Tree Filtering: Efficient Structure-Preserving Smoothing With a Minimum Spanning Tree

Linchao Bao, Yibing Song, Qingxiong Yang, *Member, IEEE*, Hao Yuan, and Gang Wang, *Member, IEEE*

***Abstract*—We present a new efficient edge-preserving filter – "tree filter" – to achieve strong image smoothing. The proposed filter can smooth out high-contrast details while preserving major edges, which is not achievable for bilateral-filter-like techniques. Tree filter is a weighted-average filter, whose kernel is derived by viewing pixel affinity in a probabilistic framework simultaneously considering pixel spatial distance, color/intensity difference, as well as connectedness. Pixel connectedness is acquired by treating pixels as nodes in a minimum spanning tree (MST) extracted from the image. The fact that a MST makes all image pixels connected through the tree endues the filter with the power to smooth out high-contrast, fine-scale details while preserving major image structures, since pixels in small isolated region will be closely connected to surrounding majority pixels through the tree, while pixels inside large homogeneous region will be automatically dragged away from pixels outside the region. The tree filter can be separated into two other filters, both of which turn out to have fast algorithms. We also propose an efficient linear time MST extraction algorithm to further improve the whole filtering speed. The algorithms give tree filter a great advantage in low computational complexity (linear to number of image pixels) and fast speed: it can process a 1-megapixel 8-bit image at around 0.25 seconds on an Intel 3.4GHz Core i7 CPU (including the construction of MST). The proposed tree filter is demonstrated on a variety of applications.**

***Index Terms*—bilateral filtering, collaborative filtering, edge-preserving smoothing, high-contrast detail smoothing, joint filtering, minimum spanning tree, structure-preserving smoothing, tree filtering.**

I. Introduction

Edge-preserving image smoothing has been serving as the foundation for many computer vision and graphics applications. Real-world natural images are often filled with various trivial details and textures, which may degrade the performance of many computer vision and graphics algorithms including, for example, low-level image analysis (e.g., edge detection, image segmentation), image abstraction/vectorization for visual effects or compact storage, content-aware image editing, etc. Serving as the pre-processing or key intermediate step for these algorithms, edge-preserving smoothing is to remove trivial details (smoothing) while respecting major image structures (edge-preserving).

Most of the existing edge-preserving smoothing operators distinguish details from major image structures based on pixel color/intensity differences. One of the most representative operator is the well-known bilateral filter [1], which averages nearby similar pixels to filter each pixel. Other similar operators include anisotropic diffusion [2], weighted least square (WLS) filter [3], edge-aware wavelets [4], guided filter [5], geodesic smoothing [6], [7], domain transform filter [8], local Laplacian filter [9], L_0 smoothing [10], etc. Although the filtering responses of these operators differ from each other, the common behavior of such kind of operators is to smooth out *low-contrast* details from input images as they typically only use pixel color/intensity contrasts (or image gradients) to distinguish details from major image structures. We refer these operators as *bilateral-filter-like* techniques in this paper.

Bilateral-filter-like techniques find their successful places in many applications, especially where low-contrast details need to be enhanced [11], [3], [12]. For other applications where *high-contrast* trivial details need to be smoothed (one example is the scene simplification task), however, such kind of techniques are often not wise choices.

A family of *local-histogram-based* filters [13], [14], [15] (e.g., median filter and local mode filters) address this problem by analyzing local pixel population within the sliding window, whose main idea is to replace the color/intensity of each pixel with the color/intensity of neighboring majority pixels (e.g., using some certain robust statistics drawn out from local histogram). Such kind of operators can smooth out high-contrast, fine-scale details, but they often face a problem of serious deviation from the original sharp edges (especially at corners) since local histogram completely ignores image geometric structures.

Subr et al. [16] explicitly point out that details should be identified with respect to spatial scale, regardless of their color/intensity contrasts. They propose to smooth out high-contrast, fine-scale oscillations by constructing local extremal envelopes. Recently, Xu et al. [17] propose to extract major structures from textured images based on Relative Total Variation (RTV) in an optimization framework. Su et al. [18] try to combine the strong smoothing ability of traditional low-pass filter and the edge-preserving ability of bilateral filter, in order to smooth out high-contrast textures. All these novel methods intend to address the problem of smoothing out details with high contrasts while respecting major image structures, but they all require either solving large linear systems or more complex optimization techniques, which prevent them from

Manuscript received January 26, 2013; revised July 3, 2013 and August 29, 2013; accepted November 5, 2013. This work was supported by a GRF grant from the Research Grants Council of Hong Kong under Grant U 122212. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Sina Farsiu.

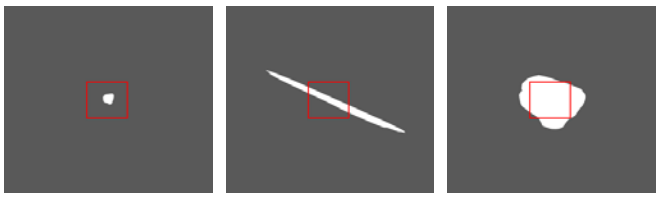
L. Bao, Y. Song, and Q. Yang are with the Department of Computer Science at City University of Hong Kong, Hong Kong (e-mail: {linchaobao, dynamicstevenson}@gmail.com; qiyang@cityu.edu.hk).

H. Yuan is with the BOPU Technologies, Shenzhen, China (e-mail: hao@bopufund.com).

G. Wang is with the School of Electrical and Electronics Engineering, Nanyang Technological University, Singapore and Advanced Digital Science Center, Singapore (e-mail: wanggang@ntu.edu.sg).

serving as an efficient filtering tool in many applications. Detailed analysis and comparison are provided in Sec. II-A2 and Sec. V-A, respectively.

We hereby reexamine the definition of the notion "detail" before we present the tree filter. We agree with Subr et al. [16] that "details" should be distinguished from major image structures by their spatial scales, rather than by their contrasts. However, we notice that a reliable method for distinguishing between different "spatial scales" in 2D discrete signal space worth further discussion. Specifically, unlike 1D signal space in which it is easy to identify fine-scale details, in 2D signal space, simple method for identifying fine-scale details (e.g., using a sliding window) will fail since slender (thin and long, see Fig. 1(b)) structures might be lost. We argue that if a connected component in an image is large enough (even if it is slender), it should be considered as an important image structure thus need to be preserved (see Fig. 1). (Note that the discussion for accurate definition of "connected component" is out of the scope of this paper, and we only use the concept of "connected component" to refer to homogeneous image region containing pixels with similar colors/intensities.)



(a) Small region (b) Slender region (c) Large region

Fig. 1. Three cases of homogenous image region (red square stands for a sliding window). It is easy to identify region (a) as detail to be smoothed and (c) as major structure to be preserved. For region (b), simple approach that only looks at a local window of each pixel will identify it as detail part by part. We argue that region (b) is large enough as a whole to be identified as major structure and thus should be preserved.

In this paper, we present a new edge-preserving smoothing filter – "tree filter" – for smoothing out high-contrast details while preserving major image structures. Compared with previous complex operators for smoothing high-contrast details, tree filter is a simpler yet effective weighted-average filter and can be computed much more efficiently using proposed algorithms. It utilizes a minimum spanning tree (MST) extracted from input image to determine the weights of the filter kernel. The MST enables a non-local fashion of distinguishing small connected components (details) from large connected components (major structures), thus tree filter is able to deal with the slender region case in Fig. 1(b). Moreover, tree filter can be separated into two other filters, both of which turn out to have fast algorithms. We also propose an efficient linear time MST extraction algorithm to further improve the whole filtering speed. The algorithms give tree filter a great advantage in low computational complexity (linear to number of image pixels) and fast speed: it can process a 1-megapixel 8-bit image at around 0.25 seconds on an Intel 3.4GHz Core i7 CPU (including the construction of MST). The speed advantage makes tree filter a practical filtering tool for many applications.

II. Preliminaries and Related Work

In this section, we provide some basic concepts and notions, as well as a brief review of related work.

A. Edge-preserving Smoothing

1) Bilateral-filter-like Techniques: Bilateral filter (BLF) [1] is an image-dependent, weighted-average filter in which the weight is determined by both pixel spatial distance and color/intensity difference. Specifically, for each pixel i in image I, the bilateral filtered output B_i is computed by

$$B_i = \sum_{j \in \Omega} b_i(j) I_j, \tag{1}$$

where Ω is the set of all pixels in the entire image and $b_i(j)$ is the *bilateral weight* of pixel j contributing to i. The bilateral weight $b_i(j)$ is calculated by

$$b_i(j) = \frac{G_{\sigma_s}(\|i-j\|) G_{\sigma_r}(\|I_i - I_j\|)}{\sum_{p \in \Omega} G_{\sigma_s}(\|i-p\|) G_{\sigma_r}(\|I_i - I_p\|)}, \tag{2}$$

where the spatial weighting function $G_{\sigma_s}(x)$ and color/intensity (a.k.a. *range*) weighting function $G_{\sigma_r}(x)$ are typically 2D Gaussian functions with variances σ_s and σ_r, respectively. Note that although Ω covers the entire image, far pixels from i will have weights approximately to zero due to spatial Gaussian kernel.

If we use another guidance image $\tilde{I}$ instead of the original image I to calculate the range weighting kernel, the filter becomes

$$\tilde{B}_i = \sum_{j \in \Omega} \tilde{b}_i(j) I_j, \tag{3}$$

$$\tilde{b}_i(j) = \frac{G_{\sigma_s}(\|i-j\|) G_{\sigma_r}(\|\tilde{I}_i - \tilde{I}_j\|)}{\sum_{p \in \Omega} G_{\sigma_s}(\|i-p\|) G_{\sigma_r}(\|\tilde{I}_i - \tilde{I}_p\|)}, \tag{4}$$

which is often called joint bilateral filter [19].

