100+ Times Faster Weighted Median Filter (WMF)

Qi Zhang[†] Li Xu[‡] Jiaya Jia[†]
[†] The Chinese University of Hong Kong
[‡] Image & Visual Computing Lab, Lenovo R&T
zhangqi@cse.cuhk.edu.hk xulihk@lenovo.com leojia@cse.cuhk.edu.hk
http://www.cse.cuhk.edu.hk/leojia/projects/fastwmedian/

Abstract

Weighted median, in the form of either solver or filter, has been employed in a wide range of computer vision solutions for its beneficial properties in sparsity representation. But it is hard to be accelerated due to the spatially varying weight and the median property. We propose a few efficient schemes to reduce computation complexity from $O(r^2)$ to $O(r)$ where r is the kernel size. Our contribution is on a new joint-histogram representation, median tracking, and a new data structure that enables fast data access. The effectiveness of these schemes is demonstrated on optical flow estimation, stereo matching, structure-texture separation, image filtering, to name a few. The running time is largely shortened from several minutes to less than 1 second. The source code is provided in the project website.

1. Introduction

There is no need to explain how common and elementary local filters are in computer vision. In this paper, we study and retrofit a fundamentally useful tool, i.e., *weighted median*, considering its importance along several major lines of applications. As a local operator, weighted median can effectively filter images while not strongly blurring edges. More importantly, it mathematically corresponds to *global optimization* [14], which produces results with fully explainable connections to *global energy minimization*. In many computer vision problems, including stereo matching, optical flow estimation, and image denoising, applying local weighted median suitably is equivalent to employing solvers that involve global L_1 sparse regularizers, which is rather common now in sparsity pursuit in a lot of tasks.

Unweighted median filter was accelerated in previous work. But it is almost impossible to apply these methods to weighted median because of the spatially varying weights for each local window. These varying weights hamper simplifying computation in a straightforward way. This is why many previous computer vision systems still rely on a direct implementation of weighted median, which has to spend several minutes even using small windows on a one-megapixel image.

On the other hand, there have been several schemes to speed up weighted *average*, such as bilateral filter [20, 16, 4, 2, 1, 9, 6, 24, 18], domain transform filter [8], and guided filter [10]. Weighted median is different from them due to a few special characteristics. 1) The filtering kernel is not separable. 2) It cannot be approximated by interpolation or down-sampling. 3) There is no iterative solution. Taking prior techniques for accelerating bilateral filter as an example, Gaussian weights are generally assumed, which may not be satisfied in weighted median.

Our contribution in this paper lies on a few algorithms to accelerate weighted median. The novelty attributes to three main techniques making respective use of data-weight distribution by a new joint histogram to dynamically allocate weights and pixels, a median tracking algorithm to reduce time of seeking median by leveraging color coherency of images, and data structure sparsity to reduce the data access cost. They are put together to construct a practically workable system with $O(r)$ complexity where r is the kernel size. It is notable that this complexity is only associated with a small constant, empirically effective to shorten running time.

1.1. Related Work

Median Filter While there is little work accelerating weighted median, simpler unweighted filter finds several solutions. Huang [11] proposed a sliding window approach leveraging histograms to compute median in $O(r)$ time, which is further accelerated to $O(\log r)$ with the distributed histograms [21]. Constant time median algorithms [17, 5] were also proposed, focusing on reducing histogram update complexity. Kass and Solomon [12] introduced another constant time algorithm based on integral of smoothed local histograms.

Two main issues make these methods not directly usable

in weighted median filter (WMF). First, the constant associated with variable r can be very large. For example, an 8-bit single channel image needs 256 histogram bins to count pixels, which is still a big constant in practice, not to mention higher-precision inputs. Second, traditional histograms are not suitable for WMF in that all weights are altered completely when the local window shifts for even one pixel. This almost removes traditional reusability of data in overlapping regions for adjacent windows.

Weighted Average Filter Bilateral filter (BLF) [20] is a representative weighted average that sums pixels according to the Gaussian distance in space and range. Paris and Durand [16] formulated it in a high dimensional space, and accelerated it by down-sampling since Gaussian is a low-pass filter. With this thought, different data structures, such as bilateral grid [4], Gaussian KD-trees [2], permutohedral lattice [1] and adaptive manifold [9] were adopted. A few other algorithms used sub-sampling in range. Durand and Dorsey [6] and Yang *et al*. [24] decomposed bilateral filter into several spatial ones and obtained the final result by linear interpolation. Porikli [18] used integral histograms.

These methods are based on the condition that BLF uses Gaussian weights and takes average, which is not satisfied in weighted median.

