cost volume* C is then defined on a sampling integer grid of the volume $[0, M-1] \times [0, N-1] \times [d_{\min}, d_{\max}]$, such that, for any disparity value $d \in [d_{\min}, d_{\max}]$:

$$C(i,d) := N(i, i+d). \tag{8}$$

This assumes that $i + d$ is in the domain of the right image (we note by a slight abuse of notation $i + d := (i_x + d, i_y)$ if $i = (i_x, i_y)$). If it is not the case, we set $C(i,d) := (1-\alpha)\cdot\tau_1 + \alpha\cdot\tau_2$ which is the maximal possible cost.

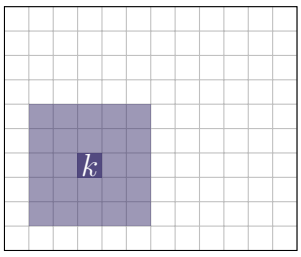
2.2 Filtering with Guided Filter

The pixel-to-pixel comparison alone is too sensitive to noise. So, each slice $p(i) = C(i,d)$ has to be filtered. The guided filter [4] is chosen since it is designed to preserve edges, by using the left image I_L as *guidance image*.

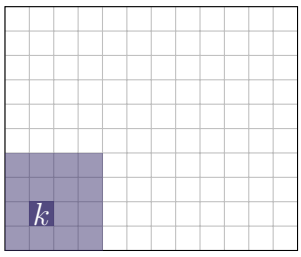
Grayscale guidance image. Let I be the guidance (grayscale) image and q be the output of the filter. The key assumption of the guided filter is that it is a local linear model between I and q. For each square window ω_k with radius r_{GF} centered at k (by definition, a square with integer radius r is a square with side length $2 \times r + 1$ and $|\omega_k|$ denotes the number of pixels in ω_k), a k-dependent filtered image q_k is defined to be a linear transform of I in ω_k

$$\exists\, a_k, b_k, \forall i \in \omega_k, \qquad q_k(i) = a_k \cdot I(i) + b_k. \tag{9}$$

If k is too close to the image boundaries, then we only take into account the part of the window that remains in the domain of the image (see Figure 3).



(a) Window of radius $r_{\text{GF}} = 2$ centered at $k = (3,6)$. $|\omega_k| = 25$.



(b) Window of radius $r_{\text{GF}} = 2$ centered at $k = (1,8)$. $|\omega_k| = 16$.

Figure 3: Examples of square windows.

In the window ω_k, since a_k and b_k are constant, we can write

$$\nabla q_k = a_k \nabla I. \tag{10}$$

That is, strong edges in I imply strong edges in q_k. Hence, the guided filter preserves the edges of the guidance image. To determine coefficients a_k and b_k, the L^2-norm between the filter input p and the filter output q_k is minimized. Hence, an energy-based formulation of this problem is:

$$\arg\min_{a_k,b_k} \left\{ E(a_k,b_k) := \sum_{i\in\omega_k} \Big(a_k \cdot I(i) + b_k - p(i) \Big)^2 + \varepsilon |\omega_k| a_k^2 \right\}. \tag{11}$$

The first term is a data term and the second term is a regularization term preventing a_k from being too large. The parameters a_k and b_k are then obtained by solving the Euler equation ($\nabla E = 0$):

$$a_k = \frac{\frac{1}{|\omega_k|}\sum_{i\in\omega_k} I(i)p(i) - \mu_k \overline{p}_k}{\sigma_k^2 + \varepsilon} \tag{12}$$

$$b_k = \overline{p}_k - a_k \mu_k \tag{13}$$

where μ_k and σ_k^2 are the mean and variance of I in the window ω_k, and $\overline{p}_k$ is the mean of p in ω_k:

$$\mu_k := \frac{1}{|\omega_k|}\sum_{i\in\omega_k} I(i), \qquad \sigma_k^2 := \frac{1}{|\omega_k|}\sum_{i\in\omega_k} I(i)^2 - \mu_k^2 \qquad \text{and} \qquad \overline{p}_k := \frac{1}{|\omega_k|}\sum_{i\in\omega_k} p(i). \tag{14}$$

The coefficients computed above depend on the window ω_k. However, any pixel i is involved in several windows ω_k, which leads to many values of coefficients a_k and b_k. A simple way to solve this problem is to take the average of all these values:

$$q(i) := \frac{1}{|\omega_i|}\sum_{k|i\in\omega_k} q_k(i) = \frac{1}{|\omega_i|}\sum_{k|i\in\omega_k} \big(a_k \cdot I(i) + b_k \big) = \overline{a}_i \cdot I(i) + \overline{b}_i \tag{15}$$

with $\overline{a}_i$ (resp. $\overline{b}_i$) the mean of a_k (resp. b_k) in the window ω_i, namely

$$\overline{a}_i := \frac{1}{|\omega_i|}\sum_{k|i\in\omega_k} a_k \qquad \text{and} \qquad \overline{b}_i := \frac{1}{|\omega_i|}\sum_{k|i\in\omega_k} b_k. \tag{16}$$

Notice that $i \in \omega_k \Leftrightarrow k \in \omega_i$, which explains the normalization factor $1/|\omega_i|$.

The filter output q is then computed following four steps:

1. Computing the images μ_k, $\overline{p}_k$, $\frac{1}{|\omega_k|}\sum_{i\in\omega_k} I(i)p(i)$, and σ_k^2;
2. Computing the images a_k and b_k with expressions (12) and (13);
3. Computing the images $\overline{a}_i$ and $\overline{b}_i$ by (16);
4. Computing q thanks to (15).

Note that the first three steps use a *box filter* (mean in a square window). An efficient implementation of the box filter, independent of the radius of the square window, is shown in the next section. The above filter may be better understood when reformulated as a variable kernel convolution W^{GF}:

$$q(i) = \sum_j W_{ij}^{\mathrm{GF}} \cdot p(j) \qquad \text{with} \qquad W_{ij}^{\mathrm{GF}} := \sum_{k\in\omega_i\cap\omega_j} \frac{1}{|\omega_i|\cdot|\omega_k|}\left(1 + \frac{\big(I(j)-\mu_k\big)\big(I(i)-\mu_k\big)}{\sigma_k^2+\varepsilon}\right). \tag{17}$$

It is easily observed by this formula that pixels j in the neighborhood of i having similar color have a stronger weight in the resulting value of $q(i)$.

Color guidance images. For color guidance images ($I = (I^r, I^g, I^b)^T$), the linear model can be written as

$$q_k(i) = a_k^T I(i) + b_k. \tag{18}$$

The vectors a_k and b_k are then given by

$$a_k = (\Sigma_k + \varepsilon \mathrm{I}_3)^{-1} \begin{pmatrix} \frac{1}{|\omega_k|}\sum_{i\in\omega_k} I^r(i)p(i) - \mu_k^r \overline{p}_k \\ \frac{1}{|\omega_k|}\sum_{i\in\omega_k} I^g(i)p(i) - \mu_k^g \overline{p}_k \\ \frac{1}{|\omega_k|}\sum_{i\in\omega_k} I^b(i)p(i) - \mu_k^b \overline{p}_k \end{pmatrix} \qquad \text{and} \qquad b_k = \overline{p}_k - a_k^T \begin{pmatrix} \mu_k^r \\ \mu_k^g \\ \mu_k^b \end{pmatrix}, \tag{19}$$

with μ_k^r, μ_k^g, and μ_k^b the mean images of each color channel of the guidance image I:

$$\mu_k^{[1]} := \frac{1}{|\omega_k|}\sum_{i\in\omega_k} I^{[1]}(i) \qquad \text{for } [1] \in \{r, g, b\}, \tag{20}$$

Σ_k is the covariance matrix of the guidance image I in the window ω_k,

$$\Sigma_k := \begin{pmatrix} \sigma_k^{rr} & \sigma_k^{rg} & \sigma_k^{rb} \\ \sigma_k^{rg} & \sigma_k^{gg} & \sigma_k^{gb} \\ \sigma_k^{rb} & \sigma_k^{gb} & \sigma_k^{bb} \end{pmatrix} \text{ where } \sigma_k^{[1][2]} := \frac{1}{|\omega_k|}\sum_{i\in\omega_k} I^{[1]}(i)I^{[2]}(i) - \mu_k^{[1]}\mu_k^{[2]} \text{ for } [1],[2] \in \{r, g, b\} \tag{21}$$

and I_3 is the identity matrix of size 3×3. In this case, the kernel weights are given by

$$W_{ij}^{\mathrm{GF}} = \frac{1}{|\omega_i| \cdot |\omega_j|}\sum_{k\in\omega_i\cap\omega_j} \Big(1 + \big(I(i) - \mu_k\big)^T \big(\Sigma_k + \varepsilon \mathrm{I}_3\big)^{-1} \big(I(j) - \mu_k\big)\Big). \tag{22}$$

In conclusion, each slice of the cost volume $C(\cdot, d)$ is filtered with the guidance image I_L:

$$C^{\mathrm{filt}}(i,d) := \sum_{j\in\mathcal{L}} \frac{1}{|\omega_i| \cdot |\omega_j|}\sum_{k\in\omega_i\cap\omega_j} \Big(1 + \big(I_L(i) - \mu_k\big)^T \big(\Sigma_k + \varepsilon \mathrm{I}_3\big)^{-1} \big(I_L(j) - \mu_k\big)\Big) \cdot C(j,d). \tag{23}$$

The generalization to multi-channel images as guidance, as for example hyper-spectral satellite imagery, would follow the same process with matrices of different size.

2.3 Implementation of the Box Filter [3]

In this section, an implementation of the box filter proposed by Crow [2] is described. The simplest way to compute the mean in a square window of an image I is to sum the value of each pixel of the window, and then divide by the window size. Thus, for a window size of $|\omega_k|$, $|\omega_k| - 1$ sums and one division are needed. Hence, the complexity of the process is linear with the radius of the window.