Bilateral filter is widely used for its simplicity and effectiveness in many applications [20]. However, its brute force implementation is very slow. There are many accelerated versions utilizing quantization and/or downsampling techniques [21], [22], [23], [24], [25], [26], which can achieve rather fast speed. Specially, when a constrained range filter is used, bilateral filter can be implemented recursively thus can achieve an extremely fast speed [27]. Besides, some other fast edge-preserving filters try to achieve similar filtered results to bilateral filter using new approaches, e.g., linear regression based method [5], geodesic distance transform based method [7], domain transform based method [8], adaptive manifold based method [28]. For example, the fastest method is reported by the adaptive manifold paper [28], which can process 10-megapixel color image at around 50fps on modern GPU. Similar to the bilateral filter, these methods are not designed to smooth out fine scale details with high intensity contrasts.

In order to avoid the artifacts introduced by edge blurring or edge sharpening in image edge-preserving decomposition applications, Farbman et al. [3] propose an edge-preserving filtering method based on weighted least square (WLS) optimization, whose objective function is regularized by image

gradients. The main idea of their method is to force the filtered results at regions where gradient is large to be as close as possible to the input image, but that at other regions to be smoothed. Fattal [4] achieves a very fast speed for edge-preserving decomposition using a novel edge-avoiding wavelets approach, but the filtered results commonly seem noisy and are not satisfactory for most applications. Paris et al. [9] propose a technique to perform edge-preserving filtering based on local Laplacian pyramid manipulation and also show their method can avoid artifacts over edges. A recent accelerated version of the filter [29] utilizing downsampling and interpolation techniques makes it become a practical and ideal choice for some applications, such as HDR tone mapping, to generate artifact-free results. These methods can effectively avoid edge blurring or sharpening which may be introduced by bilateral filter, but they are not designed to smooth out high-contrast, fine-scale details since they are commonly based on image contrasts or gradients.

Besides, Xu et al. [10] proposed an edge-preserving smoothing method based on a global optimization on the L_0 norm of image gradients (i.e., counting gradient jumps) to produce piecewise constant images. The method can filter input signals into staircase-like signals and thus achieve an impressive, strong smoothing effect. Since it is also based on image gradients, it will preserve high-contrast, fine-scale details.

Although the filtering responses of the above operators differ from each other, the common behavior of these operators is to smooth out *low-contrast* details from input images as they typically only use pixel color/intensity contrasts (or image gradients) to distinguish details from major image structures. We refer these operators as *bilateral-filter-like* techniques in this paper. Note that our proposed tree filter is not designed to behave like such operators.

2) High-contrast Detail Smoothing: In order to smooth out high-contrast, fine-scale details from images, local-histogram-based filters [13], [14], [15] attempt to solve it by looking into the distribution of neighboring pixels around each pixel rather than image contrasts or gradients. The simplest, well-known example is the median filter, which is to replace each pixel with the median of its neighboring pixels. More robust smoothing can be achieved by using other robust statistics such as *mode* instead of median. For example, closest-mode filter is to replace each pixel with the closest mode to center pixel in smoothed local histogram, and the dominant-mode filter is to instead use the mode having the largest population (not related to center pixel) [15]. Although mode filters can generally produce more smoothing results with sharp edges, they often face a problem of serious deviation from the original edges (especially at corners) since local histogram completely ignores image geometric structures.

Subr et al. [16] propose a method for smoothing out high-frequency signal oscillations, regardless of their contrasts, by constructing local extremal envelopes. The envelopes are constructed by first locating image local extremal points using sliding window and then computing interpolation between local extremal points using weighted least square minimization. After constructing a maximal envelope and a minimal envelope, respectively, for each image, the output is computed as the average of the two envelopes. The simple strategies employed by their method make it suffer from several weaknesses when filtering natural images. First, using sliding window to locate local extrema makes the method sensitive to irregular high-frequency textures or details (see Fig. 8(f)). Second, it will falsely remove slender significant regions due to the sliding window, as described in Sec. I. Third, the averaging between extremal envelopes often leads to results with considerably shifted colors/intensities (e.g., the results presented in their paper commonly seem brighter than input images).

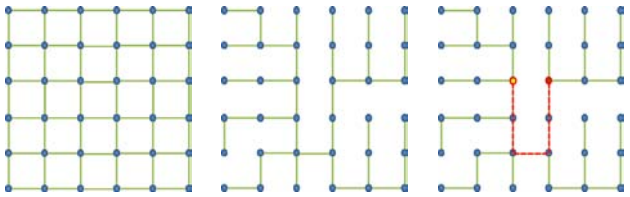
Xu et al. [17] design a novel local variation measure, namely Relative Total Variation (RTV), to distinguish textures from major image structures regardless of contrasts, and propose to perform smoothing in an optimization framework. The RTV is designed based on their key observation that the aggregation result of signed gradient values in a local window often has a larger absolute value for major edges than for textures, since the gradients for textured region are usually inconsistent within a local window and the aggregation will counteract each other. Their method can produce impressive results for highly textured images (such as mosaic images or graffiti on textured materials), but it may overly smooth natural images.

Su et al. [18] strive to construct a special guidance image and then use it to perform joint bilateral filtering on the input image to achieve strong smoothing. The guidance image is constructed by performing a low-pass filtering on input image followed by an edge sharpening step using L_0 smoothing [10]. However, the solution strongly relies on the L_0 smoothing technique to compensate for edge loss due to low-pass filtering (in the edge sharpening step), which is brittle and may not work well in many cases. Besides, the whole pipeline involves too many parameters and is sensitive to parameter choice in each step, thus in practice it is hard to tune parameters to produce satisfactory results.

B. Minimum Spanning Tree for Image

By treating an image as a standard $4-$connected, undirected grid (planar graph) with nodes being all the image pixels and edges between nearest neighboring pixels being weighted by color/intensity differences, a minimum spanning tree (MST) can be computed by removing edges with large weights (Kruskal algorithm) [30], leaving the remaining edges connecting through all pixels as a tree (see Fig. 2). The MST and related algorithms can be found in many image processing tasks, e.g., segmentation [31], [32], denoising [33], abstraction [34]. In this paper, we address the problem of efficient image smoothing for high-contrast details. We use the notion *tree distance* to refer to the length of the path between two nodes on the tree (letting the distance between neighboring nodes be 1). For example, the tree distance between the two marked nodes in Fig. 2(c) is 5.

The MST extracted from image has an important property which makes the tree distance be an edge-aware metric: MST can automatically drag away two dissimilar pixels that are close to each other in the spatial domain (see Fig. 2(c)). More importantly, small isolated region surrounded by large homogeneous region with dissimilar color/intensity (see Fig.



(a) Planar graph (b) MST (c) Tree distance

Fig. 2. Illustration of a MST from image. (a) a planar graph in which nodes are image pixels and edges are with costs weighted by color/intensity differences between neighboring pixels. (b) a MST extracted from the planar graph, in which edges with large costs are removed during the MST construction. (c) the tree distance between the two pixels on the MST is 5.

(a) Small region (b) Slender region (c) Large region

Fig. 3. MST rank maps for images in Fig. 1. The rank value of each pixel is its layer number in the tree (from tree root). Brighter color in the rank map indicates larger rank value. (The top-left image pixel is the root node of the MST.)

1(a)) will be connected to the surrounding region with a short tree distance during the MST construction (because MST ensures that all image pixels should be connected together through the tree). On the other hand, if the isolated region is large enough (see Fig. 1(b) and 1(c)), most of the pixels inside it will be connected to the surrounding region with large tree distances. This can be illustrated by visualizing the MST *rank map* (a *rank* value of a node refers to its layer number from root node) corresponding to each of the above cases (note that although tree distance is not the same as rank difference, the rank map can serve as a good visualization tool for inspecting tree distance). From the rank maps (Fig. 3) corresponding to the images in Fig. 1 we can see that, both the slender region and the large region can be easily identified from the rank map (which means pixels inside the regions have large rank differences, i.e., large tree distances, to pixels outside the regions), while the small isolated region can hardly be found on the rank map (which means the rank differences between pixels inside the region and pixels outside the region are not significant). Although smaller rank difference does not necessarily mean smaller tree distance, it is often the case for pixels that are near to each other in image spatial domain (which is exactly the case for small isolated regions).

One obvious problem of the MST is that there might be some "false edges" introduced, which can be easily notified at the right side of the large region in Fig. 3(c). Be aware that although the rank values of pixels at the right side are similar to that of some pixels inside the region, the tree distances between them are actually not that short. The actual problem is that the tree distance from the downside pixel to the upside pixel is large, but in fact, they are similar and close to each other in the original image. The same problem will happen on a constant image, where any two neighboring pixels that are expected to be close to each other might have arbitrarily far distance on the tree.

Another subtle yet notable problem of the MST is the "leak" problem, which can be found in a close inspection (e.g., in Fig. 3(c), the "leak" happens at the bottom of the region). Since the MST forces every pixel to eventually be connected through the tree, even an isolated region with hard edges has to contain at least one bridge to the rest of the image, through which the nearby dissimilar pixels may have short tree distances. Another case when "leak" may happen is near blurry edges, where there is gradual transition between dissimilar colors/intensities.

Therefore, in order to utilize MST to perform edge-preserving smoothing, pixel spatial distance and color/intensity difference beside tree distance need to be involved. We will address these problems in the proposed tree filter.

III. Tree Filter

We now present the tree filter, a weighted-average filter that can smooth out high-contrast details while preserving major image structures.