Weighted Median The most related work is recent utilization of WMF in stereo matching [15]. This method performs edge preserving filters many times depending on the number of input labels to generate histograms. It is tailored for stereo matching where the input contains only a limited number of discrete disparities. Filtering images with hundreds of intensity levels would make it slower. Besides, this filter is based on edge-preserving guided filter [10] or domain transform filter [8]. The spatial weight structure is thus specific to the filter it employs. Different from this method, we propose a general weighted median filter that handles higher precision inputs and arbitrary weight assignment. It can be applied to many applications besides stereo matching.

Global Approach Weighted filters are related to edge-preserving smoothing operators where global solvers exist [7, 22]. In this paper, we show that weighted median filter helps accelerate a number of global optimization problems under complex objective definitions.

2. Technical Background

Weighted median filter (WMF) is an operator that replaces the current pixel with the weighted median of neighboring pixels within a local window. Formally, in processing pixel p in image I, we consider only pixels within the local window $\mathcal{R}(p)$ of radius r centered at p. Different from conventional unweighted median filter, for each pixel $q \in \mathcal{R}(p)$, WMF associates it with a weight w_{pq} based on the affinity of pixel p and q in corresponding feature map $\mathbf{f}$, i.e.,

$$w_{pq} = g(\mathbf{f}(p), \mathbf{f}(q)), \tag{1}$$

where $\mathbf{f}(p)$ and $\mathbf{f}(q)$ are features at pixels p and q in $\mathbf{f}$. g is a typical influence function between neighboring pixels, which can be in Gaussian $\exp\{-\|\mathbf{f}(p) - \mathbf{f}(q)\|\}$ or other forms.

Denoting $I(q)$ as the value at pixel q in image I and $n = (2r+1)^2$ as the number of pixels in $\mathcal{R}(p)$, we express the value and weight elements for all pixels in $\mathcal{R}(p)$ as $\{(I(q), w_{pq})\}$. By sorting values in an ascending order, the weighted median operator for pixel p returns a new pixel p^* and $I(p)$ is replaced by $I(p^*)$. This process to get p^* is

$$p^* = \min k \quad \text{s.t.} \quad \sum_{q=1}^{k} w_{pq} \geq \frac{1}{2} \sum_{q=1}^{n} w_{pq}. \tag{2}$$

This definition means for all pixels before p^*, the sum of corresponding weights should be approximately half of all weights summed together. $\mathbf{f}$ map in Eq. (1) determines weights. Practically, $\mathbf{f}(p)$ can be intensity, color, or even high-level features depending on problem definition.

2.1. The Challenge

To show the difficulty in approximating weighted median filter (WMF), we start with an easier problem, i.e., unweighted median filter, which can be regarded as a special case with all $w_{pq} = 1$ in Eq. (2). For each filter, to find where the median is, one must order values. The trivial solution is to sort all data in complexity $O(r^2 \log r)$ and then find the median in $O(1)$ time.

Because data are usually range-limited – an 8-bit image has at most 256 different gray-scale levels – it is feasible to use counting-based sorting by histograms [11, 17, 12, 21, 5] to make it run faster. Maintaining a histogram of values in the local window achieves sorting naturally where the median can be found by counting elements in each bin. Another advantage is the data reusability. For a 100×100 kernel, adjacent local windows share 99% common pixels. They can be used again when searching for the next median [11]. This reduces complexity to $O(r)$.

Nevertheless, in WMF, when weight w_{pq} varies according to feature distance, above schemes cannot be used anymore. It is because even for the same pixel q, changing the filter center p also updates w_{pq}. All values put into a bin of the histogram need recalculation substantially. In other words, shared pixels in two adjacent windows do not save much computation due to the same amount of weight computation each time. Storing weights in the above histogram is not helpful. Running time spent to weight estimation still amounts to $O(r^2)$.

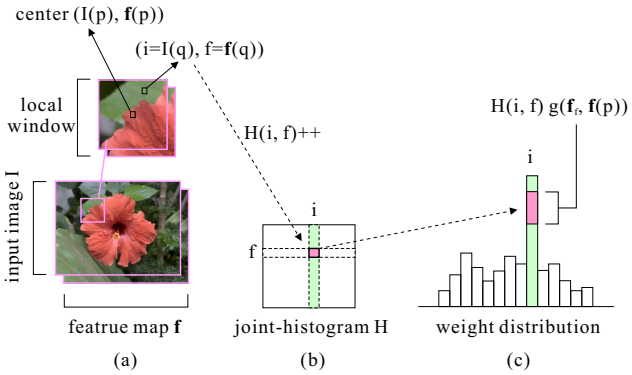


Figure 1. Joint-histogram illustration. (a) Input image and a local window. (b) Joint-histogram containing all pixels in the window according to value index i and feature f. (c) Total weight calculated for each i.

3. Our Framework

Our method is unlike previous bilateral or other average-based filters because we cannot apply down-sampling, linear interpolation, separable kernel estimation, or iterative algorithms. In what follows, we present three new steps to greatly accelerate WMF.