Another way to do it would be to first compute an integral image S of summed areas, of the same size as I:

$$\forall\, i = (i_x, i_y), \quad S(i) = \sum_{k_x=0}^{i_x}\sum_{k_y=0}^{i_y} I(k_x, k_y). \tag{24}$$

Then a sum in any rectangle $[i_x, j_x] \times [i_y, j_y]$ is given by (see Figure 5)

$$S(j_x, j_y) - S(i_x - 1, j_y) - S(j_x, i_y - 1) + S(i_x - 1, i_y - 1), \tag{25}$$

an eventual negative index removing the involved terms of S in the formula. Hence the computation of the mean only needs one sum, two subtractions and one division; it no longer depends on the radius of the window.



(a) Guidance image (b) Slice $d = -4$ of the cost volume (c) Filtering by guided filter

Figure 4: Filtering by guided filter of a slice of the cost volume for the Tsukuba pair ($\tau_1 = 7$, $\tau_2 = 2$, $\alpha = 0.9$, $\varepsilon = 10^{-4}$, $r_{\mathrm{GF}} = 9$).

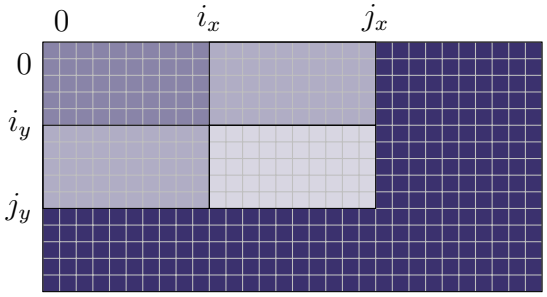
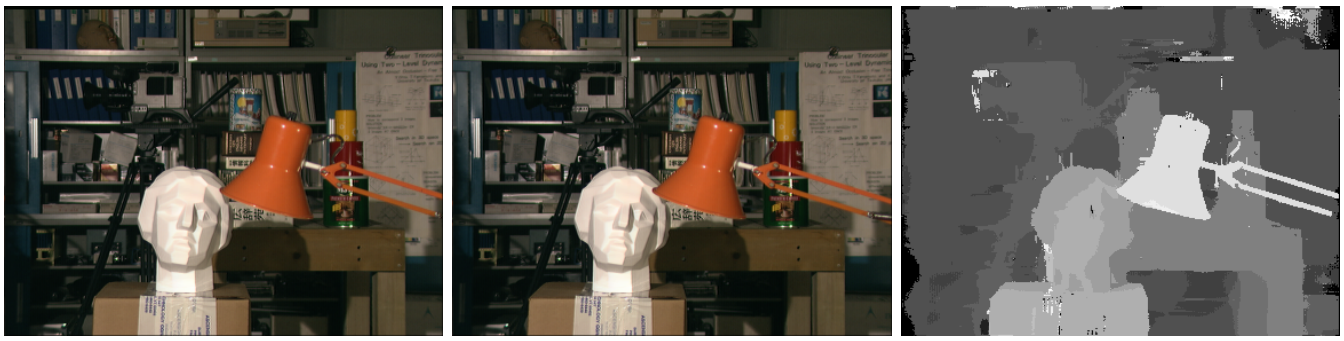


Figure 5: Calculation of a summed area from the integral image.

2.4 Disparity Selection

Once the cost volume is filtered, the disparity of any pixel i is the disparity that minimizes the cost associated to i in the cost volume ("Winner-take-all" strategy):

$$d(i) = \arg \min_{d' \in [d_{\min}, d_{\max}]} \left\{ C^{\mathrm{filt}}(i, d') \right\}. \tag{26}$$



(a) Left image (b) Right image (c) Disparity map

Figure 6: Result of the winner-take-all algorithm on the Tsukuba pair.

2.5 Left-Right Consistency

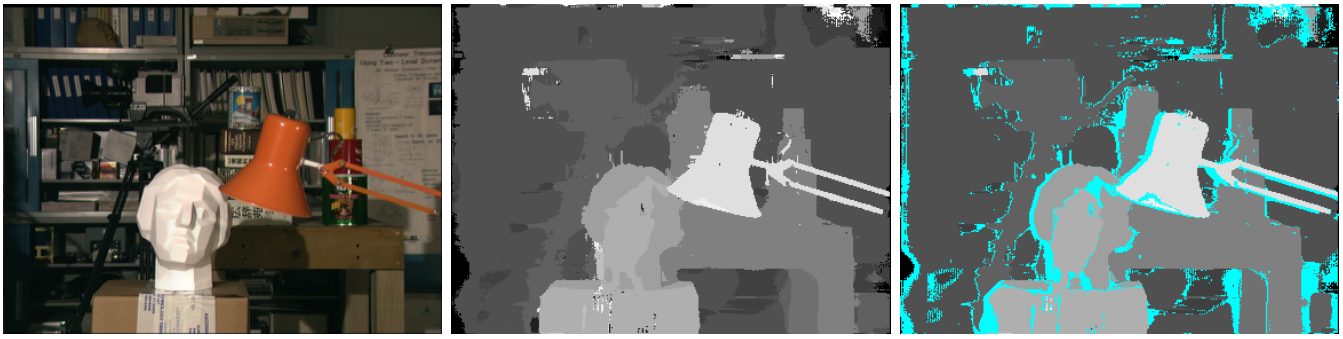
The left-right consistency checks whether the disparity computed from the left image I_L coincides with the disparity computed from the right image I_R. At pixel precision, a disparity is rejected when

$$d(i) \neq -d'\big(i + d(i)\big) \tag{27}$$

where d' denotes the disparity map computed from right to left image. This mask is expected to reject all occluded pixels and some flat region pixels of the scene. Thus rejected pixels are called "occluded pixels" in what follows. To handle subpixel disparities as well, this can be reformulated in the more general constraint:

$$\big|d(i) + d'\big(i + d(i)\big)\big| \geq d_{LR}, \tag{28}$$

with $d_{LR} = 1$. Equation (27) is (28) when $d_{LR} = 0$.



(a) Left image (b) Disparity map (c) Left-right consistency rejections

Figure 7: Result of the left-right consistency check on the Tsukuba pair. Most of the occluded pixels are detected by the left-right consistency mask. So are some pixels on constant disparity regions.

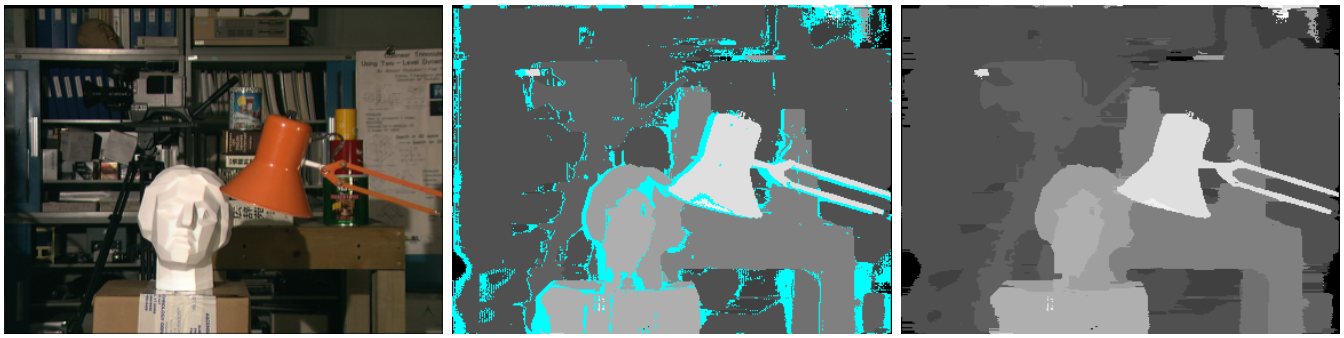
2.6 Densification Process

This step aims at filling the disparity map after the rejection test. Only occluded pixels are affected by this process, non-occluded ones keep their original disparity. It is done in two stages: the basic filling stage and the post-processing stage.

In the first stage a disparity value is given to the pixels rejected by the previous step. Since the mask mainly rejects occluded pixels, it often involves points of the scene that are hidden by other objects. In such cases, the rejected pixels are mainly in a region of discontinuity of the scene, where a closer object is in front of a farther one. Hence a strategy would be to adopt the disparity of the farther object, by exploring the closest nonoccluded pixels (on the same line). We recall that the farther object is given by the higher disparity. Hence, this leads to the following filling-in formula:

$$\forall\, i = (i_x, i_y),\ d_{\text{fill}}(i) := \max \left\{ \begin{array}{l} d\left(i_x + \min\limits_{\substack{j_x \leq 0 \\ (i_x + j_x, i_y) \text{ nonoccluded}}} \{j_x\}, i_y \right), \\ d\left(i_x + \min\limits_{\substack{j_x \geq 0 \\ (i_x + j_x, i_y) \text{ nonoccluded}}} \{j_x\}, i_y \right) \end{array} \right\}. \tag{29}$$

Notice that if i is nonoccluded, $d_{\text{fill}}(i) = d(i)$.



(a) Left image (b) Left-right consistency rejections (c) Simple filling

Figure 8: Result of the filling on the Tsukuba pair. Streak-line artifacts appear.