A. Motivation

As described in the previous section, tree distance on MST can serve as an edge-aware metric for (inversely) measuring pixel *affinity*[1] which can distinguish small isolated region from large homogeneous region, except that it often faces the "false edge" and "leak" problems. Inspired by the idea of *collaborative filtering* [35][2] commonly used in recommendation systems, which is to make predictions about the interests of a user by collecting preferences of other users having similar tastes, we can collaboratively solve the problems by consulting nearby similar pixels.

Specifically, suppose a pixel i is located at the "leak" point of a large homogeneous region, it may have a short tree distance to a dissimilar pixel j outside the region, which means there is a strong affinity between i and j by simply measuring tree distance. However, this is not what we want since we hope their affinity to be weak in order to keep the main image structure. Here comes the solution: if pixel i asks many other nearby similar pixels, denoted as ks, whether each of them has a short or long tree distance to j, and then combines all the answers together to make its final decision – whether it has a weak or strong affinity to j, the result will be more reliable. Since i is inside the large homogeneous region, there will be many similar ks nearby, many of which should have large distances to j (because they are not "leak" point). Thus the final decision will probably be "weak". Consider another case when pixel i is located at a small isolated region (Fig. 1(a)), nearby similar ks will also have short distances to j, hence the final decision of whether the affinity between i and j is weak or strong will be "strong". For the "false edge" problem, the scenario is similar.

Based on the above idea, we next define the tree filter, and then interpret it intuitively by viewing pixel affinity in

[1]In this paper, we use *affinity* to refer to the desired impact that two pixels exert on each other when performing edge-preserving smoothing. Stronger affinity means greater impact.

[2]Note the concept *collaborative filtering* here is not the same as that in BM3D denoising algorithm [36].

a probabilistic framework simultaneously considering pixel spatial distance, color/intensity difference, as well as tree distance.

B. Definition

We define the tree filter as follows. For each pixel i in image I, the tree filtered output S_i is computed by

$$S_i = \sum_{j \in \Omega} w_i(j) I_j, \tag{5}$$

where Ω is the set of all pixels in the entire image and $w_i(j)$ is the *collaborative weight* of pixel j contributing to i. The collaborative weight $w_i(j)$ is calculated by

$$w_i(j) = \sum_{k \in \Omega} b_i(k) t_k(j), \tag{6}$$

where Ω is again the set of all pixels in the entire image and $b_i(k)$ and $t_k(j)$ are the bilateral weight and the *tree weight*, respectively. The bilateral weight $b_i(k)$ is the same as that defined in Eq. (2), which is used for selecting nearby similar pixels ks (the weight is attenuated with the increase of either spatial or range distance between i and k). The tree weight $t_k(j)$ is determined by the tree distance from k to j (denoted as $D(k, j)$):

$$t_k(j) = \frac{F_\sigma(D(k,j))}{\sum_{q \in \Omega} F_\sigma(D(k,q))}, \tag{7}$$

where $F_\sigma(x)$ is a falling off exponential function controlled by parameter σ:

$$F_\sigma(x) = \exp\left(-\frac{x}{\sigma}\right). \tag{8}$$

Claim *The sum of all collaborative weights for a particular pixel i is* 1.

Proof

$$\begin{aligned} \sum_{j \in \Omega} w_i(j) &= \sum_{j \in \Omega} \sum_{k \in \Omega} b_i(k) t_k(j) = \sum_{k \in \Omega} \sum_{j \in \Omega} b_i(k) t_k(j) \\ &= \sum_{k \in \Omega} b_i(k) \sum_{j \in \Omega} t_k(j) = \sum_{k \in \Omega} b_i(k) \cdot 1 = 1. \quad \square \end{aligned}$$

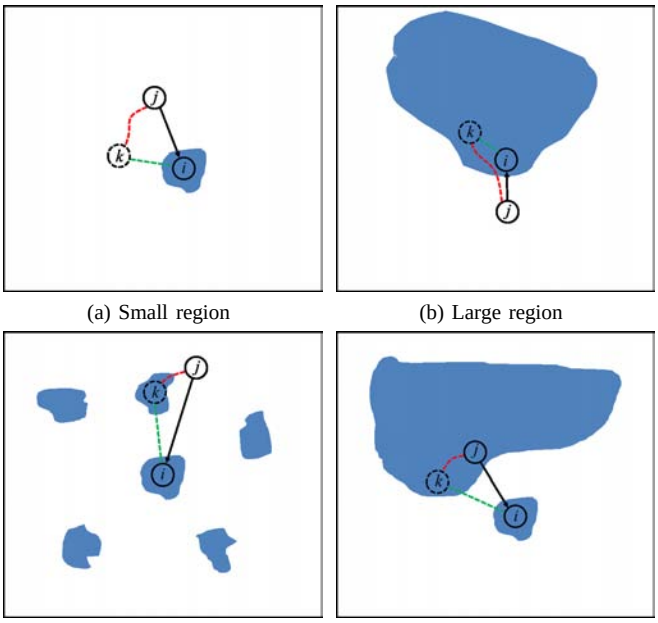
C. Explanation

The definition shows that tree filter is a weighted-average filter. The weight of a pixel j contributing to pixel i, namely collaborative weight $w_i(j)$, can be easier to understand if we view it in a probabilistic framework. If we consider the weight $w_i(j)$ as the probability of pixel j supporting pixel i, denoted as $p(j)$, then it can be formulated using *mixture model* as follows (we do not mean to estimate a mixture model but just use the concept to understand the weight $w_i(j)$). We take each of the pixel k in the image as one component of the mixture, whose probability $p(k)$ is measured by the similarity (both spatial and range) between pixel k and i. The conditional probability of pixel j belonging to each component k, denoted as $p(j|k)$, is determined by the tree distance from pixel j to pixel k (the farther tree distance, the lower probability). Then the probability of pixel j supporting pixel i, $p(j)$, can be calculated by probability marginalization

$$p(j) = \sum_{k \in \Omega} p(j|k) p(k),$$

which is exactly the same form as Eq. (6).

The reason why tree filter is able to smooth high-contrast details and preserve major image structures (including large homogeneous regions and slender regions that contain sufficient connected pixels) can be intuitively explained as follows.



(a) Small region (b) Large region

(c) Textured region (d) Failure case

Fig. 4. Several cases when calculating collaborative weight $w_i(j)$ (black arrow). The green dash line stands for bilateral weight $b_i(k)$ and red dash line stands for tree weight $t_k(j)$. Note that k should run through all pixel locations in the image while calculating the $w_i(j)$ of one specific j.

Case 1 (Fig. 4(a)): "Small isolated region" – pixel i is located at a small isolated region and there is no similar pixel outside the isolated region. Consider the process of filtering pixel i: when calculating $w_i(j)$ for each pixel j, only the ks within the isolated region have large bilateral weights $b_i(k)$, thus $w_i(j)$ is approximately equivalent to the tree weight $t_i(j)$ (i.e., only consider ks located near i). Therefore the tree filtered output for pixel i is

$$S_i = \sum_{j \in \Omega} w_i(j) I_j \approx \sum_{j \in \Omega} t_i(j) I_j. \tag{9}$$

Since tree weight $t_i(j)$ only considers the tree distance on MST, the filtering actually completely ignores pixel contrasts (see Sec. II-B). The effect is just like a traditional low-pass filtering (like Gaussian filtering), which is desired for smoothing details.

Case 2 (Fig. 4(b)): "Large homogeneous region" – consider the critical case that pixel i is located at the "leak" point of the large region. Through comparison, it is easy to understand that a j inside the region has much larger weight $w_i(j)$ than a j outside the region, since the inside j will have much more ks with both higher bilateral weights $b_i(k)$ and tree weights

$t_k(j)$ than the outside j. Therefore the tree filtering for pixel i is a weighted average which gives higher weights to js inside the region and lower weights to js outside the region. In this manner the edge of the region gets preserved. For slender region having sufficient pixels, the case is the same.

Case 3 (Fig. 4(c)): "Textured region" – pixel i is located at a small isolated region and there are similar small isolated regions nearby. In this case, pixels in each small isolated region have short tree distances to surrounding dissimilar pixels. When calculating $w_i(j)$ of any j, the ks located at all isolated regions will have large bilateral weights. Thus a j will have large weight $w_i(j)$ if it has a short tree distance to such ks, no matter whether the j is inside or outside an isolated region. As a result, the tree filtering for pixel i will give large weight to similar pixels at every isolated region and the surrounding dissimilar pixels near every region. In this way, smoothing is achieved regardless of contrasts.

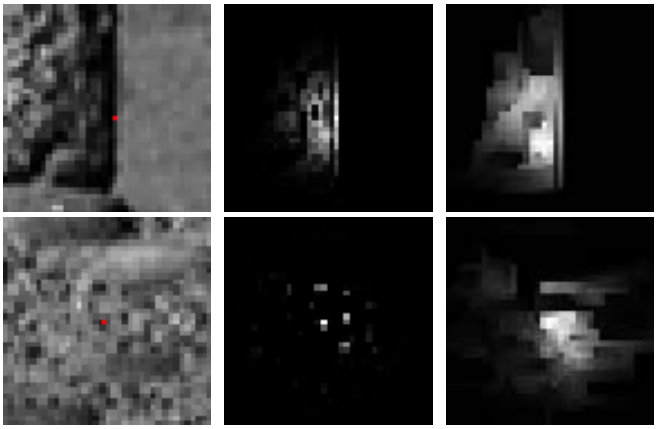
Failure Case (Fig. 4(d)): One failure case is that when pixel i is located at a small isolated region which is near to a large homogeneous region. In this case, the filtering will only average over similar pixels to pixel i (just like case 2) and thus the small isolated region (which we hope to remove) remain there after the filtering (because of the large number of similar pixels in the nearby large region). We will further discuss this problem in Sec. V-B.