1. *Joint-histogram*, which is an unconventional but very useful two-dimensional histogram structure for storing pixel count.

2. *Median tracking* strategy using *balance counting box (BCB)* for dynamically finding median given the neighboring-pixel WMF result.

3. *Necklace table*, data structure to leverage data sparsity for instant data access.

4. Joint-Histogram

We explain the first technique, i.e., the joint-histogram, in this section. As mentioned in Section 2.1, it is not suitable to use conventional histograms to store weights in the WMF system. Our new joint-histogram can, contrarily, regenerate weights every time the window shifts. It is a two-dimensional count-based histogram (see Fig. 1(b)), which does not store weights directly, but instead the pixel counts in different bins according to their features. The two dimensions are indexed by pixels values and features respectively.

Specifically, each pixel q in the filter is associated with its value $I(q)$ and feature $\mathbf{f}(q)$, as explained in Section 2. Suppose there are N_I different pixels values, we denote the pixel value index for $I(q)$ as i after listing all possible pixel values in an ascending order as $\mathcal{I} = \{I_0, I_1, \ldots, I_{N_I-1}\}$. Also we denote the total number of different features as N_f. A feature index f for $\mathbf{f}(p)$ can be accordingly resulted in after ordering all features as $\mathcal{F} = \{\mathbf{f}_0, \mathbf{f}_1, \ldots, \mathbf{f}_{N_f-1}\}$. A feature can be intensity, color, or even high-level structures in different applications.

In our 2D joint-histogram H, pixel q is put into the histogram bin $H(i, f)$ in the f-th row and i-th column. Thus, the whole joint-histogram is constructed as

$$H(i, f) = \#\{q \in \mathcal{R}(p) | I(q) = I_i, \mathbf{f}(q) = \mathbf{f}_f\}, \quad (3)$$

where operator $\#$ counts the number of elements. This counting scheme enables fast weight computation even when the window shifts. Now, for any pixel belonging to bin (i, f), considering filter center p, its weight can be immediately computed as $g(\mathbf{f}_f, \mathbf{f}(p))$. By traversing the joint-histogram, all weights can be obtained (see Fig. 1(c)). The total weight for all pixels with value index i is

$$w_i = \sum_{f=0}^{N_f-1} H(i, f)\, g(\mathbf{f}_f, \mathbf{f}(p)). \quad (4)$$

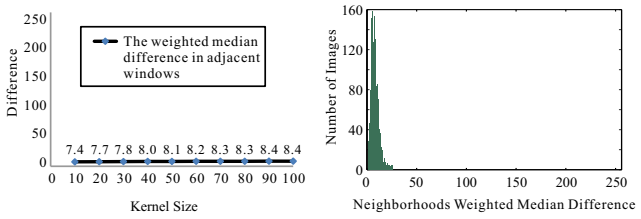
With w_i available for all i, weighted median can be found.

Algorithm By incorporating the joint-histogram with the sliding window strategy, our algorithm processes the input image in a scanline order. For each location, two steps are performed. Initially at the first location in the image, the joint-histogram counts the number of pixels with index (i, f) within the filter range. The number is put into the corresponding cell.

- **Step 1 - Median Finding** We traverse the joint-histogram to compute weights $\{w_k\}_{k=0}^{N_I-1}$ according to Eq. (4). They are summed to get the total weight w_t in the first pass. Then in the second pass, we sum weights up to half of w_t and output this pixel value.
- **Step 2 - Shift & Update** We shift the filter to the next location if it is not the end of this image. We update the joint-histogram by reducing the count in the corresponding cell by one for each pixel leaving the filter and increase it by one for each pixel moving in.

Analysis This algorithm greatly speeds up median filter. We analyze each step's efficiency. Step 1 needs $O(N_I * N_f)$ time because it traverses the joint-histogram, which is independent of the filter size and needs only constant time. The complexity of Step 2 is $O(r)$ since we only update up to $2 \times (2r + 1)$ pixels in the non-overlapping regions in the two adjacent windows.

The bottleneck is seemingly in Step 2 due to its higher complexity order. In fact, Step 1 is computationally more expensive because the cost to traverse the histogram cannot be overlooked. Note big-constant is a common problem for filter approximation, especially for those based on histograms. In order to further accelerate it, we propose two more strategies in the following sections, making use



(a) (b)

Figure 2. Local color consistency. (a) Average difference of weighted medians in adjacent windows on 1000+ natural images under different window sizes. The difference is only 7–8 grayscales in the 8-bit range. (b) Distribution of average difference between neighboring weighted medians. Almost all images have average median changing no more than 10% of total range, which shows that local color consistency is a general property.