Simple filling can generate streak-line artifacts. Hence an edge-preserving filter is used to remove them, by smoothing the disparity map. The left image is first filtered by the median filter (applied on 3×3-windows): for $[1] \in \{r, g, b\}$,

$$I^{[1]}_{L,\text{filt}}(i) = \text{median}\Big\{ I^{[1]}_L(i+j) \;\Big|\; j = (j_x, j_y) : j_x, j_y \in \{-1, 0, +1\} \Big\}. \tag{30}$$

Let i be an occluded pixel. Then for each j in the square window ω'_i of radius r_{WMF} centered on i, the bilateral filter weight is computed by

$$W^{\text{BF}}_{ij} := \exp\left(-\frac{\|i-j\|^2}{\sigma_s^2} \right) \cdot \exp\left(-\frac{\|I_{L,\text{filt}}(i) - I_{L,\text{filt}}(j)\|_c^2}{\sigma_c^2} \right), \tag{31}$$

where $\|\cdot\|$ denotes the Euclidean distance and $\|\cdot\|_c$ the Euclidean distance in RGB color space:

$$\|I_{L,\text{filt}}(i) - I_{L,\text{filt}}(j)\|_c^2 = \left|I^r_{L,\text{filt}}(i) - I^r_{L,\text{filt}}(j)\right|^2 + \left|I^g_{L,\text{filt}}(i) - I^g_{L,\text{filt}}(j)\right|^2 + \left|I^b_{L,\text{filt}}(i) - I^b_{L,\text{filt}}(j)\right|^2. \tag{32}$$

This filter gives higher weight to pixels spatially close and of similar color, which are likely to belong to the same object. A disparity cumulative histogram h is then computed, where each value is weighted by the weight (31):

$$\forall\, d \in [d_{\min}, d_{\max}], \quad h(d) = \sum_{j \in \omega'_i | d(j) \le d} W^{\text{BF}}_{ij}. \tag{33}$$

The final disparity of i is then set to be the median value of h:

$$d_{\text{final}}(i) := \min\left\{ d \;\middle|\; h(d) \ge \frac{1}{2} h(d_{\max}) \right\}. \tag{34}$$

3 Algorithm Summary and Implementation

A class `Image` defined in file `image.h` is used throughout. It is a container for an array of pixels that is reference counted: a copy of the image does not reallocate memory, it just adds a reference to the same array in memory (a coding pattern named *shallow copy*). The pixel by pixel operators of addition, subtraction and multiplication, used in the cost-volume filtering, are implemented as



(a) Left image (b) Simple filling (c) Filtering the former by weighted median filter

Figure 9: Result of the post-processing on the Tsukuba pair. The disparity map is smoothed.

overloaded operators, easing readability of the code. Some simple image filters are implemented in file `filters.cpp`. Among them, `boxFilter` performs local averages using summed area tables. The method `gradX` computes the x-derivative of the image.

Algorithm 1 gives the pseudocode of the cost-volume filtering described above. The code is in file `cost_volume.cpp`. Notice that we do not need to store in memory the full cost-volume, only the slice at current disparity d needs to be stored. The cost image p is computed in function `compute_cost`. The filtering itself is in function `filter_cost_volume`. Our code implements the algorithm for color images even for actually grayscale images, which yields the same result, though at higher computation cost.

The consistency check is described in Algorithm 2. The disparity d' from I_R to I_L is computed, using range $[-d_{\max}, -d_{\min}]$. The consistency between d and d' is checked in function `detect_occlusion` of file `occlusion.cpp`. The threshold d_{LR} (0 by default) is used.

The filling process is described in Algorithm 3. The simple filling is performed by using methods `fillMaxX` or `fillMinX` of class `Image`, depending on the camera motion direction (user input parameter). The filling based on bilateral weights (function `fill_occlusion` of file `occlusion.cpp`) is parallelized using openMP. Two functions in file `filters.cpp` are used: `weighted_histo` computes the local histogram of disparities weighted by bilateral weights, while `median_histo` computes the median of the histogram.

4 Method Parameters

Several parameters should be tuned to use this method. Table 1 recalls all of them. Default values of these parameters suggested in [8] are also given. Default values of τ_1, τ_2, ε and σ_c are adapted to 8-bit images with intensity range in $[0, 255]$ (most frequent case), but should be changed for higher bit depths.

Results obtained with different values of the parameters are then shown in order to see their influence. The experiments were done on the Tsukuba pair. Unless explicitly stated otherwise, only one parameter is changed at a time. The other parameters are tuned by default. Errors are also shown (absolute disparity error > 0.5) using a mask generated by Middlebury based on ground truth measurements (Figure 10). Note that this mask does not take into account the image borders. These results were also submitted to the Middlebury site in order to automatically get the percentage of bad pixels (with a wrong disparity). In the following figures (Figure 11 to Figure 18), the first row shows the disparity map obtained by our implementation. The second row shows the disparity map

Algorithm 1: Cost-volume guided filtering algorithm

Input: Color images I_L and I_R, disparity range $[d_{\min}, d_{\max}]$.
Output: Disparity map $d(.)$

Compute the images μ_k^r, μ_k^g, μ_k^b (20), and Σ_k (21).
Cost(.)$\leftarrow \infty$
foreach $d \in [d_{\min}, d_{\max}]$ **do**
    Building the slice d of cost volume:
    foreach *pixel i in the left image* **do**
        Compute the color term (3) and the gradient term (6) for disparity d.
        Compute the cost associated to pixel i for disparity d (7).
        Set the cost value in $p(i) = C(i, d)$ (8).
    Cost-volume slice filtering by guided filter:
    Compute the image $\bar{p}_k$ of p (14).
    Compute the images $\frac{1}{|\omega_k|}\sum_{i\in\omega_k} I_L^r(i)p(i)$, $\frac{1}{|\omega_k|}\sum_{i\in\omega_k} I_L^b(i)p(i)$, and $\frac{1}{|\omega_k|}\sum_{i\in\omega_k} I_L^b(i)p(i)$.
    Compute the images of coefficients a_k et b_k (19).
    Compute the images $\bar{a}_i$ and $\bar{b}_i$ (16).
    Compute the output of the filter $q = C^{\text{filt}}(\cdot, d)$ (15).
    Disparity selection:
    foreach *pixel i in the left image* **do**
        if *Cost(i)*$> q(i)$ **then**
            Cost(i)$\leftarrow q(i)$
            $d(i) \leftarrow d$ (26).

Algorithm 2: Left-right consistency check

Compute the disparity map d' from I_R to I_L.
foreach *pixel i in the left image* **do**
    If $|d(i) + d'(i + d(i))| > d_{LR}$, label the pixel i as occluded.

Algorithm 3: Disparity map filling

Simple filling:
foreach *occluded pixel i* **do**
    Find the two nearest neighbors (on the same line) that are not occluded.
    Adopt the higher (assuming left to right camera motion) disparity (29).
Post-processing:
Filter the left image by median filter (30).
foreach *occluded pixel i* **do**
    Compute the bilateral filter weights on a neighborhood of i (31).
    Use these weights to compute a weighted median filter on the neighborhood of i (33), (34).

Parameter	Suggested value	Description
r_{GF}	9	Guided filter radius.
α	0.9	Parameter in $[0, 1]$ that balances the color and the gradient contribution. The bigger this parameter, the more important the contribution of the gradient term.
τ_1	7	Threshold used for the color penalty AD distance.
τ_2	2	Threshold used for the gradient penalty AD distance.
ε	$255^2 \cdot 10^{-4}$	Guided filter regularization parameter.
d_{LR}	0	Left-right disparity difference tolerance
r_{WMF}	9	Weighted median filter radius.
σ_{s}	9	Weight of the spatial term in the bilateral filter.
σ_{c}	$255 \cdot 10^{-1}$	Weight of the color term in the bilateral filter.

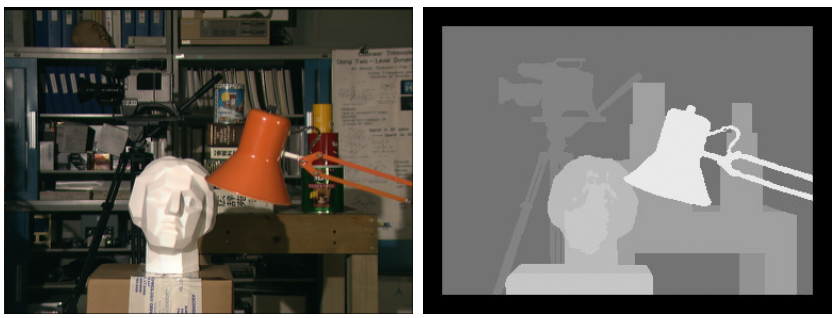
Table 1: Method parameters.

with the mask (in pink) generated by Middlebury.

We first tested the influence of the parameters of the computation of the cost volume and its filtering by the guided filter. Figure 11 shows the results after the whole process (including filling and post-processing) when the size of the filter r_{GF} is tuned. A larger filter size gives smoother results often visually more satisfying. However, the disparity map can be too smooth and errors generated on the boundaries of some objects. Indeed, when we look at the percentage of bad pixels on the boundaries of objects, we get 19.7% bad pixels for $r_{\mathrm{GF}} = 19$ while there are only 16.4% bad pixels for the default parameters. Nevertheless, the results remain globally better with a large filter size. Next, the influence of the parameter α was studied (see Figure 12). The larger this parameter, the heavier the gradient term weight. Results are better for large values of α. In other terms, the gradient comparison seems much more reliable than the color comparison. This is due to the fact that the scene is textured. Note that the disparity estimation is better on the lamp when α is small, because the lamp has no texture.