D. Filter Kernel

The above explanation can be easier to understand by explicitly plotting the filter kernel for different cases. Fig. 5 shows two examples of the kernel plot for pixels in a real image. For pixel located in large homogeneous region (first row), the tree filter kernel only assigns nonzero weights to nearby similar pixels, just like the bilateral filter kernel (though not the same). For pixel located in textured region (second row), unlike the bilateral filter kernel which only assigns large weights to nearby similar pixels, the tree filter kernel assigns large weights to not only the nearby similar pixels, but also their surrounding pixels (having short tree distances to them). This enables strong smoothing on the textured region, regardless of pixel contrasts.

E. Parameters

Tree filter has three parameters, σ_s, σ_r, and σ, due to the functions for calculating bilateral weights and tree weights, respectively. The σ_s and σ_r control the selection of nearby similar pixels, which are the same as in the bilateral filter. The σ determines the attenuation speed of tree weight as tree distance increases. In this paper, we follow the recent convention of the parameters in bilateral filter [20] (that is, σ_s is measured by pixel number and σ_r is a real number between 0 and 1). Similar to σ_s, σ can also be measured by integer number (since the tree distance is 1 between neighboring nodes). In practice, however, we find that using a real number between 0 and 1 related with image size (i.e., for an image having h by w pixels, we substitute $\sigma \times \frac{1}{2}\min(h, w)$ into the exponential function instead of the original σ to calculate tree weights) is easier to control the amount of smoothing. Thus we present σ in such a manner in this paper.



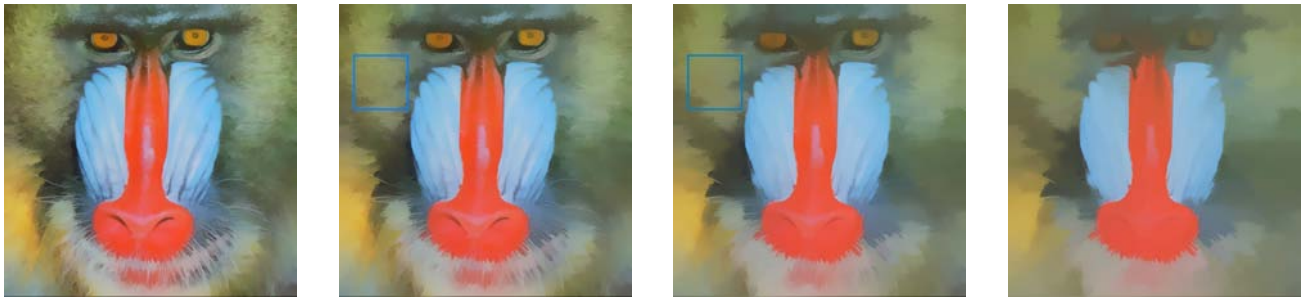
(a) Image patch (b) Bilateral filter (c) Tree filter

Fig. 5. Illustration of filter kernels. The kernels are centered at the pixels denote by red dots. Note that the MST in tree filter is extracted from the original full image (not from the patch itself).

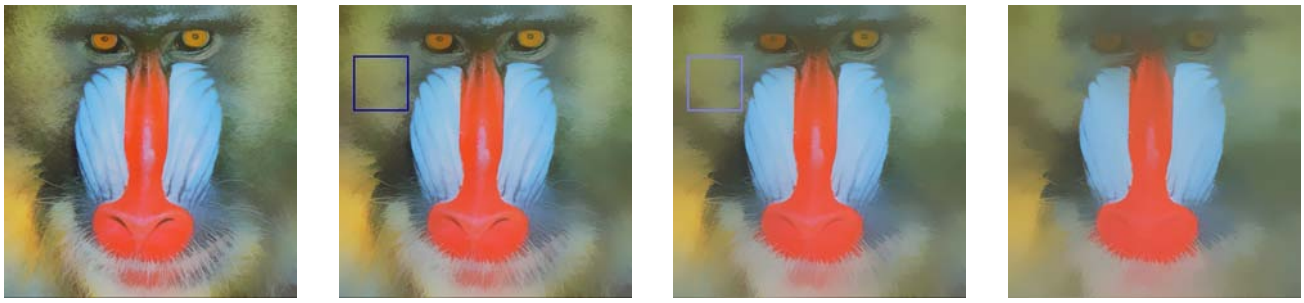
The three-degree-of-freedom parameter tuning seemingly makes it difficult for tree filter to produce satisfactory results. However, in order to produce results with sharp edges, we usually fix σ_r to a small value (typically $\sigma_r = 0.05$) (since we do not want to select dissimilar pixels for collaborative filtering) and adjust σ together with σ_s to achieve different amount of smoothing. Unless otherwise specified, we use $\sigma_r = 0.05$ to produce all the results in this paper.

Fig. 6 shows the tree filtering results of the "baboon" image (Fig. 7(a)) in different parameter settings. With a quick glance from the upper row to the lower row, it is easy to find that, for a certain σ, smaller σ_s tends to yield blocky and sharp results, while larger σ_s will generate smoother results. A closer inspection (Fig. 7(b)) further reveals that smaller σ_s can generally perform well on smoothing out fine-scale, high-contrast details, but may result in "false edges" or "leak" because of fewer pixels participating in the collaborative filtering. Larger σ_s can solve the "false edge" and "leak" problem but may cause details reappear since too many pixels participating in the collaborative filtering will lead to the failure case described in Sec. III-C (details near large homogeneous region preserved). In extreme cases, $\sigma_s = 0$ means no collaborative filtering happens and $\sigma_s = \infty$ means all similar pixels in the entire image will participate into the collaborative filtering. In practice, the parameter tuning for σ often needs to make trade-offs between detail-smoothing and edge-preserving. As described above, with smaller σ_s, the filter's smoothing ability for high-contrast details is strong but it may face "false edge" and "leak" problem. On the other hand, with larger σ_s, the filter can generate results more respecting to original edges, but details may reappear. We find $\sigma_s = 2 \sim 8$ can often produce desired results in practice, according to specific images and applications.

Observation on the filtering results from left to right shows the role of the σ. As σ increases, larger-scale region will be recognized as detail. This is because the σ in the weighting function Eq. (8) controls the falling rate. With a larger σ value, the falling rate becomes slower and pixels with larger tree distance will still be assigned larger tree weights. Thus the

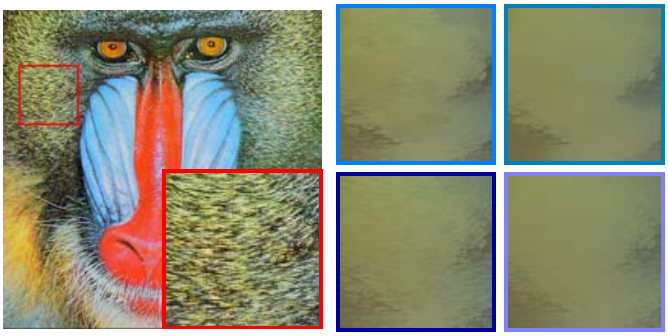


(a) $\sigma = 0.05$, $\sigma_s = 4$ (b) $\sigma = 0.10$, $\sigma_s = 4$ (c) $\sigma = 0.20$, $\sigma_s = 4$ (d) $\sigma = 0.40$, $\sigma_s = 4$



(e) $\sigma = 0.05$, $\sigma_s = 8$ (f) $\sigma = 0.10$, $\sigma_s = 8$ (g) $\sigma = 0.20$, $\sigma_s = 8$ (h) $\sigma = 0.40$, $\sigma_s = 8$

Fig. 6. Effect of tree filtering when varying parameters σ and σ_s (σ_r is fixed to 0.05). Close-ups of the second and third columns are shown in Fig. 7.



(a) "Baboon" image (b) Close-ups of Fig. 6

Fig. 7. The "baboon" image (size 512×512) and close-ups of tree filtering results in Fig. 6 (the second and third columns).

collaborative filtering will involve more dissimilar pixels and pixels inside homogeneous region will have larger chance to be averaged with dissimilar pixels outside the region. However, the side-effect of a too large σ is that the "leak" problem may be more serious. This is analogous to the overly-blurred-edge effect in other low-pass filters (such as Gaussian filter) with aggressively large parameters. To respect the original edges, we usually do not use too large σ value (typically $\sigma = 0.01 \sim 0.20$) in practice.

IV. Fast Implementation

The straightforward implementation of tree filter is very slow, since it requires searching and computing tree distances among all pixels. In this section, we present the fast algorithms for implementing tree filter, which give tree filtering a low computational complexity (linear to pixel number) and a fast speed. For example, it takes about 0.25 seconds for filtering a 1-megapixel 8-bit image on our CPU (Intel 3.4GHz Core i7-2600 CPU with 4GB RAM, using a single core).

A. Separable Implementation

Substituting Eq. (6) into Eq. (5) and rewriting the tree filter kernel, we have

$$S_i = \sum_{j\in\Omega}\sum_{k\in\Omega} b_i(k)t_k(j)I_j = \sum_{k\in\Omega}\sum_{j\in\Omega} b_i(k)t_k(j)I_j \quad (10)$$

$$= \sum_{k\in\Omega} b_i(k)\sum_{j\in\Omega} t_k(j)I_j \stackrel{\text{def}}{=\!=} \sum_{k\in\Omega} b_i(k)T_k, \quad (11)$$

where T_k is computed by

$$T_k = \sum_{j\in\Omega} t_k(j)I_j. \quad (12)$$

Note Eq. (11) is actually a joint bilateral filtering performed on image T (using input image I to calculate bilateral weights), where T is obtained by performing a weighted average (defined by Eq. (12)) on the input image I using tree distance. We here name the weighted average using tree distance as *tree-mean filtering*. Thus the tree filtering actually can be implemented by a tree-mean filtering followed by a joint bilateral filtering.

