Segment Graph Based Image Filtering: Fast Structure-Preserving Smoothing

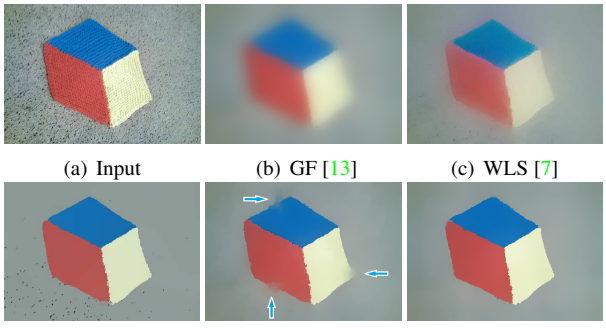
Feihu Zhang[1,2] Longquan Dai[2] Shiming Xiang[2] Xiaopeng Zhang[2]
[1]School of Computer Science, Northwestern Polytechnical University, Xi'an, China
[2]NLPR-LIAMA, Institute of Automation, Chinese Academy of Sciences, Beijing, China
hi.yexu@gmail.com, lqdai@foxmail.com, {smxiang, xpzhang}@nlpr.ia.ac.cn

Abstract

In this paper, we design a new edge-aware structure, named segment graph, to represent the image and we further develop a novel double weighted average image filter (SGF) based on the segment graph. In our SGF, we use the tree distance on the segment graph to define the internal weight function of the filtering kernel, which enables the filter to smooth out high-contrast details and textures while preserving major image structures very well. While for the external weight function, we introduce a user specified smoothing window to balance the smoothing effects from each node of the segment graph. Moreover, we also set a threshold to adjust the edge-preserving performance. These advantages make the SGF more flexible in various applications and overcome the "halo" and "leak" problems appearing in most of the state-of-the-art approaches. Finally and importantly, we develop a linear algorithm for the implementation of our SGF, which has an $O(N)$ time complexity for both gray-scale and high dimensional images, regardless of the kernel size and the intensity range. Typically, as one of the fastest edge-preserving filters, our CPU implementation achieves 0.15s per megapixel when performing filtering for 3-channel color images. The strength of the proposed filter is demonstrated by various applications, including stereo matching, optical flow, joint depth map upsampling, edge-preserving smoothing, edges detection, image denoising, abstraction and texture editing.

1. Introduction

Natural image often contains rich trivial details and textures, which may reduce the performance of many visual computing algorithms. Serving as the pre-processing or a key step for these algorithms, edge-preserving smoothing aims to remove trivial details while preserving major image structures. This makes it the foundation for many vision and graphics applications, including low-level image analysis (e.g., edge detection, image segmentation), image abstraction/vectorization, content-aware image editing *et al*.



(a) Input (b) GF [13] (c) WLS [7]
(d) L_0-smoothing [23] (e) TF [2] (f) Ours

Figure 1: Problems illustrations. (a) Image with high-contrast details and textures as input. (b) "halo" artifacts around major edges. (c) WLS [7] distributes the blurring ("halo") globally. (d) L_0-smoothing [7] could not smooth out some high-contrast details. (d) "leak" problem at some edges (see arrows). (e) Our method keeps the image structure very well.

Although many state-of-the-art edge-preserving smoothing techniques have been proposed recently, they may suffer from various problems, including "halo" artifacts, residual artifacts, "leak" problem and the time-consuming shortcoming. "Halo" artifacts (as shown in Fig.1(b)) usually appear around the major edges and are shared by a variety of smoothing techniques, including the widely used bilateral filter (BF), guided filter (GF) *et al*. These methods do not distinguish trivial details from major image structures, and they use intensity/color as the criterion to smooth out all contents in the images. Many improved researches (such as WLS [7], CLMF [16], L_0-smoothing [23] *et al*.) attempted to solve the "halo" problem and generate sharper edges. Although they are different from each other, the common behavior of these operators is to smooth out low-contrast details from input images. However, they typically only use pixel color/intensity contrasts (or image gradients) to distinguish details from major image structures, and as a result, they will cause new troubles when utilized to remove high-contrast details and textures. For example, L_0-smoothing [23] could not remove all of the trivial details (Fig.1(d)). While for WLS [7], the globally distributed blurring ("halo") becomes clear again (Fig.1(c)).