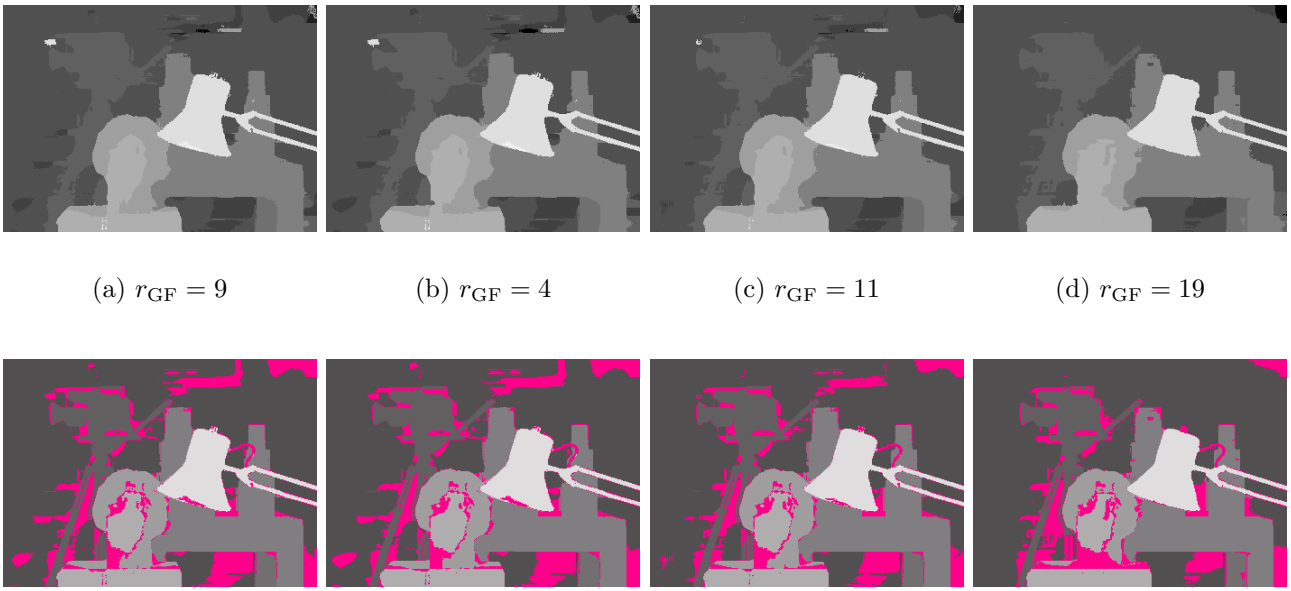
Figure 13 shows the error percentages for four of the Middlebury images with respect to r_{GF} and α, the two main parameters and the ones that can be tuned in the online demo. This shows that the default parameters are well adapted to get optimal results in the Middlebury benchmark.

The other parameters needed for the computation of the cost volume are the thresholds τ_1 and τ_2.



(a) Left image (b) Ground truth

Figure 10: Ground truth used to evaluate the results.



(a) $r_{\mathrm{GF}} = 9$ (b) $r_{\mathrm{GF}} = 4$ (c) $r_{\mathrm{GF}} = 11$ (d) $r_{\mathrm{GF}} = 19$

(e) $r_{\mathrm{GF}} = 9$ (f) $r_{\mathrm{GF}} = 4$ (g) $r_{\mathrm{GF}} = 11$ (h) $r_{\mathrm{GF}} = 19$

Figure 11: Influence of the guided filter size. (a)+(e) Default parameters ($r_{\mathrm{GF}} = 9$). 12.9% are bad pixels. (b)+(f) $r_{\mathrm{GF}} = 4$, bad pixels: 15.9%. (c)+(g) $r_{\mathrm{GF}} = 11$, bad pixels: 11.8%. (d)+(h) $r_{\mathrm{GF}} = 19$, bad pixels: 11.7%. When the filter size grows, the disparity map is cleaner (wrong disparities are removed, e.g. in the background), but errors appear on the object boundaries (e.g. on the space between the lamp and the table), due to an oversmooth map.

By default, they are chosen very low, which limits the influence of a large difference (of color or gradient). Figure 14 shows that the results are more accurate when these thresholds are larger. The results degrade when the thresholds are too large, though.

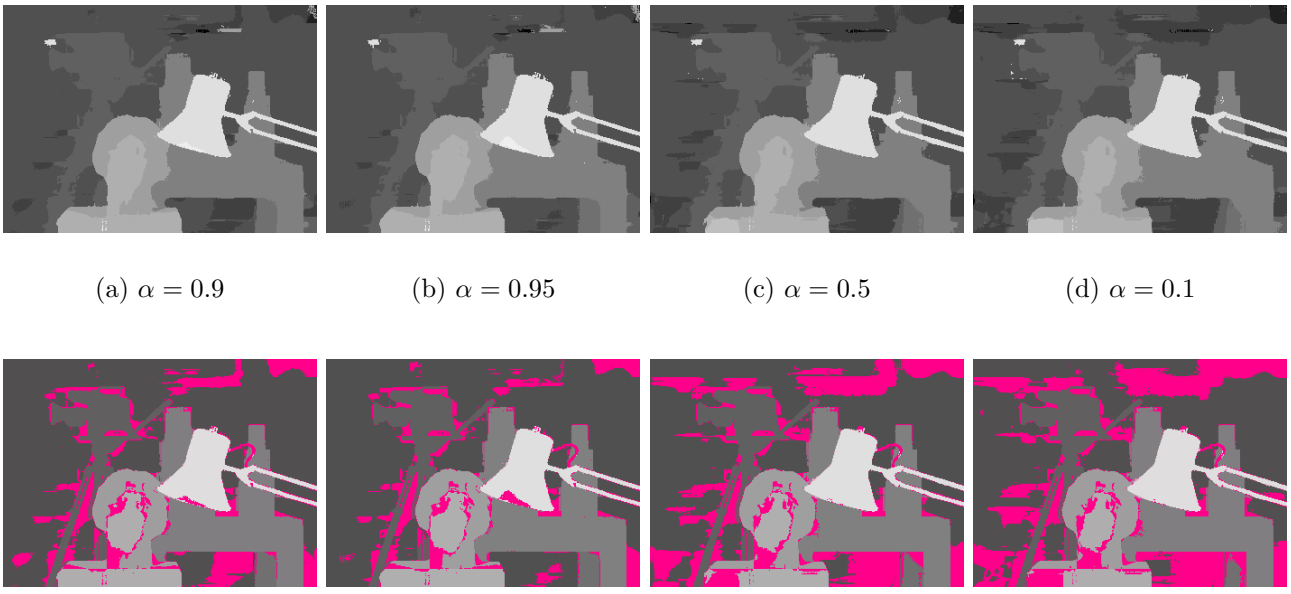
The last parameter involved in the filtering of the cost volume is the regularization parameter ε. The larger this parameter, the smoother the disparity map. Figure 15 shows that even though the number of bad pixels is lower for larger values of ε, the results are not visually satisfactory since the boundaries of many objects seem loose while ε grows.

The influence of the parameters involved in the filling process was also tested. We started with the size of the weighted median filter. Figure 16 shows that this parameter is not a key parameter, since the results do not significantly change when we tune it. The disparity estimation is slightly enhanced when the filter size grows. However, the larger the filter, the slower the computation.

Then, the parameters σ_{c} (Figure 17) and σ_{s} (Figure 18) are tuned, which control the importance of the color distance and the spatial distance in the bilateral filter. The experiments do not uncover any significant changes. One can notice that when these parameters are too large, the results are worse, because the bilateral filter is then not discriminatory enough. On the contrary, when they are too small, the filter rejects too many pixels.

By tuning properly the parameters to get better results, we managed to reach 9.01% bad pixels (and 8.44% in nonoccluded regions). Figure 19 shows the disparity map obtained with the following parameter values: $r_{\mathrm{GF}} = 11$, $\alpha = 0.95$, $\tau_1 = 20$, $\tau_2 = 10$, $\varepsilon = 255^2 \cdot 10^{-4}$, $r_{\mathrm{WMF}} = 29$, $\sigma_{\mathrm{c}} = 255 \cdot 10^{-1}$, $\sigma_{\mathrm{s}} = 15$. However, one should keep in mind that the optimal values of the parameters depend on the scene.

We can compare the result before and after the densification process. Figure 20 shows the disparity map obtained with and without this step. The densification cleans the disparity map, by



(a) $\alpha = 0.9$ (b) $\alpha = 0.95$ (c) $\alpha = 0.5$ (d) $\alpha = 0.1$

(e) $\alpha = 0.9$ (f) $\alpha = 0.95$ (g) $\alpha = 0.5$ (h) $\alpha = 0.1$

Figure 12: Influence of the parameter α. (a)+(e) Default parameters ($\alpha = 0.9$), bad pixels: 12.9%. (b)+(f) $\alpha = 0.95$, bad pixels: 10.7%. (c)+(g) $\alpha = 0.5$, bad pixels: 20.9%. (d)+(h) $\alpha = 0.1$, bad pixels: 23.5%. The larger the parameter, the better the results. Hence, the gradient comparison seems more reliable than the color comparison. This is specific to this scene (lots of textures). This is not true for scenes with many flat regions (the border of the lamp here is better estimated for a smaller value of α).