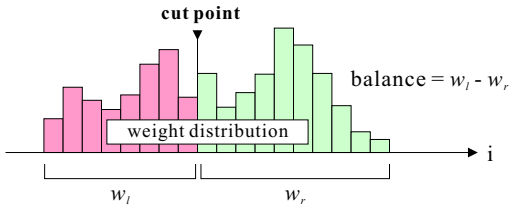


Figure 3. Illustration of the balance and cut point.

of image color consistency and sparsity. They lead to 300-2000 times more speedup for Step 1. These techniques can be applied to other histogram-based algorithms owing to their generality and capability of adopting various weight definitions.

5. Median Tracking

In our extensive experiments, colors in natural images and features in other domains are found locally consistent in most cases. One pixel has a very large chance to be similar to some of its neighbors. This fact enables an algorithm to reduce computation to seek median in the joint-histogram by 97%.

Commonly existing locally stable features make the WMF result remain unchanged most of the time when processing images in a scanline order. We conduct several quantitative experiments; one is illustrated in Fig. 2. It shows that the average difference of weighted median results in adjacent windows is around 7–8 for 8-bit grayscale images, which is within only 3% of total range. We propose a new counting scheme to exploit this finding.

5.1. Cut Point and Balance

We first introduce the concepts of *cut point* and *balance*. By definition, a weighted median is found right at the location where the cumulated weight reaches half of the total weight w_t. We regard the position to find the weighted median at current filter as a cut point, as shown in Fig. 3,

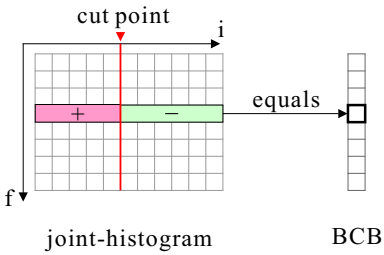


Figure 4. Relationship between BCB and our joint-histogram. By separating each row into the left and right parts according to the cut point in the joint-histogram, their total count difference is put into the corresponding cell in BCB.

where the sum of weights w_l to the left is similar to the sum w_r on the right. In this regard, the difference between w_l and w_r is close to zero. This is referred to as the *balance*, which helps find weighted median by solving

$$\min |w_l - w_r| \quad \text{s.t.} \quad w_l - w_r \notin \mathbb{R}^-.$$

Now it is clear if a cut point produces the smallest non-negative balance, the weighted median is found.

With the above definitions, a weighted median is actually a suitable cut point, which can be quickly obtained in general given the result computed in previous filter process in an adjacent window. Our median tracking procedure is presented below.

Median Tracking Scheme Given the cut point $c(p)$ and corresponding balance obtained by WMF on the window centered at p, we estimate the new weighted median in the adjacent window at $p+1$. Starting from $c(p)$, if it yields a positive balance in window $p+1$, we shift the cut point to the left by one pixel, i.e. $c(p)-1$. It repeats until the balance becomes negative. Then the point before the last one is set as $c(p+1)$. Contrarily, if $c(p)$ yields a negative balance initially, we right shift it until the balance turns positive.

Advantages Due to the statistical local color/feature consistency, only a few shifts (no more than 8 on average) are enough to find the median in window $p+1$. Compared to the common practice to sum weights starting from the first pixel or the straightforward implementation to cumulate cells in our joint-histogram, the above strategy speeds up weight sum calculation at least 256/8 = 32 times.

5.2. Balance Computation

One thing we need discuss further is to compute balance. Naive computation based on its definition scans all weights, which is inefficient. We notably improve it by introducing a new counting scheme and present an intriguing way in result update in windows with fast scan.

Our solution is to maintain a *balance counting box (BCB)* during the course of computing the balance. BCB is

a 1D array B of N_F cells – the same size of columns in our joint histogram. As shown in Fig. 4, the f-th cell records the difference of pixel numbers on the two sides of the cut point in the corresponding row of the joint-histogram. Formally, the f-th cell $B(f)$ is constructed as

$$B(f) = \#\{q \in \mathcal{R}(p) | I(q) <= c, \mathbf{f}(q) = \mathbf{f}_f\} \\ -\#\{r \in \mathcal{R}(p) | I(r) > c, \mathbf{f}(r) = \mathbf{f}_f\}, \quad (5)$$

where c is the cut point location. $B(f)$ measures imbalance of pixel numbers with regard to feature $\mathbf{f}_f$ on two sides of c.

Understanding BCB B is very important in our system. It indicates how pixels distribute in the two sides of the cut point for each weight because w_{pq} is completely determined by $\mathbf{f}(q)$ when p is the filter center. In WMF computation, the cut point could vary even for adjacent windows that overlap a lot. But BCB makes it possible to re-calculate the balance in the new window quickly, even *without* needing to know the detailed weight for each pixel. This is a fascinating step, as a number of operations to scan data are saved naturally.