Unlike the "halo" artifacts, "leak" problem is introduced by tree filtering (TF) scheme [2, 27] and only occurs at some of the major object edges. This is because that TF [2, 27] constructs a minimum spanning tree (MST) to connect all pixels in the image together and aggregates the smoothing effects globally along the MST. As a result, connections will inevitably cross some strong edges, and these edges will be corrupted (as illustrated in Fig 1(e)).

Although some approaches, like texture smoothing [24], do not suffer from above problems, they have extremely high computational cost. Therefore, they cannot be applied to applications with high speed requirement. It is thus desired to design a new image filter which should be able to

1. smooth out high-contrast details/textures effectively;
2. keep the image structure (major edges) pretty well;
3. run in linear time complexity (be suitable for real-time processing).

In this paper, we design a new edge-aware structure (segment graph) to represent the image, and we further develop a double weighted average image filter named segment graph filter (SGF), in which the three goals could be simultaneously achieved. Moreover, other superiorities are possessed, including local nature, ease of implementation, scalability, flexibility and adaptability to a variety of application scenarios.

2. Related Work and Problem Analysis

The widely used structure-preserving smoothing techniques can be categorized into two types. One contains the optimization based filters, among which Farbman *et al*. [7] propose an edge-preserving filtering method based on weighted least square (WLS) optimization. Xu *et al*. [23] develop an edge-preserving smoothing algorithm according to the optimization on the L_0 norm of image gradients to produce piecewise constant images. Generally speaking, these two methods could generate impressive performance when used to smooth out low-contrast details since they are designed based on image gradients. However, these intensity contrast or gradient-oriented approaches could not suppress high-contrast details and textures. Xu *et al*. [24], on the other hand, design a local variation measure, namely, Relative Total Variation (RTV), to distinguish textures from major image structures regardless of contrasts. This method produce favorable results for highly textured images, but it may overly smooth natural images. Also as one of the common limitations, the performance of these methods comes at the price of huge time consumption.

The weighted average based smoothing approaches, including our method, are another kind of edge-preserving smoothing techniques. Typically, bilateral filter (BF), as the first proposed edge-preserving filter, has been widely used due to its simplicity. However, computational efficiency (non-linear) is still one of major challenges which limits its performance in many applications, e.g. stereo matching, optical flow. Even though, many acceleration algorithms have been developed for BF [4, 12, 26, 28], they are usually the approximation approaches [4, 12, 26] or their performance depends on the intensity range of the image [28]. The linear GF is introduced in [13] as a linear transform of the guidance image. Being computationally much faster than BF, GF has demonstrated its unique advantages over BF in some applications such as detail enhancement, stereo and optical flow [19] *et al*. However, both BF and GF cannot get rid of the "halo" artifacts which would concentrate the blurring near the object edges (Fig.1(b) and 2(b)). Subsequently, the cross-based local multipoint filter (CLMF) [16] is proposed to remove "halos" and generate sharp edges in linear time. It takes a spatial-aware window to select the most relevant pixels for smoothing. However, its performance degenerates greatly when used for high-contrast details smoothing.

Being different from above point-wise intensity based filters, Karacan *et al*. develop a patch-based texture removal algorithm by using the similarity measures built on a region covariance descriptor [14]. Cho *et al*. jointly employ the idea of Karacan and the weighted averaging scheme of BF to present a texture filter (BTF) [6] for texture removing. These two local filters have shown superiorities in texture/structure separation. However, just like the optimization based techniques, the impressive performance comes at the sacrifice of the time complexity, and they may also suffer from other problems (for example, over smoothing of some image edges). In addition, Zhang *et al*. propose the iterative rolling guidance filter (RGF) scheme [29] with controlling of the level of details during filtering. While, it tends to round off the object edges or corners (Fig.9(i)). Recently, geodesic filtering algorithms are well studied for image denoising [5, 11]. However, they are not strong enough for high-contrast details smoothing.