The direct implementation of tree-mean filtering is still very slow. Fortunately, using the *MST non-local aggregation* algorithm proposed in our recent work [37], the tree-mean filtering can be recursively implemented and achieve a very fast speed. Specifically, substituting Eq. (7) into Eq. (12), we have

$$T_k = \frac{\sum_{j\in\Omega} F_\sigma(D(k,j)) \cdot I_j}{\sum_{q\in\Omega} F_\sigma(D(k,q)) \cdot 1}, \quad (13)$$

where both the numerator and denominator can be computed efficiently using the MST non-local aggregation algorithm , which has a computational complexity linear to the number of image pixels [37]. Note the difference of tree distance definition between this paper and [37]: the length between

neighboring nodes is a constant 1 in this paper, while it is related to color/intensity difference in [37]. Nevertheless, the algorithm in [37] is applicable here. According to our experiments, the whole tree-mean filtering can process 1-megapixel 8-bit image in about 0.05 seconds on our CPU.

The joint bilateral filter has many fast approximation versions, we here employ the simple and fast implementation by our previous work [24], which also has a computational complexity linear to pixel number and can process 1-megapixel 8-bit image in about 0.10 seconds on our CPU (using 8-layer approximation).

B. MST Extraction

Now we present an efficient linear time MST extraction algorithm, specially designed for 8-bit depth image (which may have multiple channels). Let E and V denote the edges and nodes of the MST, respectively. The fastest implementation of Prim's algorithm [38] for building MST requires $O(|E| + |V| \log |V|)$ time using a Fibonacci heap [39]. However, in our case, all possible values of edge weight are integers from 0 to 255 (for multi-channel color images, we use the maximum of color differences among all channels as the edge weight), which allow us to use a priority queue data structure to implement insertion, deletion, and extraction of minimum in constant time.

Specifically, the data structure consists of a bitset[3] and 256 doubly-linked lists. The bitset has a size of 256, and it is used to track what keys are currently in the priority queue. If there is at least one node with key i in the queue, then the bit with position i in the bitset is set to 1, otherwise it is set to 0. The 256 doubly-linked lists are numbered from 0 to 255, where the list i consists of the graph nodes that have a key value of i.

Insertion into this priority queue can be done in constant time by inserting the node into the corresponding list, and setting the corresponding bit in the bitset. Deleting a node is done by removing the node from the corresponding list, and then resetting the corresponding bit in the bitset if the list becomes empty after the deletion. The above insertion and deletion processes are done in constant time in a straightforward manner. Extracting a node with the minimum key value is done by first finding the smallest bit position that is set to 1 in the bitset, where the bit position represents the minimum key value, and then the node can be extracted from the corresponding list in constant time[4].

[3]For example, the `std::bitset` in the GNU C++ Library.

[4]The trick to find the smallest bit position is to call the `_Find_first()` method of `std::bitset` in GNU C++ Library, which runs in $O(256/w)$ time, where w is the bit-length of an integer. The GNU C++'s bitset is implemented using $256/w$ unsigned integers, where each unsigned integer represents w bits. This means that, for a 32-bit program (i.e., $w = 32$), the bitset only visits 8 words in the worst case, and for a 64-bit program (i.e., $w = 64$), it only visits 4 words in the worst case. Each visit invokes a very fast CPU instruction that can find the first bit position with a value 1 in the binary representation of a machine word in constant time. In practice, the keys are usually small, so the search for the first 1-bit can be stopped once the 1-bit is found, without visiting the remaining words (i.e., the unsigned integers). Note that Microsoft Visual C++'s `std::bitset` does not contain a `_Find_first()` method, so we implemented the GNU C++'s bitset by ourselves with the help of `_BitScanForward_` intrinsic (which is used to find the first 1-bit in a word) in Microsoft Visual C++.

Therefore, using the data structure described above, the Prim's algorithm runs in $O(|E| + |V|)$ time. By constraining the input graph to be a $4-$connected, undirected grid, the Prim's algorithm runs in $O(|V|)$, and is linear in the number of nodes in the graph. Thus for 8-bit depth image, a MST can be constructed using the above algorithm in linear complexity. It takes about 0.07 seconds on our CPU to build a MST for a 1-megapixel image (either grayscale or color image).

Since the MST may be easily affected by image noise when dealing with natural image, in practice we suggest to preprocess the input image using a Gaussian filter with small variance (typically 1 pixel) before building a MST from it. The additional Gaussian filtering takes about 0.03 seconds for a 1-megapixel image in our implementation.

V. More Analysis

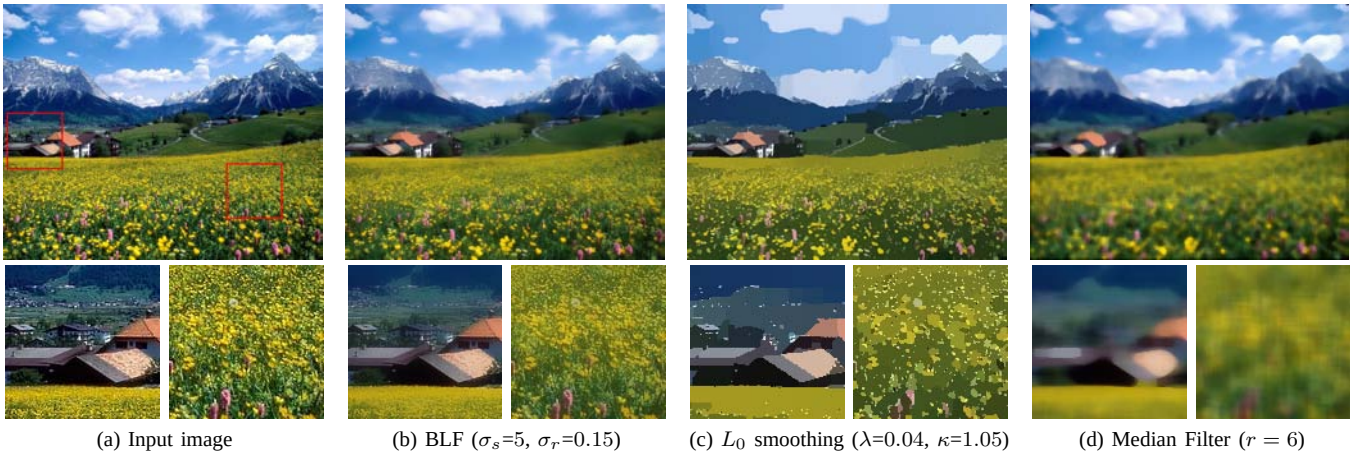
In this section, we provide a comparison of the tree filter to a few other operators addressing high-contrast detail smoothing. The limitation and several potential improvement of the tree filter are also discussed.

A. Comparison

Fig. 8 shows the comparison of edge-preserving smoothing on a "flower farm" image. The flower farm in the image is full of high-contrast details that we want to smooth out. Bilateral-filter-like techniques will commonly fail in this case since they distinguish details by contrasts or gradients (for two representatives, see Figs. 8(b) and 8(c)). The local-histogram-based filters, such as median filter or dominant mode filter [15], which do not depend on center pixel, face a problem of serious deviation from original edges (see left close-up window of Fig. 8(d)) since they completely neglect the geometric information in the image. One exception in the family of local-histogram-based filters is the closest mode filter [15], which depends on the closest mode to center pixel in a local window. The closest mode might change dramatically when sliding a window on irregularly textured regions (such as the flower farm region in the image), hence there are prominent unnatural spots standing out in the output (Fig. 8(e)). The local-extrema-based method proposed by Subr et al. [16] also has this problem (Fig. 8(f)): instead of depending on closest mode, it depends on local extrema.

The recent optimization-based method by Xu et al. [17] can consistently produce high-quality smoothing results for textured images, but since its objective function is regularized by a variation measure (RTV), which is also computed using sliding window, the results may have some deflection near corners (see left close-up windows of Fig. 8(g) and Fig. 10). Moreover, the method relies on solving large sparse linear system and thus its computational cost is high. In our experiments, their Matlab implementation takes about 45 seconds to process a 1-megapixel image (although optimized C++ implementation is expected to be faster, it still takes a few seconds on CPU). In contrast, our tree filter can generate comparable results in a much faster speed (Fig. 8(h)). Fig. 10 shows another two examples of the comparison.

Another recently proposed method that can achieve edge-preserving smoothing regardless of image contrasts is in [18].



(a) Input image (b) BLF (σ_s=5, σ_r=0.15) (c) L_0 smoothing (λ=0.04, κ=1.05) (d) Median Filter ($r = 6$)

(e) Closest Mode (σ_K=0.1, σ_W=3) (f) Subr et al. [16] (k=3) (g) Xu et al. [17] (λ=0.03, σ=5) (h) Ours (σ=0.1, σ_s=5)

Fig. 8. Comparison of high-contrast detail smoothing. The parameter settings are corresponding to each operator's own formulation and tuned with our best efforts for smoothing out high-contrast details while preserving major structures (e.g., smooth the flower region and keep the edges of houses clear). Only Xu et al. [17] and our tree filter can successfully smooth out high-contrast trivial details (see the right close-up windows). Note the subtle difference between the two operators: Xu et al. [17] can produce more flattened results, while our tree filter can generate results with more accurate edges around corners.