With BCB, the balance b in a window can be obtained as

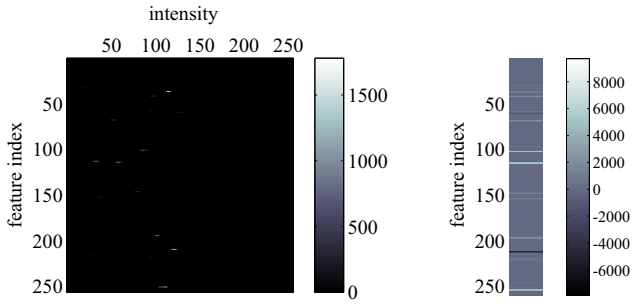
$$b = \sum_{f=0}^{N_f-1} B(f)\, g(\mathbf{f}_f, \mathbf{f}(p)). \quad (6)$$

This simple expression brings us a brand new way to calculate weighted median. That is, instead of cumulating weights, which is costly, we can calculate a lightweight balance measure counting pixel number difference. It is sufficient to tell if the weighted median is found. Specifically, if b is negative, we not only know the cut point c is wrong, but also get the idea in which direction we should move c. Corresponding conclusion can be reached for positive b.

BCB Algorithm In our joint-histogram framework, we maintain and update BCB as well as the cut point as follows.

1. Initially, we set the cut point to 0, and initialize BCB by counting pixels for each row of the joint-histogram.
2. We seek median by shifting the cut point and checking the balance in Eq. (6). This process is the same as the *median tracking scheme* presented in Section 5.1. Every time the cut point shifts, we update BCB based on the joint-histogram.
3. To process a new window largely overlapped with the current one, we update BCB by only examining exiting/entering pixels, using only a small number of plus and minus operations.

This algorithm avoids traversing the joint-histogram for computing all weights. It also captures the key properties of weighted median. The employment of the cut point, balance, and BCB simplifies computation a lot as mentioned above.



(a) Joint-histogram (b) BCB

Figure 5. Data sparsity in joint-histogram (a) and BCB (b) for the example shown in Fig. 1(a). A very small number of cells contain non-zero values. Features used here are 256 clustered colors of the image. We explain it in Sec. 7.

6. Necklace Table

In addition to above algorithms, we also exploit the sparsity of data distributions for WMF in our new structures. When constructing the joint-histogram and BCB, it always happens that 90%-98% bins/cells are empty in our quantitative evaluation. It means traversing these elements wastes time. Fig. 5 shows an example.

In this section, we present another data structure called *necklace table*. It is a counting table enabling fast traversal. It supports very fast operators as

- $O(1)$ data access;
- $O(1)$ element insertion;
- $O(1)$ element deletion;
- $O(N)$ traversal where N is the *non-empty* cell number.

Note these four fast operations cannot be achieved simultaneously by array or linked-list. Our data structure is composed of two parts. They are

- **counting array** that is accessed instantly by index and
- **necklace**, which is an unordered double-linked cycle built on the counting array for non-empty cells.

One necklace-table example built on our BCB is shown in Fig. 6(a). Each cell can be accessed randomly by index. But when the goal is to traverse all non-empty cells, the necklace can achieve sequential access. The necklace in this array is initialized as a self-linked head node. Two links are established for each non-empty cell pointing to previous and next elements.

The above mentioned operations are realized following the example in Fig. 6(b) & (c). Data access and traversal are straightforward. For element insertion, we can instantly access the array and update the count. If the cell is empty originally, we insert it to the necklace between the head and its next element. Element deletion reverts this process.

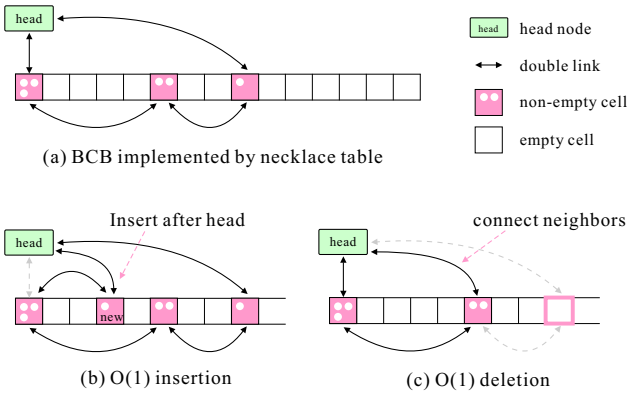


Figure 6. Demonstration of the necklace table.

Unlike traditional linked-list that keeps an order for data access, our necklace is unordered thanks to the available array. It allows for very fast traversal by skipping all empty cells. This data structure is used to implement our BCB and all columns of the joint-histogram. It further accelerates element traversal for 10-50 times depending on data sparsity.