On the other hand, by treating an image I as a standard four connected, undirected grid (planar graph) with nodes being the image pixels and edges being weighted by color/intensity differences (Eq.4), a minimum spanning tree (MST) can be computed by removing edges with large weights leaving the remaining edges connecting through all pixels as a tree (Kruskal algorithm or other linear algorithm [9, 15]). Then, the similarity between any two vertices could be decided by their shortest distance on the MST. The MST extracted from the image has two important properties which make the tree distance be an edge-aware metric for high-contrast details smoothing. 1) MST can automatically drag away two dissimilar pixels that are close to each other in the spatial domain. 2) Small isolated region surrounded by large homogeneous region with dissimilar color/intensity (noise, high-contrast details and textures) will

(a) visualized input (b) GF [13] (c) MST [27] (d) TF [2] (e) SGF denoising (f) SGF smoothing

Figure 2: (a) Color visualized noisy input (PSNR=23.9dB), (b) "halo" appears around major edges (PSNR=32.1dB). (c) No "halos", but "leaks" and "false edges" destroy the image structure (PSNR=28.4dB), (d) TF [2] just alleviates the "leak" problem, but it still appears around some object edges (35.2dB). (e) With a small smoothing window, our method could be used for denoising (**45.5dB**), and all the object edges are preserved well. (f) Some undesirable details ("stars") removed by using large window while other contents are preserved.

be connected to the surrounding region with a short tree distance during the MST construction.

Although the tree distance has the edge-aware property and can be used to remove the "halo" artifacts, there is an obvious problem which might limits its development in edge-preserving smoothing. That is the "leak" problem (Fig.2(c)). Since the global 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. As a result, when used for smoothing images, these major edges are easily destroyed. To alleviate the "leak" problem, Bao *et al*. combine the joint BF and the original MST filtering scheme [27] to develop their own edge-preserving TF [2]. Besides the tree distance, it also involves the pixel spatial distance and color/intensity difference as the smoothing criterions. However, the improvement comes at the sacrifice of the time complexity. Moreover, the "leak" problem is just alleviated but not fundamentally solved (Fig.2(d)).

In fact, the "leaks" just appear in the global MST scheme. To take full advantage of the edge-aware property of the tree distance and remove the "leaks" fundamentally, instead of using MST, it is more reliable to design a new edge-aware structure to represent the image and further to develop the image filtering approach for structure-preserving smoothing.

3. Segment Graph Based Image Filter

In order to solve the "halo" problem which appears in most of the local filters, tree distance can be introduced. Instead of using the global MST scheme which suffers from the "leak" problem, we design a more reliable edge-aware structure to represent the image, namely the segment graph. Based on this, we develop our segment graph based image filter (SGF), a linear local filter that can smooth out high-contrast details while preserving major image structures.

3.1. Segment Graph Construction

As a key building block to many vision algorithms, recently, superpixel decomposition of a given image has been actively studied. For example, the recently proposed SLIC superpixel algorithm [1] can decompose an input image I into non-overlapping superpixels and yield adherence to the major image boundaries. The size of the superpixels can be easily adjusted and more importantly, it can run very fast (linearly).

We construct a well-designed segment graph based on the superpixel segmentation. The segment graph construction can be achieved by three step: 1) decompose the entire image domain I into superpixel regions, in which the pixels usually share the similar intensity; 2) for each superpixel region S, we build a local MST; 3) treat each sperpixel S as node of the segment graph and choose the minimal edge between S and each of its K neighborhoods S_i $(0 \leq i < K)$, namely, $E_{min}(S, S_i) = min\{W(u, v)|u \in S, v \in S_i\}$ to connect the adjacent nodes/superpixels.

Part of the segment graph structure is illustrated in Fig.3(a). We will analyze the effects of the segment graph on the filtering results in section 4. Next, we begin to build our structure-preserving image filter.

3.2. Filtering Kernel

Unlike most of the weighted average filters, we novelly design our SGF as a double weighted average filter. Given a scalar-valued input image I , the filter computes an output image J by

$$J_p = \frac{1}{K_p} \sum_{0 \leq i < k} \omega_2(p, S_i) \sum_{q \in S_i} \omega_1(p, q) I_q. \quad (1)$$

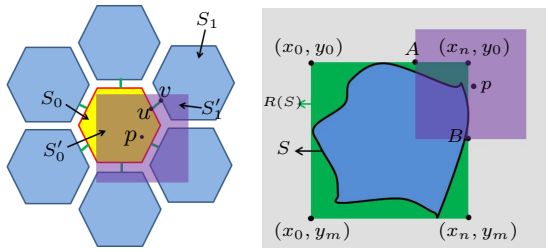
Here, K_p is a normalizing term. S_i represents superpixel region around the pixel p. The output J_p at pixel p is a double weighted average of the intensity value I_q in a specific neighbor region $\Omega = \cup_{0 \leq i < k} S_i$ $(q \in S_0)$. The two weight functions ω_1 and ω_2 are defined as totally different forms.