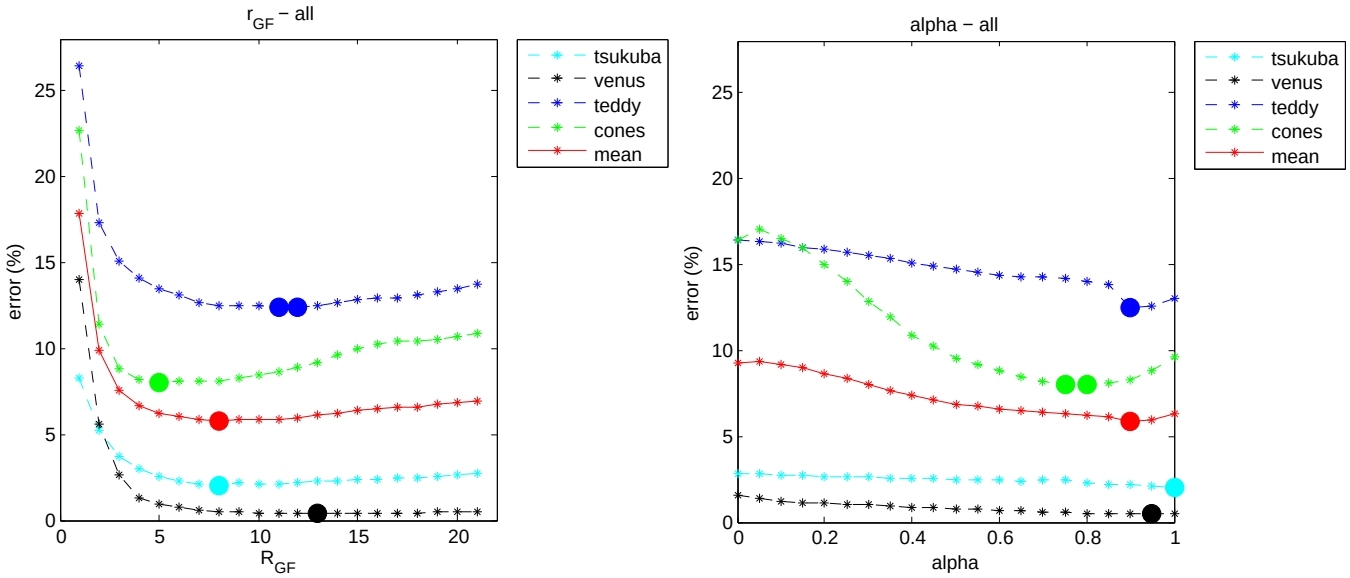
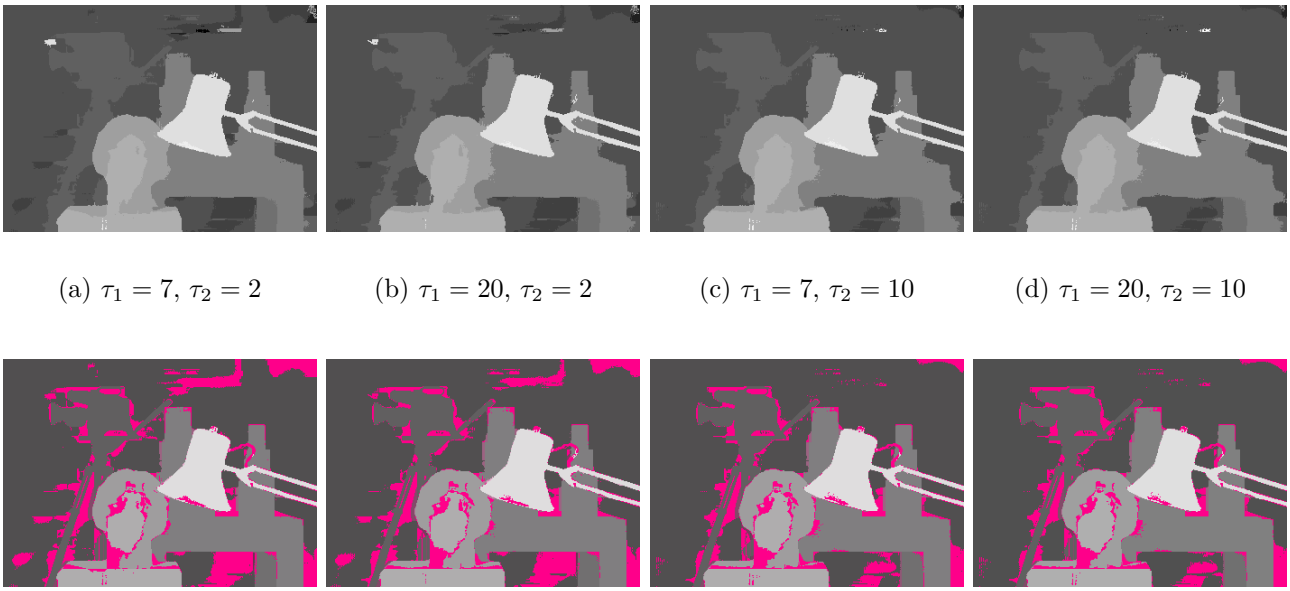


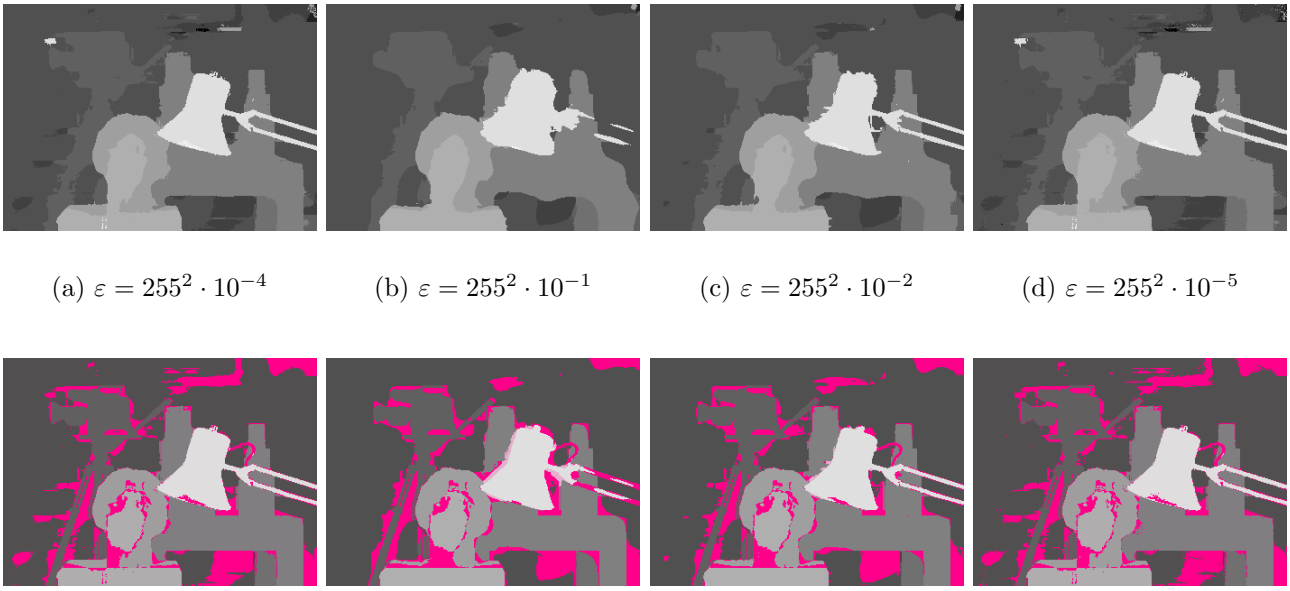
Figure 13: Influence of the parameters r_{GF} (left) and α (right) on errors in the Middlebury benchmark.



(a) $\tau_1 = 7$, $\tau_2 = 2$ (b) $\tau_1 = 20$, $\tau_2 = 2$ (c) $\tau_1 = 7$, $\tau_2 = 10$ (d) $\tau_1 = 20$, $\tau_2 = 10$

(e) $\tau_1 = 7$, $\tau_2 = 2$ (f) $\tau_1 = 20$, $\tau_2 = 2$ (g) $\tau_1 = 7$, $\tau_2 = 10$ (h) $\tau_1 = 20$, $\tau_2 = 10$

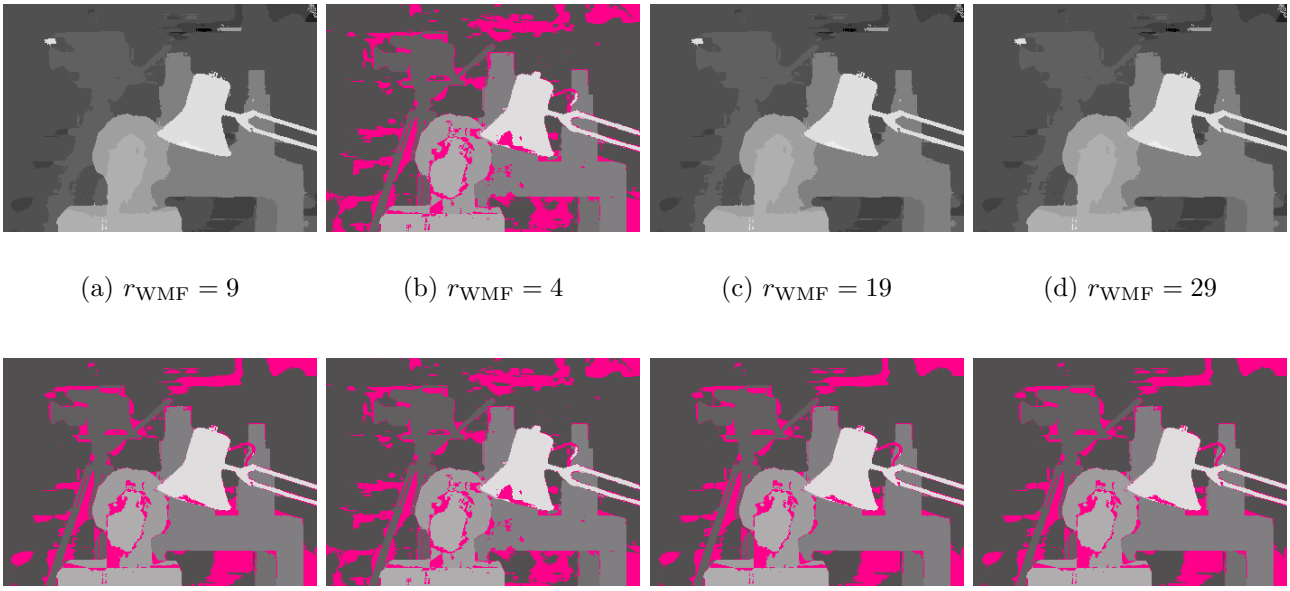
Figure 14: Influence of thresholds τ_1 and τ_2. (a)+(e) Default parameters ($\tau_1 = 7$, $\tau_2 = 2$), bad pixels: 12.9%. (b)+(f) $\tau_1 = 20$, bad pixels: 12.1%. (c)+(g) $\tau_2 = 10$, bad pixels: 9.65%. (d)+(h) $\tau_1 = 20$, $\tau_2 = 10$, bad pixels: 9.49%. The results improve when the thresholds are chosen a little larger. Actually, by default, they are set too small. Thus, big differences do not affect the comparison too much. However, errors appear when they are too large (typically larger than those presented here).