7. Discussion and Experiments

Short Summary Acceleration by using median tracking and necklace table is substantial. Without them, the whole 2D joint-histogram has to be visited to compute weights. Median tracking reduces search range regarding pixel values; only the median difference between two windows matters. Necklace table further accelerates it in the feature space. With these two algorithms, only a small portion (0.02%–0.3%) of cells are visited each time. The big constant discussed in Section 4 is eliminated in our framework.

Implementation Notes Inputs to our system are I and $\mathbf{f}$. When handling floating point values in I in optical flow and stereo matching, we cluster the input to a limited number of discrete values for WMF computation. High-order features $\mathbf{f}$, such as RGB color and patch-based features, are also clustered initially for high efficiency in constructing the joint-histogram. In our experiments, when quantizing RGB features in a color image into 256 different values, i.e., $N_f = 256$, the average PSNR is 45, high enough for satisfying result generation. Our framework also supports arbitrary weight definition on features simply by varying $g(\cdot)$ in Eq. (1).

Performance Evaluation Our experiments are conducted on a PC with an Intel i7 3.4GHz CPU and 8GB memory. Only a single thread is used without involving any SIMD instructions. Our system is implemented using C++. The following evaluation is on 1000+ natural images.

Table 1 shows the execution time of our algorithm, which is almost linear to the window and image sizes. For QVGA-size images, the performance is near realtime ($\approx$ 28 fps).

Image\Window	10×10	50×50	100×100
320×240	0.036s	0.078s	0.114s
640×480	0.095s	0.208s	0.331s
1024×768	0.223s	0.462s	0.724s
1920×1080	0.450s	1.013s	1.667s

Table 1. Execution time with respect to different image sizes and window sizes. The input is a 8-bit single-channel image and the feature is its intensity, i.e., $N_I = 256$ and $N_F = 256$.

Algorithms	Time	Gaussian	Reciprocal	Cosine
Brute-force	90.7s	Exact	Exact	Exact
Constant [15]	23.4s	38.76dB	38.57dB	32.69dB
Ours	0.9s	44.14dB	44.36dB	58.65dB

Table 2. Execution time to filter one-megapixel RGB images with a 20×20 kernel and average PSNRs using different weight forms. "Gaussian", "Reciprocal", and "Cosine" represent weights defined as $\exp\{-\|\mathbf{f}(p) - \mathbf{f}(q)\|\}$, $\|\mathbf{f}(p) - \mathbf{f}(q)\|^{-1}$, and $\frac{\mathbf{f}(p)\cdot\mathbf{f}(q)}{\|\mathbf{f}(p)\|\|\mathbf{f}(q)\|}$ respectively. The C++ code for [15] is provided by the authors. For our method, the feature set contains 256 clustered RGB colors.

	Step 1	Step 2	Time Saving
Joint	156.9s	0.4s	-
Joint + MT	2.2s	0.5s	98.28%
Joint + NT	3.2s	0.5s	97.65%
Joint + MT + NT	0.3s	0.6s	99.43%

Table 3. Efficiency of median tracking and necklace table. We filter one-megapixel RGB images with a 20×20 kernel. Joint, MT, and NT are shorts for joint-histogram, median tracking, and necklace table.

We compare running time of our algorithm, constant time WMF [15], and brute-force implementation in Table 2. For a one-megapixel input, our algorithm is much faster than others. Note the method of [15] is a constant-time one. Since our algorithm has a much smaller constant, it is more efficient even if the window size is unusually large. When the kernel size r is larger than 100, our algorithm is 1000+ times faster than the brute-force implementation. Moreover, our weight can be set differently as shown in Table 2.

The median tracking (MT) (Sec. 5) and necklace table (NT) (Sec. 6) strategies save a lot of time in the joint-histogram traversal step. In Table 3, we show these parts save similar amounts of time ($\approx$ 98%) when performed separately. Their combination is the even most efficient, as the overall running time is reduced by 99.4%. This is a notably large percentage.

L_1-norm and Sparse-norm Filter Weighted median serves as a basic tool in solving popular energy minimization problems. There is an equivalence, proved in [14], between the weighted median operator in Eq. (2) and a