Internal Weight: For the internal weight function ω_1, we take the tree distance for major consideration and define ω_1 as

$$\omega_1(p, q) = \exp(-\frac{D(p, q)}{\sigma}). \quad (2)$$

In such a Gaussian function, σ controls the attenuation speed of $D(p, q)$ as it increases. ω_1 is inversely proportional to the tree distance $D(p, q)$ between two pixels p and q. $D(p, q)$ can be achieved by

$$D(p, q) = \sum_{0 \leq i < n} W(p_i, p_{i+1}). \quad (3)$$



(a) (b)

Figure 3: (a) Illustration of filtering kernel of our SGF. The superpixels are presented as hexagons. Green edges represent the connecting edges E_{min}. The pixel p is in the superpixel S_0 (yellow hexagon). And its filtering window Ω_p is shown with violet mask which overlaps with four superpixel regions $\{S_0, S_1, S_2, S_3\}$. As a result, the overlapped regions are $\{S'_0, S'_1, S'_2, S'_3\}$ and $\Omega_p = \bigcup_{0\le i<4} S'_i$. (b) Illustration of weighted average calculation (linear algorithm to calculate $|S'(p)|$ for any pixel p).

where $\{p_0, p_1 ... p_i, p_{i+1}, ... p_n\}$ constitute the path from pixel p to q on the tree (segment graph). $W(p_i, p_{i+1})$ can be achieved through Eq.(4), which is also the edge weight function when we construct the MST for each superpixel.

$$W(p, q) = |I_p - I_q|. \tag{4}$$

External Weight: In the filtering kernel Eq.(1), the external weight ω_2 balances the smoothing contribution of each superpixel region S_i. As illustration in Fig.3(a), just like GF and BF, we also introduce a smoothing window Ω_p (usually a user defined square window with radius r) for each pixel p. Such a window mask Ω_p will overlap with several superpixel regions. We denote these k overlapped superpixel as $\{S_0, S_1, \cdots S_i, \cdots S_{k-1}\}$ and represent the overlapped areas as $\{S'_0, S'_1, \cdots S'_i, \cdots S'_{k-1}\}$, namely, $S'_i = \Omega_p \cap S_i$. Finally, we define the weight function ω_2 as the area size ratio of S'_i and S_i.

$$\omega_2(p, S_i) = \frac{|S'_i|}{|S_i|}. \tag{5}$$

Here, $|S'_i|$ and $|S_i|$ denote the area size of S'_i and S_i.

3.3. Linear Implementation

Here, we describe the linear implementation of our SGF which includes three procedures: the internal aggregation, the external aggregation and the weighted average calculation. The overview of the implementation is presented in Algorithm 1. For simplicity, we set $C^A_{S_i}(p) = \sum_{q\in S_i} \omega_1(p, q) I_q$ and replace the filtering kernel (Eq.1) with

$$\begin{aligned} J_p &= \frac{1}{K_p} \sum_{0\le i<k} \omega_2(p, S_i) \sum_{q\in S_i} \omega_1(p, q) I_q \\ &= \frac{1}{K_p} \sum_{0\le i<k} \omega_2(p, S_i) C^A_{S_i}(p). \end{aligned} \tag{6}$$

Internal Aggregation: To calculate $C^A_{S_0}(p)$ $(p \in S_0)$, we take full use of the recently proposed linear cost aggregation algorithm [27] which contains two steps (upward aggregation and downward aggregation). For all the pixels in superpixel S_0 which have been connected with a local MST, the upward aggregation performs aggregation from leaf nodes to root node and stores the aggregation result $C^{A\uparrow}_{S_0}$ at each node. Let $p = P(q)$ denote the parent of node/pixel q. Then, for each pixel $p \in S_0$,

$$C^{A\uparrow}_{S_0}(p) = I_p + \sum_{P(q)=p} \omega_1(p, q) C^{A\uparrow}_{S_0}(q). \tag{7}$$

After the upward aggregation, downward aggregation is performed from root node to leaf nodes based on the previous upward aggregation result $C^{A\uparrow}_{S_0}$.

$$\begin{aligned} &C^A_{S_0}(p) = C^{A\uparrow}_{S_0}(p) \\ &+\omega_1(P(p), p) \cdot [C^{A\uparrow}_{S_0}(P(p)) - \omega_1(p, P(p)) \cdot C^{A\uparrow}_{S_0}(p)]. \end{aligned} \tag{8}$$

After the upward and downward aggregations, we can get the aggregated cost $C^A_{S_0}(p) = \sum_{q\in S_0} \omega_1(p, q) I_q$.