(a) $\varepsilon = 255^2 \cdot 10^{-4}$ (b) $\varepsilon = 255^2 \cdot 10^{-1}$ (c) $\varepsilon = 255^2 \cdot 10^{-2}$ (d) $\varepsilon = 255^2 \cdot 10^{-5}$

(e) $\varepsilon = 255^2 \cdot 10^{-4}$ (f) $\varepsilon = 255^2 \cdot 10^{-1}$ (g) $\varepsilon = 255^2 \cdot 10^{-2}$ (h) $\varepsilon = 255^2 \cdot 10^{-5}$

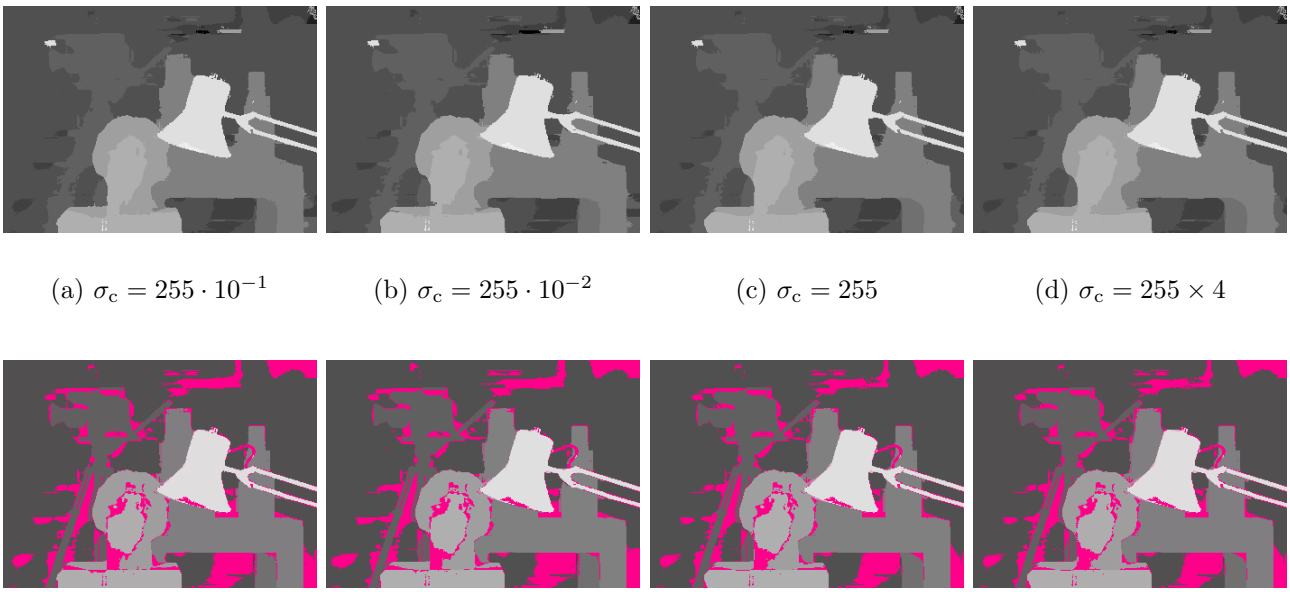
Figure 15: Influence of the regularization parameter. (a)+(e) Default parameters ($\varepsilon = 255^2 \cdot 10^{-4}$). Bad pixels: 12.9%. (b)+(f) $\varepsilon = 255^2 \cdot 10^{-1}$, bad pixels: 13.5%. (c)+(g) $\varepsilon = 255^2 \cdot 10^{-2}$, bad pixels: 12.2%. (d)+(h) $\varepsilon = 255^2 \cdot 10^{-5}$, bad pixels: 13.0%. For large values of ε, the object boundaries are loose and the disparity map is smoother.



(a) $r_{\mathrm{WMF}} = 9$ (b) $r_{\mathrm{WMF}} = 4$ (c) $r_{\mathrm{WMF}} = 19$ (d) $r_{\mathrm{WMF}} = 29$

(e) $r_{\mathrm{WMF}} = 9$ (f) $r_{\mathrm{WMF}} = 4$ (g) $r_{\mathrm{WMF}} = 19$ (h) $r_{\mathrm{WMF}} = 29$

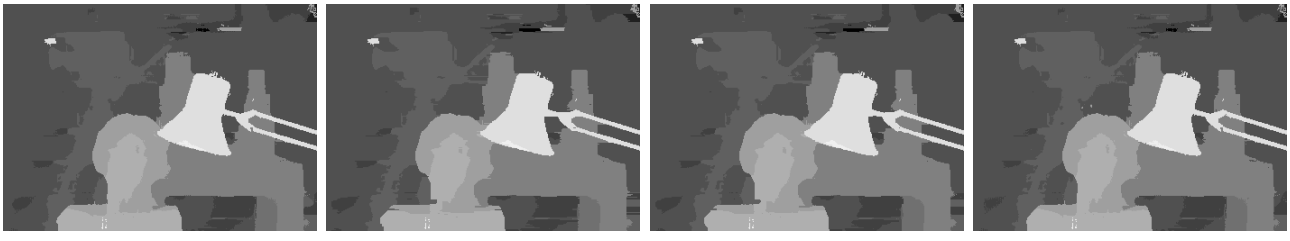
Figure 16: Influence of the size of the weighted median filter. (a)+(e) Default parameters ($r_{\mathrm{WMF}} = 9$), bad pixels: 13.0%. (b)+(f) $r_{\mathrm{WMF}} = 4$, bad pixels: 13.2%. (c)+(g) $r_{\mathrm{WMF}} = 19$, bad pixels: 12.9%. (d)+(h) $r_{\mathrm{WMF}} = 29$, bad pixels: 12.8%. This parameter shows no influence on the results. However, as expected, the results are slightly better when the filter size grows.



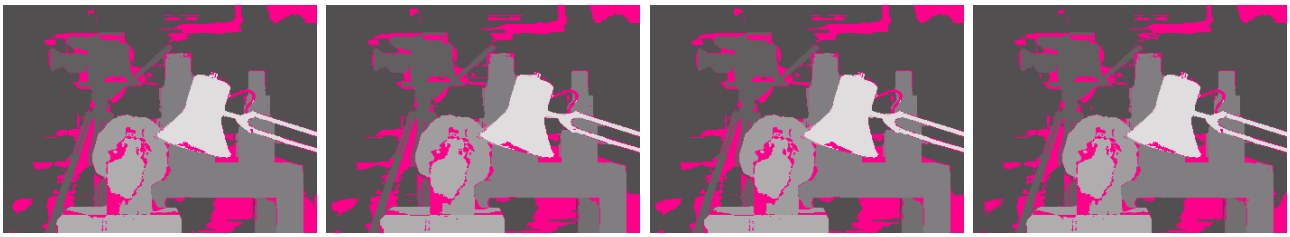
(a) $\sigma_{\mathrm{c}} = 255 \cdot 10^{-1}$ (b) $\sigma_{\mathrm{c}} = 255 \cdot 10^{-2}$ (c) $\sigma_{\mathrm{c}} = 255$ (d) $\sigma_{\mathrm{c}} = 255 \times 4$

(e) $\sigma_{\mathrm{c}} = 255 \cdot 10^{-1}$ (f) $\sigma_{\mathrm{c}} = 255 \cdot 10^{-2}$ (g) $\sigma_{\mathrm{c}} = 255$ (h) $\sigma_{\mathrm{c}} = 255 \times 4$

Figure 17: Influence of the parameter σ_{c}. (a)+(e) Default parameters ($\sigma_{\mathrm{c}} = 255 \cdot 10^{-1}$), bad pixels: 12.9%. (b)+(f) $\sigma_{\mathrm{c}} = 255 \cdot 10^{-2}$, bad pixels: 13.3%. (c)+(g) $\sigma_{\mathrm{c}} = 255$, bad pixels: 12.7%. (d)+(h) $\sigma_{\mathrm{c}} = 255 \times 4$, bad pixels: 12.8%. This parameter has no strong influence, but it should not be chosen too large or too small.

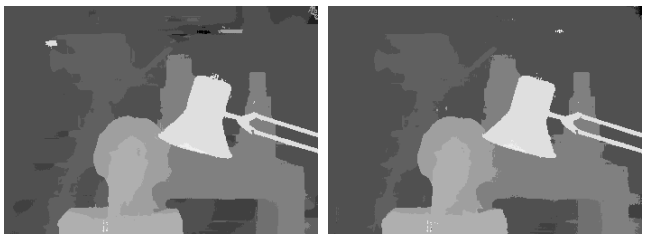


(a) $\sigma_s = 9$ (b) $\sigma_s = 10^{-1}$ (c) $\sigma_s = 2$ (d) $\sigma_s = 15$



(e) $\sigma_s = 9$ (f) $\sigma_s = 10^{-1}$ (g) $\sigma_s = 2$ (h) $\sigma_s = 15$

Figure 18: Influence of the parameter σ_s. (a)+(e) Default parameters ($\sigma_s = 9$), bad pixels: 12.9%. (b)+(f) $\sigma_s = 10^{-1}$, bad pixels: 13.4%. (c)+(g) $\sigma_s = 2$, bad pixels: 13.3%. (d)+(h) $\sigma_s = 15$, bad pixels: 12.7%. This parameter does not change the results a lot, but it should not be chosen too large or too small.