However, their pipeline involves too many steps and is brittle in practice (especially the manipulation of a low-pass filtering followed by an edge sharpening). Fig. 9 shows a comparison of our tree filter to their method. Also note that the edge sharpening step in their pipeline is based on L_0 gradient optimization, which is rather computationally intensive.

(a) JIAS [18] (b) JLLM [18] (c) Ours (σ=0.1, σ_s=8)

Fig. 9. Comparison to the two methods proposed by Su et al. [18]: joint iterative asymmetric sampling (JIAS) and joint local linear model (JLLM). Their methods rely on a low-pass filtering followed by an edge sharpening, which is brittle in practice and may easily fail on smoothing irregular details.

B. Limitation and Improvement

1) Tree-Median Filtering: As we analyzed in previous sections, tree filter uses the idea of collaborative filtering to alleviate the "leak" problem of the tree distance. However, in extreme cases, the simple strategy of collaborative weighted-average may not be able to fix the "leak" problem (see the top-right corner of Fig. 11(e), the white region is contaminated by the "leak"). Complex strategies could be employed to solve this problem, which may inspire future novel filter. But we here propose a simple solution from another perspective: to modify the tree-mean filtering step in the tree filter's implementation.

As described in Sec. IV-A, the tree-mean filtering is to calculate weighted average using tree distance. The weights assigned to other pixels completely ignores their color/intensity differences to center pixel, and thus the "leak" problem of tree distance is introduced in this way. Let's consider a more clever way for choosing an output value for center pixel: if we use tree distance to collect some nearby neighbors, and then use the histogram of these neighbors for determining the output, the "leak" problem may not be introduced. For example, use the median among these neighbors as center pixel's output – that is, replacing the tree-mean filtering with a *tree-median filtering* in the tree filter's implementation (the second step remains unchanged). Fig. 11(f) shows a result obtained by the improved tree filter. The "leak" problem get perfectly solved. Note that the overall color appearance is more like the input image than the original one. This is because it does not mix colors together like the weighted-average in tree-mean

(a) Input (b) BLF (σ_s=8, σ_r=0.2) (c) Xu et al. [17] (λ=0.015, σ=4) (d) Ours (σ=0.2, σ_s=5)

(e) Input (f) BLF (σ_s=5, σ_r=0.1) (g) Xu et al. [17] (λ=0.015, σ=3) (h) Ours (σ=0.2, σ_s=5)

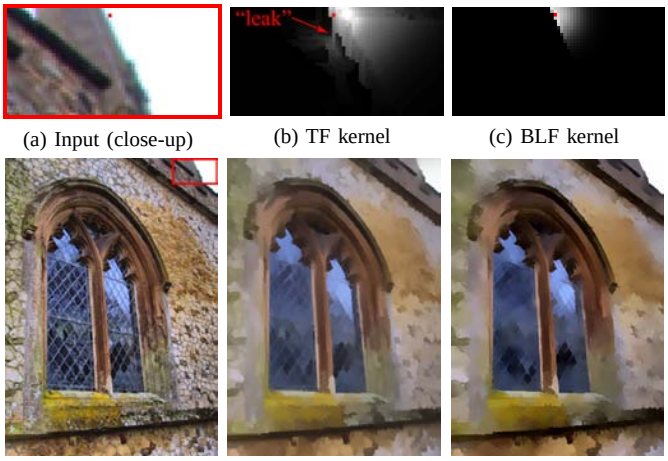
Fig. 10. Image smoothing examples. Both our tree filter and Xu et al. [17] are designed to smooth high-contrast details. Note the subtle difference between the two operators: Xu et al. [17] can produce more flattened results, while our tree filter can generate results with more accurate edges around corners.

filtering.

One problem of the tree-median filtering is that it currently does not have a fast algorithm, hence the improved tree filter will be much slower than the original tree filter. Another problem is that, if stronger smoothing is desired, increasing the parameter of tree-median filtering (e.g., the radius of collecting neighboring pixels on the tree) may not help.

2) Iterative Tree Filtering: We mentioned a failure case of the tree filtering for smoothing details in Sec. III-C. When undesired details are near a similar large homogeneous region, they cannot be removed by the tree filter because of the collaborative support from the large region (see left close-up window of Fig. 10(d)). This is particularly serious for strongly textured images such as mosaics. Fig. 12(c) shows an example of such failure case: residual textures are obvious in the filtered result, especially near large homogeneous regions (see the right close-up window).

Fortunately, we notice that, although the textures cannot be completely removed, they actually get strongly attenuated after the tree filtering. Thus if we perform another one or more iterations of tree filtering on the result, the residual textures can eventually be completely removed. Fig. 12(d) shows the result of 5 iterations of tree filtering (note the parameter σ is set to a smaller value to avoid overly smoothing). The overall look of the result is comparable to the one produced by state-of-the-art optimization-based method for texture-structure separation [17], while a closer examination shows that our method is better at preserving image structures which may be mistakenly identified as textures by the RTV (see left close-up window).



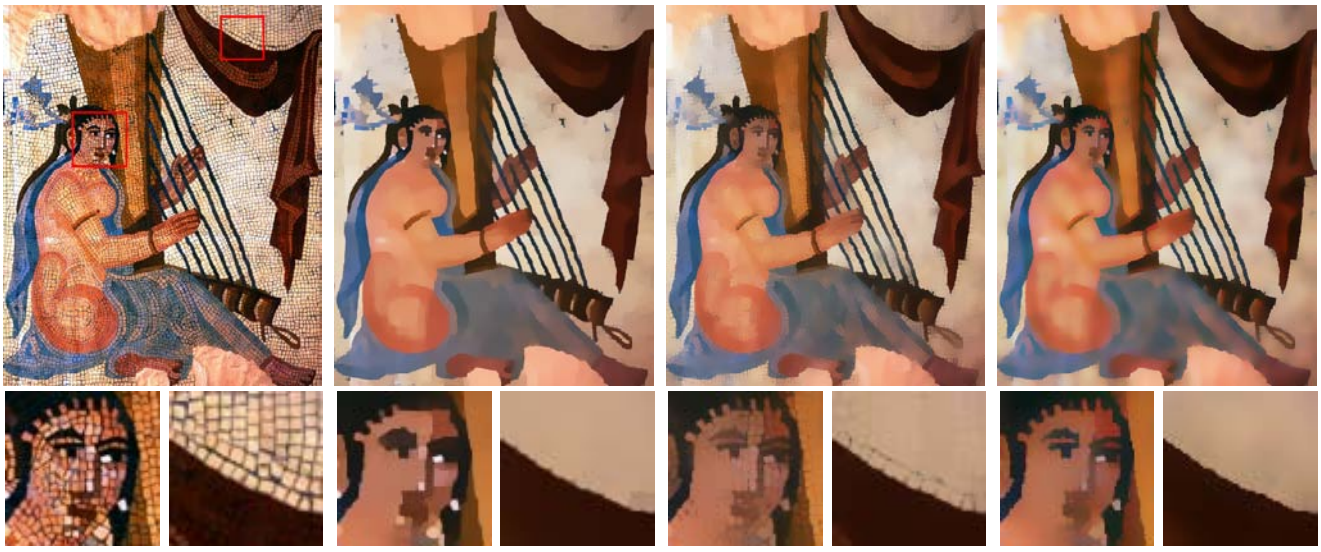
(a) Input (close-up) (b) TF kernel (c) BLF kernel

(d) Input (e) Original Tree Filter (f) Improved Tree Filter

Fig. 11. Extreme case of "leak" problem. The top-right corner region has a "bridge" connected to the other region by the MST. Thus a pixel located near the "bridge" will be contaminated by dissimilar pixels from the other region. The kernel of the tree filter for such a pixel is shown in (b), and the kernel of bilateral filter is shown in (c) for reference. Replacing tree-mean filtering with tree-median filtering improves the result, see (f). Note the difference at the top-right corner.

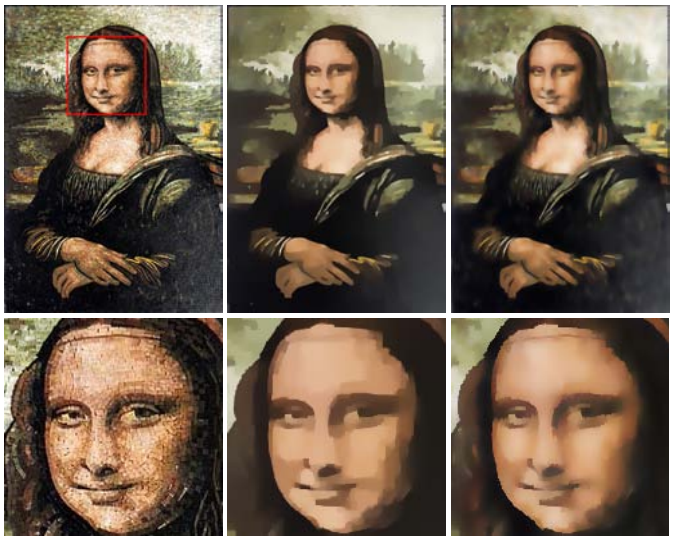
Fig. 13 shows another comparison of the smoothing on highly textured image.

3) Multi-Tree Filtering: Besides the proposed collaborative filtering scheme, the "false edges" and "leak" problem can also be treated in another way. Since the positions of the "false edges" and "leak" are quite arbitrary due to the MST construction, considering other spanning trees where "false edges" and



(a) Input (b) Xu et al. [17] (λ=0.015, σ=3) (c) Tree Filter (σ=0.1, σ_s=3) (d) Iterative TF (σ=0.02, σ_s=3)

Fig. 12. Iterative tree filtering for texture smoothing. The single iteration tree filtering will leave some residual textures, while the iterative tree filtering (5 iterations) can completely smooth out the textures. Compared to the optimization-based method [17], our method can better preserve image structures which may be mistakenly identified as textures by RTV (see the eyebrow in the left close-up window).