Neighbor Aggregation: After the internal aggregation, we perform our neighbor aggregation (step 9∼12) to calculate the rest $C^A_{S_i}(p)$ $(1 \le i < k)$. For $S_0 (p \in S_0)$ and its neighborhood $S_i (1 \le i < k)$, we can achieve $C^A_{S_i}(p)$ by Eq.(9).

$$\begin{aligned} C^A_{S_i}(p) &= \omega_1(p, v) \cdot C^A_{S_i}(v) \\ &= \omega_1(p, u) \cdot \omega_1(u, v) \cdot C^A_{S_i}(v). \end{aligned} \tag{9}$$

Where $u \in S_0, v \in S_i$ are the pixels/vertexes of the connecting edge $E_{min}(S_0, S_i) = min\{W(u, v) | u \in S_0, v \in S_i\}$ during the segment graph construction. $C^A_{S_i}(v)$ is the aggregated cost of pixel/vertex v in S_i after internal aggregation for superpixel S_i. Note that $\omega_1(p, u)$ for all pixel $p \in S_0$ can be achieved by traversing the MST in S_0 from root node u just once.

Weighted Average:[1] After the internal and neighbor aggregations (step 5∼12 in Algorithm 1), we still need a linear

Algorithm 1 Linear Implementation of the SGF

1: ♢ segment graph construction
2: over segment image I into superpixel lattices.
3: construct a local MST for each superpixel S.
4: connect S and all of its neighborhoods S_i by $E_{min}(S, S_i)$.
5: ♢ internal aggregation
6: **for** each superpixel region S **do**
7: aggregate the cost by Eq.(7) and Eq.(8).
8: **end for**
9: ♢ neighbor aggregation
10: **for** each superpixel S & its neighborhood $S_i (0 \le i < K)$ **do**
11: aggregate the cost $C^A_{S_i}$ to S (Eq.9).
12: **end for**
13: ♢ weighted average
14: **for** each pixel $p \in I$ **do**
15: calculate $\{|S'_0|, \cdots |S'_i|, \cdots |S'_{k-1}|\}$ for p.
16: **end for**
17: compute filtering result J_p for all pixels using Eq.(1) .

[1] See supplementary material for more details.

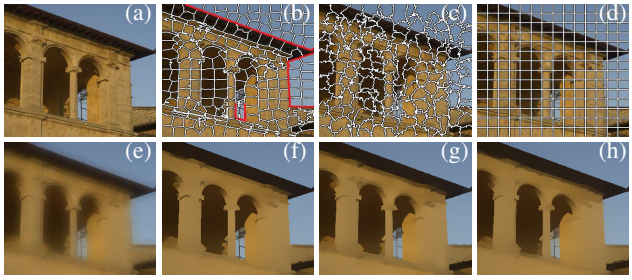


Figure 4: (a) Input, (b) SLIC superpixel segmentation, (c) random segmentation for comparison, (d) square lattices to construct the segment graph. (e) Guided filter [13] serves as the comparison ($r = 16, \varepsilon = 0.2^2$). (f)∼(h) are the smoothing results when using (b)∼(d) for segment graph construction. Parameters are all set as ($r = 16, \sigma = 0.2, \tau = 30/255$).

algorithm to calculate the external weight ω_2 for all pixels. Precisely, we need to calculate the overlapped area size $\{|S'_0|, \cdots |S'_i|, \cdots |S'_{k-1}|\}$ for all pixels $p \in I$. As illustrated in Fig.3(b), using superpixel region S as an example, we first find its minimum circumscribed rectangle $R(S)$ with its four vertexes as $(x_0, y_0), (x_n, y_0), (x_0, y_m)$ and (x_n, y_m). We treat $R(S)$ as a sub image I', and for any pixel $q \in I'$, we set $I'_q = 1$, if $q \in S$ (blue region in Fig.3(b)); otherwise, $I'_q = 0$ (green region). After that, we can get that for any rectangle region $\Omega_R \in R(S)$, the overlapped region size $|\Omega_R \cap S| = \sum_{q \in \Omega_R} I'_q$.