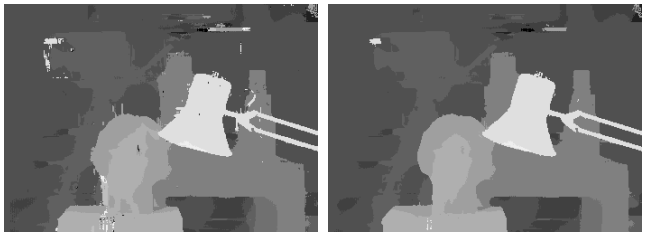
(a) Default parameters (b) Tuned parameters



(c) Default parameters (d) Tuned parameters

Figure 19: Enhancement by tuning properly the parameters. (a)+(c) Default parameters. Bad pixels: 12.9%. (b)+(d) Tuned parameters. Bad pixels: 9.01%. We managed to enhance the result, especially in the background of the scene, while keeping a good estimation anywhere else. Nevertheless, one can notice that the disparity map is smoother, which creates some errors (e.g. on the right border of the lamp or on the statue).

removing big errors (e.g. on the lamp). However, some details are lost (e.g. the wire on the lamp). Actually, one can notice that the densification step mainly enhances the result on flat regions of the disparity map. It does not affect much discontinuity regions. Then, one should consider this step to be a filtering step, as it removes big errors in flat regions.



(a) Without densification

(b) With densification



(c) Without densification

(d) With densification

Figure 20: Enhancement by the densification process. (a)+(c) Without densification. Bad pixels: 14.5%. (b)+(d) With densification. Bad pixels: 12.9%. The densification mainly enhanced the results on nonoccluded regions. It removed some obvious errors easily detected (e.g. on the lamp or on the statue), but it also removed some details (e.g. the wire on the lamp).

5 Examples

5.1 Comparison with Original Algorithm

The authors of the original algorithm [8] propose a Matlab implementation[6]. It performs similarly to our implementation, the slight differences being attributed to:

- Image color to gray conversion formulas.
- Handling of cost for pixels outside the image, term $N(i, i+d)$ of (8): the Matlab code assumes a constant color $R = G = B = b$ with $b = 3$ of I_R outside its boundaries, while we assign the maximum possible cost.
- Numerical inaccuracies.

A comparison on the Middlebury stereo benchmark is given in Table 2. The authors' implementation is *not* their Matlab code. The latter uses $d_{\max} = -1$ instead of 0, but otherwise gives results

[6] https://www.ims.tuwien.ac.at/publications/tuw-202088

Error threshold=1:

Impl.	AR	Rank	Tsukuba			Venus			Teddy			Cones		
Ours	50.5	49	1.92	2.24	7.68	0.26	0.47	2.55	6.98	12.4	16.7	2.83	8.25	7.99
Authors	38.0	27	1.51	1.85	7.61	0.20	0.39	2.42	6.16	11.8	16.0	2.71	8.24	7.66

Error threshold=0.5:

Impl.	AR	Rank	Tsukuba			Venus			Teddy			Cones		
Ours	38.1	28	11.5	11.9	16.1	5.74	6.17	10.4	12.1	18.5	26.0	8.16	13.9	15.6
Authors	35.9	26	11.2	11.7	15.6	5.99	6.43	10.8	11.3	18.1	25.3	7.71	13.7	15.1

Table 2: Results on Middlebury stereo benchmark (AR=Average Rank). First implementation is our code, the second one is the authors' CUDA code results, as stored in the benchmark.

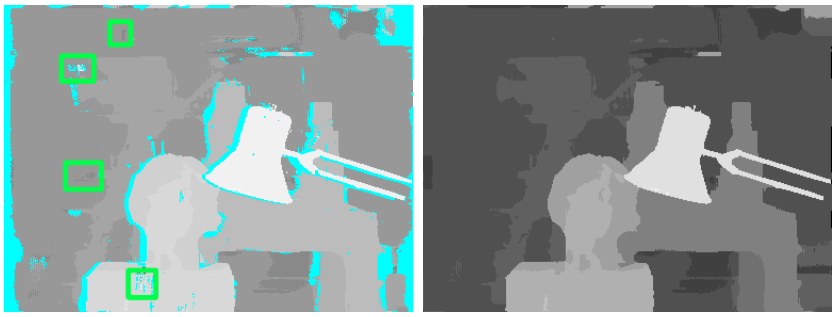


Figure 21: CUDA implementation result on Tsukuba pair. Left: disparity map with occluded pixels. Right: final disparity map.

similar to our implementation. The CUDA code (not available) gives sensibly better results for the error threshold = 1 (only disparity errors strictly larger than 1 pixel are reported). Comparing both error thresholds, we see that our implementation has significantly more errors of 1 pixel.

After a personal communication with the CUDA implementation author [5], it appears that on the Tsukuba pair, the left-right consistency threshold is $d_{LR} = 1$ instead of 0. Whereas taking such a threshold yields slightly better results on Tsukuba and Cones, it deteriorates Venus and Teddy, so does not improve the final rank in the benchmark.

Moreover, we see in the green areas of Figure 21 some wrong disparity pixels that are not detected as occluded with the CUDA code but are still corrected in the final map. This is inconsistent with the described algorithm. No explanation could be found.

5.2 Results

We now show some other results obtained by our implementation. The experiments were done on images without ground truth. Hence, the results can only be appreciated visually. If not stated otherwise, the results shown were obtained by the whole process (including occlusion detection, filling and post-processing). The algorithm was launched with the default parameters (see previous section).

Globally, the results tend to show that this method gives very good results. The default parameters are fairly well chosen, which is sufficient most of the time. Hence, one can launch the code without tuning any parameter and yet get satisfactory results. Nevertheless, they can be enhanced by tuning the parameters, which are numerous. Thus, one has to tune and test each parameter several times to find the optimal parameters.

Chair. The code was tested on the Chair pair (Figure 22). Results are fairly good. However note that this pair is easy to deal with, since almost none of the stereovision difficulties are encountered (almost no reflection, textured regions).

This result is compared with the result obtained before the left-right consistency check (Figure 23). Results are visually better. However, Figure 24 shows that this process degrades some parts of the disparity map, especially on some objects boundaries. Densification leads to better overall results, but some accuracy is lost.

Umbrella. Results on the Umbrella pair are shown in Figure 25. This experiment discloses a strong fattening effect, that is, neighbor points of objects in the foreground are labelled as if they belonged to the objects. Hence, such objects appear fatter than they actually are. This specific example shows that the boundaries of the green umbrella are not clearly drawn, since they are badly estimated (e.g. the left ear is bigger than expected). The background adopted the umbrella disparity.



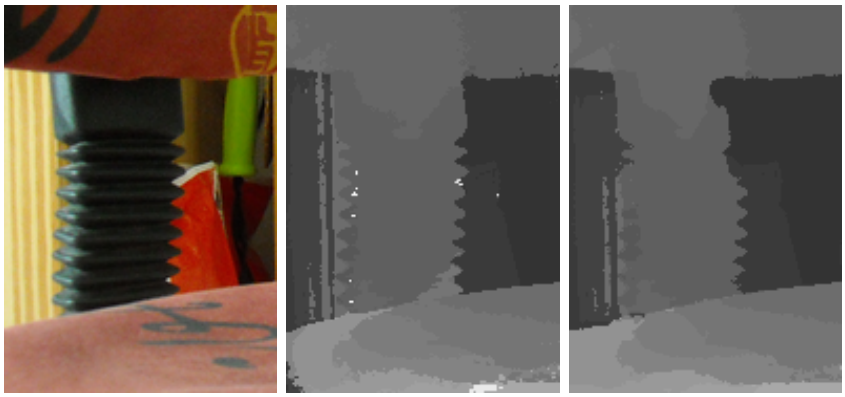
(a) (b) (c)

Figure 22: Chair pair. (a) Left image. (b) Right image. (c) Result. The result is very satisfactory, the depth of the scene seems probable.



(a) (b) (c)

Figure 23: Chair pair. (a) Left image. (b) Before the occlusion detection. (c) After the whole process. After the occlusion detection and the densification step, the disparity map is cleaner, most of obvious errors have been removed. Moreover, the disparity has been well interpolated on the left border of the image.



(a) (b) (c)

Figure 24: Chair pair (detail). (a) Left image. (b) Before the occlusion detection. (c) After the whole process. In this detail, the disparity was better estimated before the occlusion detection, even though the densification process removed some errors.