(a) Input (b) Xu et al. [17] (c) Iterative TF

Fig. 13. Iterative tree filtering for texture smoothing. Parameters of our iterative tree filtering: σ=0.01, σ_s=3, 4 iterations. The result of Xu et al. [17] is overly flattened with staircase effects, while our result seems more natural for reflecting the gradual transition in original image (see her cheek). We suggest readers to take a close look at the results in a high resolution display.

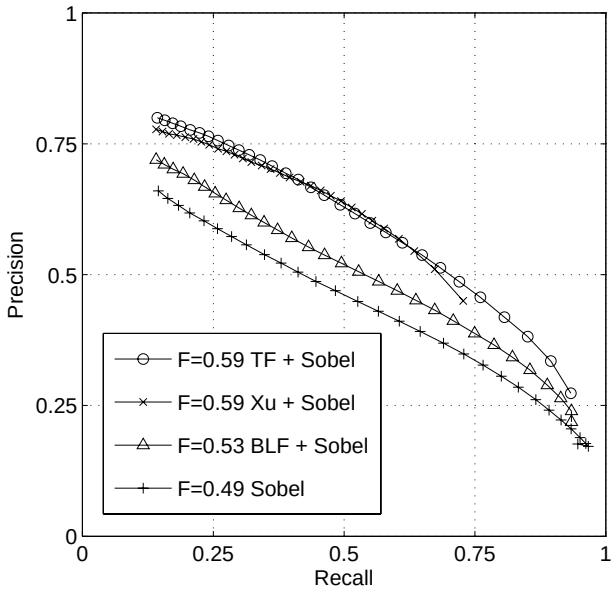


Fig. 14. Edge/boundary detection evaluation on BSDS300 [40]. The evaluation is performed on 100 test images using *grayscale* input (filtering is also performed on grayscale image). Parameters for bilateral filtering and tree filtering are $\sigma_s = 3$, $\sigma_r = 0.03$, $\sigma = 0.1$. For Xu et al. [17], parameters are λ=0.015 and σ=3. The score shown in the figure is produced using the benchmark code [40] (the higher, the better).

"leak" are located at different sites may help eliminating the problems. A natural idea is that, instead of constructing only one minimum spanning tree, we can construct several spanning trees and then use the largest tree distance (between two pixels) among all the trees to calculate pixel affinity. However, constructing several spanning trees which have different "false edges" and "leak" positions between each other, as well as efficiently calculating tree distances using several trees, is non-trivial and will be left as future work.

VI. Applications

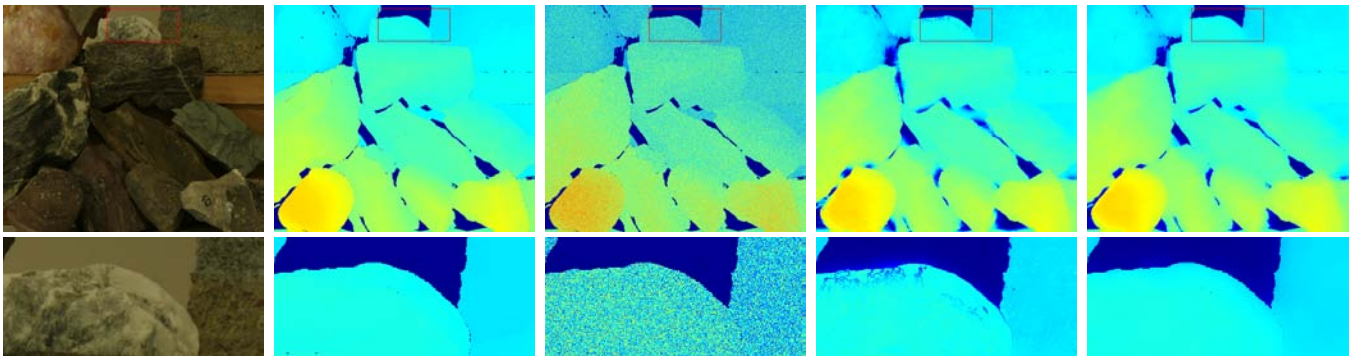
The smoothing of high-contrast details has been shown useful in many applications [17], [18]. We in this section briefly review several applications where tree filter can find its place.

A. Scene Simplification

The efficiency, as well as the ability to smooth out high-contrast details, makes tree filter an ideal tool for serving as a pre-processing tool for applications where trivial details are undesirable, e.g., edge/boundary detection, image abstraction, shape matching, scene understanding. As a first example, we demonstrate the benefits of tree filtering as a pre-processing step for edge/boundary detection. For simplicity

(a) Input (b) Sobel (0.64) (c) BLF + Sobel (0.69) (d) TF + Sobel (0.77) (e) Human

Fig. 15. An example of the edge/boundary detection results on BSDS300 (with score in the caption), see Fig.14 for details. Tree filtering effectively reduces trivial details which are not labeled as edges/boundaries by human subjects.



(a) Clean RGB image (b) Ground truth (c) With noise (19.1dB) (d) Joint BLF (28.3dB) (e) Joint TF (33.3dB)

Fig. 16. Disparity map denoising using joint filtering. From left to right: clean RGB images, ground-truth disparity map, disparity map deteriorated with Gaussian noise, denoised disparity map using joint BLF ($\sigma_s = 8$, $\sigma_r = 0.01$), denoised disparity map using our joint tree filtering ($\sigma = 0.02$, $\sigma_s = 8$, $\sigma_r = 0.01$). Note that in the close-up window, the joint BLF introduces textures in the RGB color image into the filtered result, while this does not happen in joint tree filtering thanks to the strong smoothing ability of tree filtering. The captions under subfigures show the PSNR values.

and practicality, we use a lightweight edge detector, namely Sobel detector, to perform the experimental evaluation (note that other complicated operators can also be employed here for evaluation, but we prefer such fast, simple yet effective solution since it can be easily embedded in more complex applications). Quantitative evaluation is conducted on a well-known boundary detection benchmark, namely Berkeley Segmentation Dataset (BSDS300) [40], which contains 100 test images with human labeled "ground truth" boundaries. Fig. 14 shows the improvements of employing a pre-processing step, either the bilateral filtering, tree filtering, or Xu et al. [17], before performing the Sobel detector. Fig. 15 shows one result among that of all 100 test images. It is clear in the results that tree filtering or Xu et al. [17] can effectively reduce trivial details in the scene and thus produce simplified scene for better edge/boundary detection (note that tree filtering is substantially faster than Xu et al. [17]). Similarly, a quick example of image abstraction can be assembled by adding the edges back to the filtered image (see Fig. 17).

(a) Input image (b) Cartoon-like abstraction

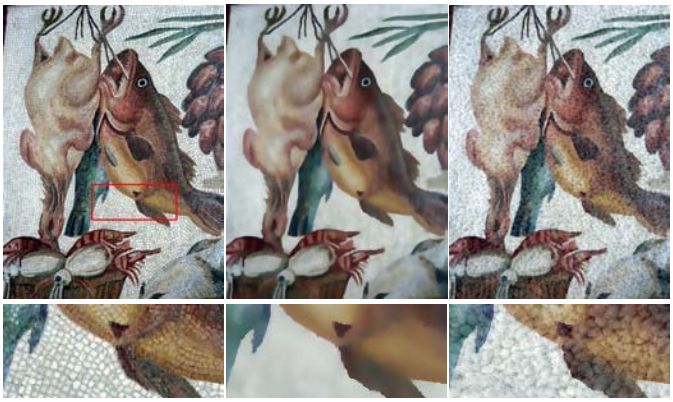
Fig. 17. Abstraction example. Note that the high-contrast textured regions cannot be flattened by bilateral-filter-like operators. Parameters of the tree filtering are $\sigma = 0.2$, $\sigma_s = 5$.

B. Joint Filtering

Instead of using the original input image to build the MST, using another guidance image to build the MST can make tree filtering more flexible and powerful. For example, in depth sensing applications where both depth image and RGB image are available (such as commercial active or passive depth sensing cameras), the obtained depth images are usually noisy and can be joint filtered using the corresponding clean RGB images as guidance [41]. To demonstrate such application, we use a dataset with ground-truth disparity map[5] obtained from structured light [42] and manually add Gaussian noise[6] to the ground-truth disparity map for denoising experiment. Fig. 16 shows an example of disparity map denoising using joint filtering. As demonstrated in the experimental result, tree filter can automatically "pick up" the major structures in guidance image to perform the joint filtering, while at the same time avoiding introducing trivial details of the guidance image into the filtered result.

[5]*Disparity* is a notion commonly used in stereo vision literature, which is inversely proportional to *depth*.

[6]Note that more realistic noise model should be established depending on specific type of depth sensor (different depth sensors have different types of noises, e.g., see [43] for a detailed discussion of denoising for *time-of-flight* depth data), which is out of the scope of this paper. We here use the simplest Gaussian noise model to demonstrate tree filter's ability to ignore details from guidance image while performing joint filtering.



(a) Input (b) Filtered (c) Texture replaced

Fig. 18. Texture replacement. We use iterative tree filtering ($\sigma = 0.01$, $\sigma_s = 2$, 3 iterations) to separate the input image into texture layer and structure layer. Replacing the wall brick texture with a textile texture yields a plausible result. We suggest readers to take a close look at the results in a high resolution display.