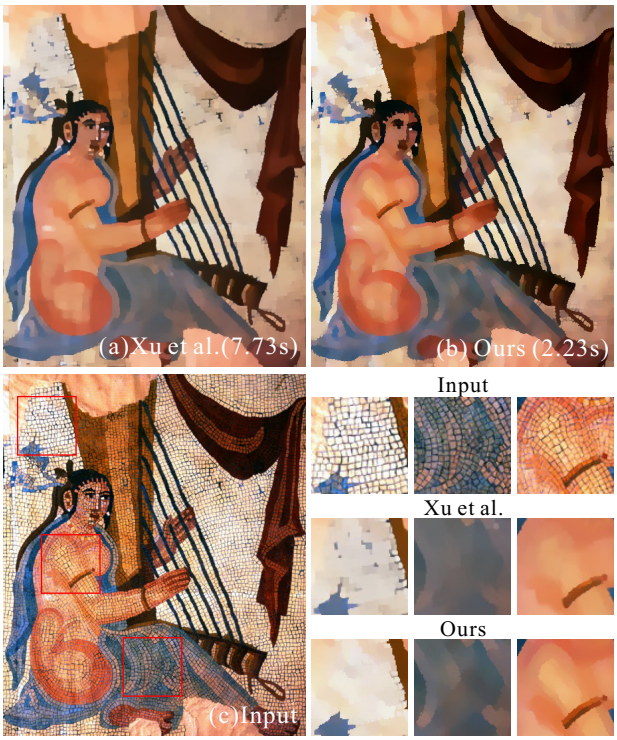


Figure 7. Our fast WMF can be used as L_0-norm filter ($\alpha = 0$ in Eq. (8)) to remove details while preserving large-magnitude structures better than global minimization [23]. (a) Global minimization in 3 iterations uses 7.73s; (b) Our filtering process in 10 iterations uses 2.23s.

global energy function defined as

$$\min_{I^*} \sum_{q \in \mathcal{R}(p)} w_{pq} \|I^*(p) - I(q)\|, \tag{7}$$

where I^* is the output image or field. The commonly employed L_1 minimization expressed by Eq. (7) manifests the vast usefulness of weighted median filter.

Further, minimization with higher-sparsity norms L_α with $0 \leq \alpha < 1$ like

$$\min_{I^*} \sum_{q \in \mathcal{R}(p)} w_{pq} \|I^*(p) - I(q)\|^\alpha, \tag{8}$$

can also be accomplished in the form of reweighted L_1. The details of reweighted L_1 were discussed in [3]. Our fast WMF with its weight defined as $w_{pq}\|I(p) - I(q)\|^{\alpha-1}$ can be used to solve these functions. We shown in Fig. 7 a L_0-norm smoothing result using our WMF method. It better preserves structures than the global minimization approach presented in [23]. Our method speed is higher. More image filtering results are put into the project website.

Non-local Regularizer WMF can be taken as a non-local regularizer in different computer vision systems. A

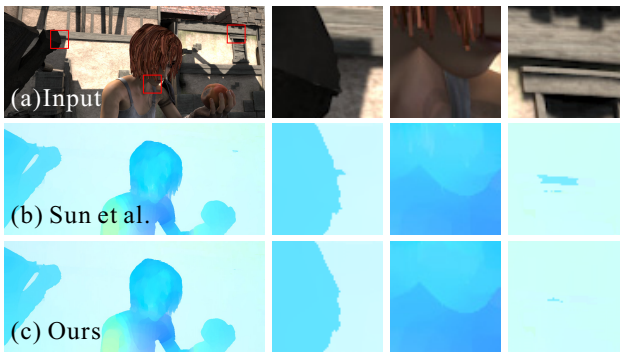


Figure 8. Estimating optical flow by our WMF. (a) Input and close-ups. (b) Result of [19]. (c) Our result by replacing global optimization by our local WMF. Our implementation uses only RGB color as features. The quality is similar but the running time is much shortened.

Figure 9. The watercolor painting effect generated with weights defined as Jaccard similarity in RGB color.

common set of optimization problems involving nonlocal total variation regularizer can be expressed as

$$\sum_{p} \sum_{q \in \mathcal{R}(p)} w_{pq} \cdot \|I(p) - I(q)\|. \tag{9}$$

It finds optimal solutions by performing WMF (see [19] for more details). Taking optical flow estimation as an example, our revolutionized WMF shortens running time from the original 850 seconds to less than 30 seconds. Results for one example are shown in Fig. 8.

Varying Weights And Features Not limited to Gaussian weights, our framework can handle arbitrary weight definition based on features. One example is to use RGB color as the feature and define the weight w_{pq} as a Jaccard similarity [13]:

$$\frac{\min(R(p), R(q)) + \min(G(p), G(q)) + \min(B(p), B(q))}{\max(R(p), R(q)) + \max(G(p), G(q)) + \max(B(p), B(q))},$$

where $R(p)$, $G(p)$, and $B(p)$ are the RGB channels of $\mathbf{f}(p)$. Even for this special weight, optimal solution can be

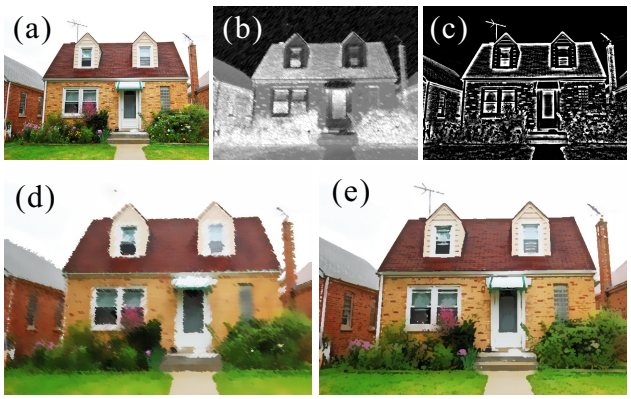


Figure 10. Cross-median filter. (d) and (e) are obtained by filtering (a) with feature maps (b) and (c) respectively. The filter smooths structure according to different features, creating the style-transfer effect.

obtained, which generates a watercolor painting effect, as shown in Fig. 9.