(a) (b) (c)

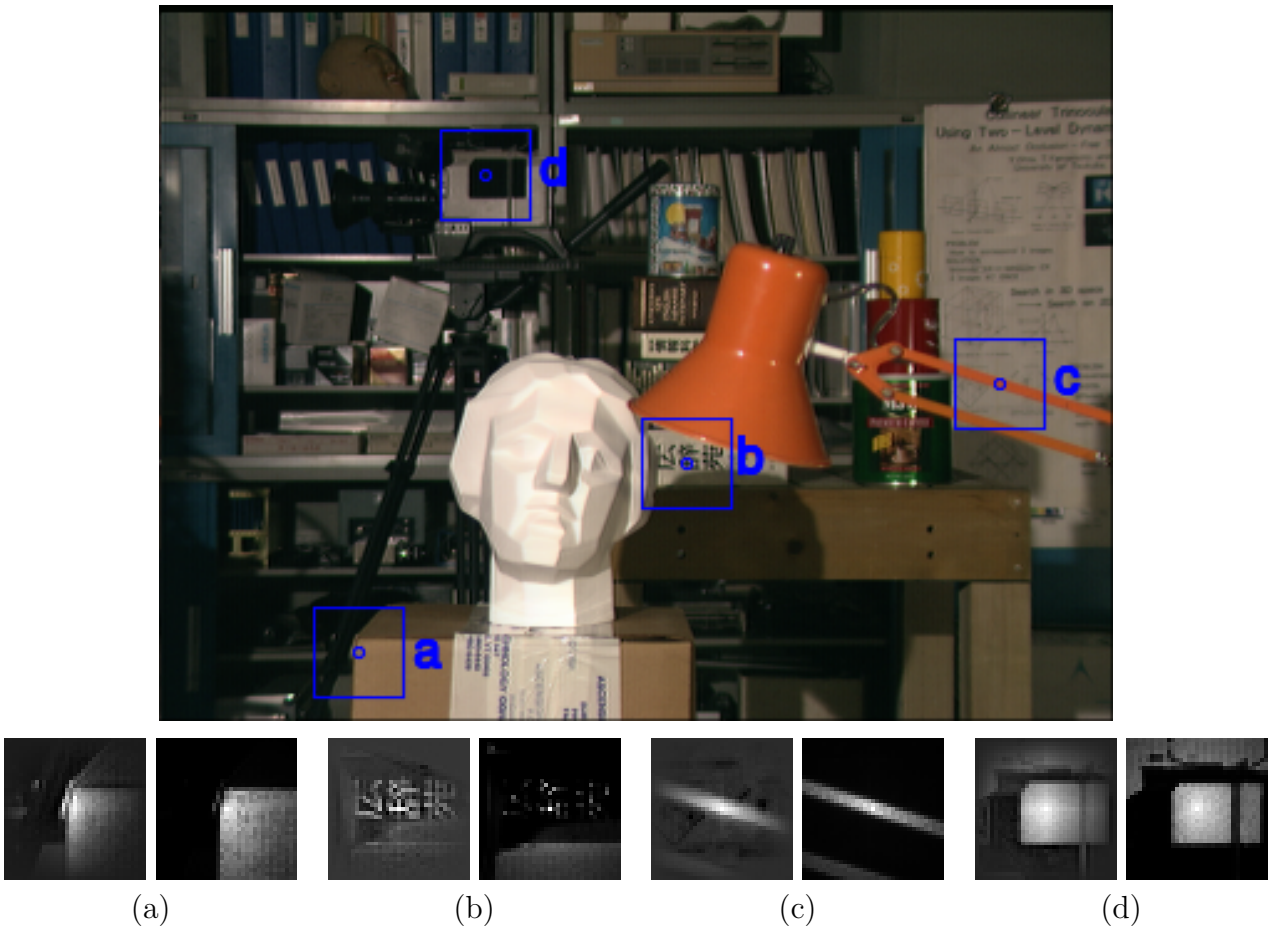
Figure 25: Umbrella pair. (a) Left image. (b) Right image. (c) Result.

6 Discussion on Adaptive-Window Approaches

The effectiveness of the method described in this article mainly relies on the use of the guided filter. Formula (17) expresses this filter with adaptive weights, which are shown in Figure 26. Hence, the filter only takes into account some selected neighboring pixels. The window is thus adapted to each pixel.

To compute the weights associated to a pixel i, we can use (17) or (23), but another possibility is for each pixel j to apply the guided filter to the Dirac image $p_j(k) = \delta_{jk}$ (1 at pixel j, 0 everywhere else) and take the filtered image at i, $W_{ij}^{\mathrm{GF}} = q_j(i)$. Note that only pixels j at l^∞-distance at most $2 \cdot r_{\mathrm{GF}}$ from i should be considered, the sum in (23) reducing to 0 otherwise.

Similar strategies, based on adaptive windows, have been proposed previously. Indeed, local stereo-matching methods highly rely on window comparison. Such comparison assumes that the depth within the window does not vary much, which is not the case on depth discontinuities. Hence, local methods require choosing the best window to perform well. The choice of a window depends on many criteria. Windows have to be large enough to avoid low signal to noise ratio. However, when they are too large, they may cover regions in which the depth varies a lot. Indeed, in such



(a) (b) (c) (d)

Figure 26: Adaptive weights in the guided filter (left image for each patch) and of Yoon-Kweon's bilateral filter [14] (right image for each patch).

regions, the comparison may be affected by projective distortion due to strong depth discontinuities. Accordingly, several approaches have been tested to design adaptive windows. This strategy is aimed at selecting for each pixel the appropriate window.

One of the former approaches consists in adapting the window size to the pixel, knowing that it should depend on local variations of intensity and disparity [6]. Indeed, high variations of disparity mean depth discontinuities, which often entails distortions. Unfortunately, this method needs to evaluate the variation of the disparity, so it strongly depends on the initial disparity estimation.

Another approach is to adapt the shape of the window to fit with the shape of the discontinuities. Boykov et al. [1] find for each pixel an arbitrarily shaped connected window chosen by evaluating the plausibility of each pixel to have the same disparity as the pixel under consideration. Veksler [11, 12] uses a set of windows (called *compact windows*) of different size and shape, and chooses the most reliable (with minimal average error). However, this method does not consider general shapes of window, which limits its performance.

More recently, a new adaptive-window method, proposed by [14], uses square weighted windows. For each pixel, the weights are computed so that only neighboring pixels of interest (i.e. belonging to the same object) have heavy weight. Since the Gestalt principles state that grouping rules based on similarity and proximity are among the strongest, the weights should depend on the color similarity and the spatial proximity. The weights are chosen so that they decrease quickly while the pixels i

and j are dissimilar or not close enough:

$$W_{ij} := \frac{1}{K_i} \exp\left(-\frac{\|i-j\|}{\sigma_s^2}\right) \cdot \exp\left(-\frac{\|I_L(i)-I_L(j)\|_c}{\sigma_c^2}\right) \tag{35}$$

where $\|\cdot\|_c$ denotes the Euclidean distance between two colors represented in color space[7]. Note that this is close to the bilateral filter used in the post-processing. Figure 26 shows examples of weights obtained by this formula. A limitation of this method is that it is computationally expensive, while the guided filter discloses similar results with a lower complexity. Note that the name "bilateral filter" [10] is the most recent one for a filter known as the sigma filter [7], the Yaroslavsky filter [13] and the SUSAN filter [9].

The bilateral weights in Figure 26 are computed with parameters: $\sigma_s^2 = 17.5$ (half the window size 35×35) and $\sigma_c^2 = 14 \cdot 3$ (3 channels). The weights appear roughly equivalent by both methods. Note that there is implicitly a decreasing tendency with the distance to central pixel i in the guided filter, because the further pixel j, the fewer windows ω_k containing i and j. A full comparison with other adaptive-window methods will be the object of future work.

Image Credits

All images by the authors (license CC-BY-SA) except:

 Middlebury.

References

[1] Y. Boykov, O. Veksler, and R. Zabih, *A variable window approach to early vision*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 20 (1998), pp. 1283–1294. http://dx.doi.org/10.1109/34.735802.

[2] F.C. Crow, *Summed-area tables for texture mapping*, ACM SIGGRAPH Computer Graphics, 18 (1984), pp. 207–212. http://dx.doi.org/10.1145/964965.808600.

[3] G. Facciolo, N. Limare, and E. Meinhardt-Llopis, *Integral images for block matching*. preprint, http://www.ipol.im/pub/pre/57, 2013.

[4] K. He, J. Sun, and X. Tang, *Guided image filtering*, in 11th European Conference on Computer Vision, Springer, 2010, pp. 1–14. http://dx.doi.org/10.1007/978-3-642-15549-9_1.

[5] A. Hosni. Personal communication (Feb. 2013).

[6] T. Kanade and M. Okutomi, *A stereo matching algorithm with an adaptive window: theory and experiment*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 16 (1994), pp. 920–932. http://dx.doi.org/10.1109/34.310690.

[7] J-S. Lee, *Digital image smoothing and the sigma filter*, Computer Vision, Graphics, and Image Processing, 24 (1983), pp. 255–269. http://dx.doi.org/10.1016/0734-189X(83)90047-6.

[7] The original article [14] uses Euclidean distance in CIELab color space, in our experiments we use L^1 distance in RGB space.

[8] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz, *Fast cost-volume filtering for visual correspondence and beyond*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2011, pp. 3017–3024. http://dx.doi.org/10.1109/CVPR.2011.5995372.

[9] S.M. Smith and J.M. Brady, *Susan–a new approach to low level image processing*, International Journal of Computer Vision (IJCV), 23 (1997), pp. 45–78. http://dx.doi.org/10.1023/A:1007963824710.

[10] C. Tomasi and R. Manduchi, *Bilateral filtering for gray and color images*, in 6th International Conference on Computer Vision (ICCV), IEEE, 1998, pp. 839–846. http://dx.doi.org/10.1109/ICCV.1998.710815.

[11] O. Veksler, *Stereo correspondence with compact windows via minimum ratio cycle*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 24 (2002), pp. 1654–1660. http://dx.doi.org/10.1109/TPAMI.2002.1114859.

[12] ——, *Fast variable window for stereo correspondence using integral images*, in IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), vol. 1, IEEE, 2003, pp. I–556. http://dx.doi.org/10.1109/CVPR.2003.1211403.

[13] L.P. Yaroslavsky, *Digital picture processing: an introduction.*, Springer-Verlag New York, Inc., ISBN 3-540-11934-5, 1985.

[14] K-J. Yoon and I-S. Kweon, *Adaptive support-weight approach for correspondence search*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 28 (2006), pp. 650–656. http://dx.doi.org/10.1109/cvpr.2005.218.