C. Texture Editing

Using the iterative tree filtering, we are able to separate highly textured image into texture layer and structure layer. The separation makes texture editing for such kind of image easier. For example, simply replacing the texture layer with another kind of texture can yield plausible result. Fig. 18 shows an example.

VII. Conclusion

We have presented the tree filter for strong edge-preserving smoothing of images in presence of high-contrast details. The tree filter utilizes a MST extracted from image, as well as the idea of collaborative filtering, to perform weighted average among pixels. Unlike previous image filtering operators, tree filter does not have a 1D version for 1D signals, because the MST explores the 2D structural nature of an image, e.g., some regions are connected if we view the image as a 2D planar graph but may not be connected if we only consider pixels row by row (as 1D signal) or window by window, which is one of the desirable features distinguishing tree filter from other operators. Thanks to the special properties of MST and the collaborative filtering mechanism, tree filter is able to smooth out high-contrast, fine-scale details while preserving major image structures. The fast implementation further makes tree filter a practical filtering tool that can serve for many applications. We believe the tree filter will shed lights on designing novel edge-aware image filters exploring the intrinsic 2D structure of images and the collaborative filtering mechanism.

References

[1] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *ICCV 1998*. IEEE, 1998, pp. 839–846.

[2] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 7, pp. 629–639, 1990.

[3] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski, "Edge-preserving decompositions for multi-scale tone and detail manipulation," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 27, no. 3, pp. 67:1–67:10, 2008.

[4] R. Fattal, "Edge-avoiding wavelets and their applications," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 28, no. 3, pp. 22:1–22:10, 2009.

[5] K. He, J. Sun, and X. Tang, "Guided image filtering," in *ECCV*, 2010, pp. 1–14.

[6] A. Criminisi, T. Sharp, C. Rother, and P. P'erez, "Geodesic image and video editing," *ACM Trans. Graph.*, vol. 29, no. 134, pp. 1–134, 2010.

[7] A. Criminisi, T. Sharp, and P. P'erez, "Geodesic forests for image editing," *MSR technical report (MSR-TR-2011-96)*, 2011.

[8] E. Gastal and M. Oliveira, "Domain transform for edge-aware image and video processing," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 30, no. 4, pp. 69:1–69:12, 2011.

[9] S. Paris, S. Hasinoff, and J. Kautz, "Local laplacian filters: Edge-aware image processing with a laplacian pyramid," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 30, no. 4, pp. 68:1–68:12, 2011.

[10] L. Xu, C. Lu, Y. Xu, and J. Jia, "Image smoothing via l0 gradient minimization," *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, vol. 30, no. 6, pp. 174:1–174:12, 2011.

[11] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 21, no. 3, pp. 257–266, 2002.

[12] Z.-F. Xie, R. W. H. Lau, Y. Gui, M.-G. Chen, and L.-Z. Ma, "A gradient-domain-based edge-preserving sharpen filter," *The Visual Computer*, vol. 28, no. 12, pp. 1195–1207, 2012.

[13] J. Van de Weijer and R. Van den Boomgaard, "Local mode filtering," in *CVPR 2001*, vol. 2. IEEE, 2001, pp. II–428.

[14] M. Felsberg, P. Forssén, and H. Scharr, "Channel smoothing: Efficient robust smoothing of low-level signal features," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 2, pp. 209–222, 2006.

[15] M. Kass and J. Solomon, "Smoothed local histogram filters," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 29, no. 4, pp. 100:1–100:10, 2010.

[16] K. Subr, C. Soler, and F. Durand, "Edge-preserving multiscale image decomposition based on local extrema," *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, vol. 28, no. 5, pp. 147:1–147:9, 2009.

[17] L. Xu, Q. Yan, Y. Xia, and J. Jia, "Structure extraction from texture via relative total variation," *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, vol. 31, no. 6, p. 139, 2012.

[18] Z. Su, X. Luo, Z. Deng, Y. Liang, and Z. Ji, "Edge-preserving texture suppression filter based on joint filtering schemes," *IEEE Trans. Multimedia*, vol. PP, no. 99, p. 1, 2012.

[19] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, and K. Toyama, "Digital photography with flash and no-flash image pairs," in *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 23, no. 3. ACM, 2004, pp. 664–672.

[20] S. Paris, P. Kornprobst, and J. Tumblin, *Bilateral filtering: Theory and applications*. Now Publishers Inc, 2009.

[21] J. Chen, S. Paris, and F. Durand, "Real-time edge-aware image processing with the bilateral grid," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 26, no. 3, pp. 103:1–103:10, 2007.

[22] F. Porikli, "Constant time o (1) bilateral filtering," in *CVPR 2008*. IEEE, 2008, pp. 1–8.

[23] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," *Intl. J. Computer Vision*, vol. 81, no. 1, pp. 24–52, 2009.

[24] Q. Yang, K. Tan, and N. Ahuja, "Real-time o (1) bilateral filtering," in *CVPR 2009*. IEEE, 2009, pp. 557–564.

[25] A. Adams, N. Gelfand, J. Dolson, and M. Levoy, "Gaussian kd-trees for fast high-dimensional filtering," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 28, pp. 21:1–21:12, Jul. 2009.

[26] A. Adams, J. Baek, and A. Davis, "Fast high-dimensional filtering using the permutohedral lattice," *Comput. Graph. Forum*, vol. 29, no. 2, pp. 753–762, 2010.

[27] Q. Yang, "Recursive bilateral filtering," in *ECCV 2012*, 2012, to appear.

[28] E. Gastal and M. Oliveira, "Adaptive manifolds for real-time high-dimensional filtering," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 31, no. 4, p. 33, 2012.

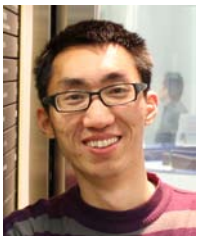
[29] M. Aubry, S. Paris, S. Hasinoff, and F. Durand, "Fast and robust pyramid-based image processing," *MIT technical report (MIT-CSAIL-TR-2011-049)*, 2011.

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[31] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *Intl. J. Computer Vision*, vol. 59, no. 2, pp. 167–181, 2004.

[32] Y. Haxhimusa and W. Kropatsch, "Segmentation graph hierarchies," *Structural, Syntactic, and Statistical Pattern Recognition*, pp. 343–351, 2004.

[33] J. Stawiaski and F. Meyer, "Minimum spanning tree adaptive image filtering," in *ICIP 2009*. IEEE, 2009, pp. 2245–2248.
[34] T. Koga and N. Suetake, "Structural-context-preserving image abstraction by using space-filling curve based on minimum spanning tree," in *ICIP 2011*. IEEE, 2011, pp. 1465–1468.
[35] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *WWW 2001*. ACM, 2001, pp. 285–295.
[36] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *IEEE Trans. Image Process.*, vol. 16, no. 8, pp. 2080–2095, 2007.
[37] Q. Yang, "A non-local cost aggregation method for stereo matching," in *CVPR 2012*. IEEE, 2012.
[38] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technology Journal*, vol. 36, pp. 1389–1401, 1957.
[39] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, vol. 34, no. 3, pp. 596–615, 1987.
[40] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *ICCV 2001*, vol. 2, July 2001, pp. 416–423.
[41] M. Mueller, F. Zilly, and P. Kauff, "Adaptive cross-trilateral depth map filtering," in *3DTV-CON 2010*. IEEE, 2010, pp. 1–4.
[42] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in *CVPR 2003*, vol. 1. IEEE, 2003, pp. I–195.
[43] F. Lenzen, K. I. Kim, H. Schäfer, R. Nair, S. Meister, F. Becker, C. S. Garbe, and C. Theobalt, "Denoising strategies for time-of-flight data," in *Time-of-Flight Imaging: Algorithms, Sensors and Applications, Springer LNCS*, 2013.

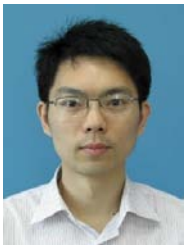


Linchao Bao is currently a Ph.D. student in the Department of Computer Science at City University of Hong Kong. He obtained a M.S. degree in Pattern Recognition and Intelligent Systems from Huazhong University of Science and Technology, Wuhan, China in 2011. His research interests reside in computer vision and graphics.

Yibing Song is currently a master student in the Department of Computer Science at City University of Hong Kong. He obtained a bachelor degree in Electrical Engineering and Information Science from University of Science and Technology of China in 2011. His research interests reside in computer vision and graphics.

Qingxiong Yang (M'11) received the B.E. degree in electronic engineering and information science from the University of Science and Technology of China, Hefei, China, in 2004, and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 2010. He is an Assistant Professor with the Computer Science Department, City University of Hong Kong, Hong Kong. His current research interests include reside in computer vision and computer graphics. He is a recipient of the Best Student Paper Award at MMSP in 2010 and the Best Demo Award at CVPR in 2007.

Hao Yuan received the PhD degree in 2010 from Purdue University, and the B.Eng. degree from Shanghai Jiao Tong University in 2006. He joined the Department of Computer Science at City University of Hong Kong as an assistant professor in 2010, and resigned in 2013. His research interests include algorithms, databases, information security, and programming languages.



Gang Wang (M'11) is an Assistant Professor with the School of Electrical & Electronic Engineering at Nanyang Technological University (NTU), and a research scientist at the Advanced Digital Science Center. He received his B.S. degree from Harbin Institute of Technology in Electrical Engineering in 2005 and the PhD degree in Electrical and Computer Engineering, University of Illinois at Urbana-Champaign in 2010. His research interests include computer vision and machine learning. Particularly, he is focusing on object recognition, scene analysis, large scale machine learning, and deep learning. He is a member of IEEE.