Then, for pixel p and its smoothing window Ω_p (with radius r), $|S'(p)| = |\Omega_p \cap S| = \sum_{q \in \Omega_p} I'_q$. However, Ω_p is not always included in $R(S)$. Fortunately, there is a corresponding rectangle $R^B_A \in R(S)$ which satisfies the conditions $R^B_A \cap S = \Omega_p \cap S$ and $|S'(p)| = \sum_{q \in R^B_A} I'_q$. It's clear that $R^B_A = R(S) \cap \Omega_p$. With A and B as the upper left and lower right vertexes of R^B_A, we can locate R^B_A by

$$\begin{aligned} A &= (x_A, y_A) = (\max(x_p - r, x_0), \max(y_p - r, y_0)); \\ B &= (x_B, y_B) = (\min(x_p + r, x_n), \min(y_p + r, y_m)). \end{aligned} \tag{10}$$

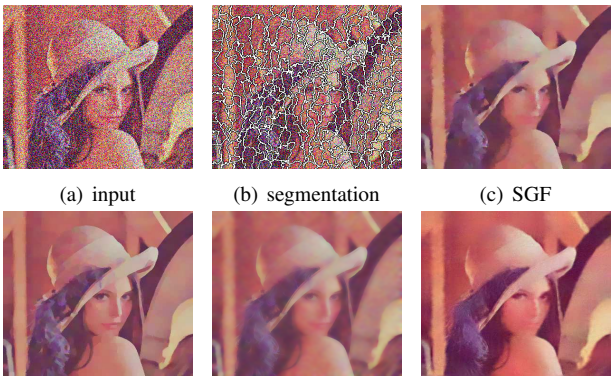
For example, in Fig.3(b), $|S'(p)| = \sum_{q \in R^B_A} I'_q$ and $R^B_A \in R(S)$ is the corresponding rectangle with its two diagonal vertexes: $A = (x_p - r, y_0)$ and $B = (x_n, y_p + r)$.

Finally, we employ the widely used linear boxfilter (or known as integral histograms) [17, 18] to calculate the summation of the pixel values in any size rectangle ($|S'(p)| = \sum_{q \in R^B_A} I'_q$). Then, we can get the filtering results by Eq.(1).

4. Algorithm Analysis and Extension

4.1. Effects of the Segment Graph

To take full advantages of our segment graph, during the neighbor aggregation course (step 9∼12 in Algorithm 1), we set a threshold τ to cut off some of the edges between S_0 and its neighborhood S_i. If the connecting edge $E_{min}(S_0, S_i) > \tau$, we stop the aggregation from S_i to S_0.



(a) input (b) segmentation (c) SGF

(d) patch-SGF (e) patch-geodesic [5] (f) NLM [3]

Figure 6: (a) Input with strong noises (standard deviation of the Gaussian noises σ_n = 100, PSNR=10.28dB), (b) unreliable SLIC superpixel results influenced by the noises, (c) result of the original SGF ($r = 12, \sigma = 0.1$, 2 iterations, PSNR=21.73dB), (d) improved SGF using patch strategy in [5] (22.93dB). (e)∼(f) state-of-the-art denoising methods (PSNR values: 22.37dB, 23.25dB)

Then, the filtering kernel (Eq.1) changes to Eq.(11).

$$\begin{aligned} J_p &= \frac{1}{K_p} \sum_{0 \le i < k} \delta_i \cdot \omega_2(p, S_i) \sum_{q \in S_i} \omega_1(p, q) I_q \\ \text{s.t.} \quad \delta_i &= \begin{cases} 0 & if\ E_{\min}(S_0, S_i) > \tau \\ 1 & otherwise \end{cases} \end{aligned} \tag{11}$$