Further, our framework supports cross/joint weighted median filtering. Thus features of others image can be taken as special guidance. A "style-transfer" example is shown in Fig. 10.

8. Concluding Remarks

Weighted median is an inevitable and practical tool in filter and energy minimization for many applications. We much accelerated it in a new framework with several effective techniques. We believe it will greatly profit building efficient computer vision systems in different domains.

Acknowledgements

We thank Qiong Yan for the help in coding and algorithm discussion. This work is supported by a grant from the Research Grants Council of the Hong Kong SAR (project No. 413110) and by NSF of China (key project No. 61133009).

References

[1] A. Adams, J. Baek, and M. A. Davis. Fast high-dimensional filtering using the permutohedral lattice. *Computer Graphics Forum*, 29(2):753–762, 2010.

[2] A. Adams, N. Gelfand, J. Dolson, and M. Levoy. Gaussian kd-trees for fast high-dimensional filtering. *ACM Transactions on Graphics (TOG)*, 28(3):21, 2009.

[3] E. J. Candes, M. B. Wakin, and S. P. Boyd. Enhancing sparsity by reweighted l1 minimization. *Journal of Fourier Analysis and Applications*, 14(5-6):877–905, 2008.

[4] J. Chen, S. Paris, and F. Durand. Real-time edge-aware image processing with the bilateral grid. *ACM Transactions on Graphics (TOG)*, 26(3):103, 2007.

[5] D. Cline, K. B. White, and P. K. Egbert. Fast 8-bit median filtering based on separability. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 5, pages V–281. IEEE, 2007.

[6] F. Durand and J. Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Transactions on Graphics (TOG)*, 21(3):257–266, 2002.

[7] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Trans. Graph.*, 27(3), 2008.

[8] E. S. Gastal and M. M. Oliveira. Domain transform for edge-aware image and video processing. *ACM Transactions on Graphics (TOG)*, 30(4):69, 2011.

[9] E. S. Gastal and M. M. Oliveira. Adaptive manifolds for real-time high-dimensional filtering. *ACM Transactions on Graphics (TOG)*, 31(4):33, 2012.

[10] K. He, J. Sun, and X. Tang. Guided image filtering. In *Computer Vision–ECCV 2010*, pages 1–14. Springer, 2010.

[11] T. S. Huang. *Two-dimensional digital signal processing II: transforms and median filters*. Springer-Verlag New York, Inc., 1981.

[12] M. Kass and J. Solomon. Smoothed local histogram filters. *ACM Transactions on Graphics (TOG)*, 29(4):100, 2010.

[13] M. Levandowsky and D. Winter. Distance between sets. *Nature*, 234(5323):34–35, 1971.

[14] Y. Li and S. Osher. A new median formula with applications to pde based denoising. *Commun. Math. Sci*, 7(3):741–753, 2009.

[15] Z. Ma, K. He, Y. Wei, J. Sun, and E. Wu. Constant time weighted median filtering for stereo matching and beyond. In *ICCV*. IEEE, 2013.

[16] S. Paris and F. Durand. A fast approximation of the bilateral filter using a signal processing approach. In *Computer Vision–ECCV 2006*, pages 568–580. Springer, 2006.

[17] S. Perreault and P. Hébert. Median filtering in constant time. *Image Processing, IEEE Transactions on*, 16(9):2389–2394, 2007.

[18] F. Porikli. Constant time o (1) bilateral filtering. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[19] D. Sun, S. Roth, and M. J. Black. Secrets of optical flow estimation and their principles. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2432–2439. IEEE, 2010.

[20] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998.

[21] B. Weiss. Fast median and bilateral filtering. *ACM Transactions on Graphics (TOG)*, 25(3):519–526, 2006.

[22] L. Xu, C. Lu, Y. Xu, and J. Jia. Image smoothing via L0 gradient minimization. *ACM Trans. Graph.*, 30(6), 2011.

[23] L. Xu, Q. Yan, Y. Xia, and J. Jia. Structure extraction from texture via relative total variation. *ACM Transactions on Graphics (TOG)*, 31(6):139, 2012.

[24] Q. Yang, K.-H. Tan, and N. Ahuja. Real-time o (1) bilateral filtering. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 557–564. IEEE, 2009.