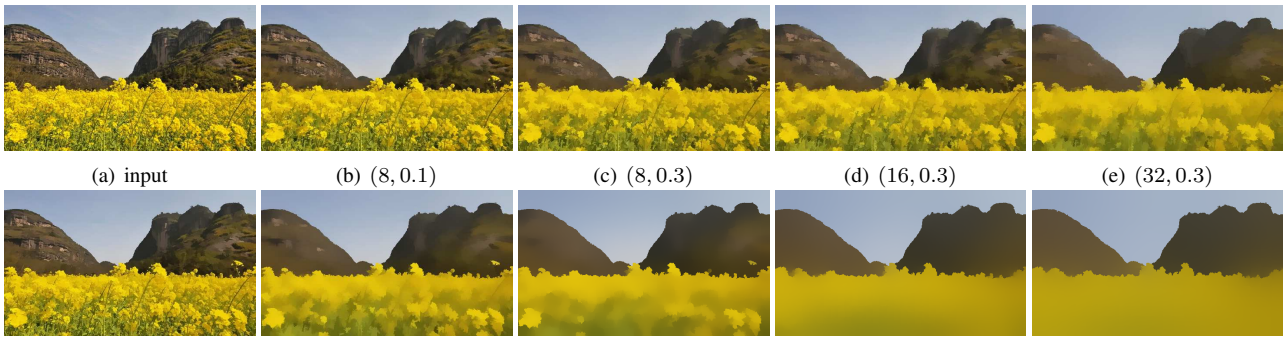
For example, in Fig.4 (b), by setting a reasonable threshold, the red edges are cut off to stop the aggregation. As an extension of the original SGF, the new filtering kernel (Eq.11) will contribute to better performance around edges and increase the flexibility in different applications.

4.2. Smoothing Effects Analysis

The double weighted design of our SGF possesses many superiorities which help to produce impressive edge-preserving smoothing performance. Here, we analyze the origins of these properties.

1) The SGF uses tree distance on the segment graph to measure the similarity between two pixels. Unlike other shortest path based geodesic approaches [5, 11], both local MST in the superpixel and the connection between superpixel nodes choose edges with the minimum weights (Eq.4), therefore, could avoid to cross the major object edges. So for any two adjacent pixels separated by major edges, the tree distance (Eq.3) will be large enough to avoid unwished smoothing effects. While for the trivial details or textures, the segment graph would connect them to the surrounding regions and bring them the smoothing effects.

2) The supported smoothing effect region for each pixel is adaptive. It's similar but not identical to many other approaches [16, 21] with variable support regions for filtering. Our SGF is implemented on the superpixel based segment graph. The shape-adaptive superpixels constitute a variable supported region for each pixel. Furthermore, with a proper threshold τ, such adaptive regions exclude pixels with



(a) input (b) $(8, 0.1)$ (c) $(8, 0.3)$ (d) $(16, 0.3)$ (e) $(32, 0.3)$

(f) $(16, 0.1)$ iteration 1 (g) iteration 3 (h) iteration 5 (i) iteration 15 (j) iteration 100

Figure 5: (a) natural iamge (resolution: 385×580) as input. (b)∼(e) smoothing results with different parameter settings. (f)∼(j) iterative filtering results. Note that the values of the subtitle gives the parameters (r, σ). τ is set as 30/255 for all the results.

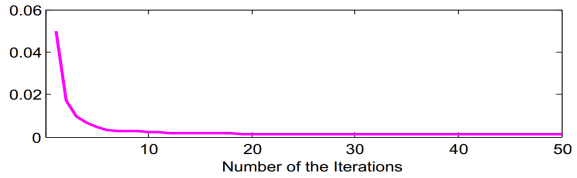
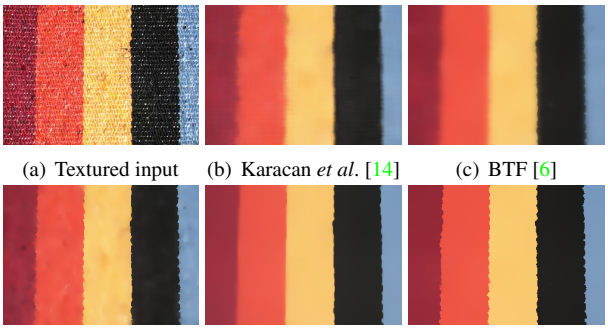


Figure 7: Per-pixel intensity difference between two iterations by using iterative SGF. Data are collected from test sample in Fig.5.



(a) Textured input (b) Karacan *et al.* [14] (c) BTF [6]

(d) TF [2] (e) Xu *et al.* [24] (f) Our SGF

Figure 8: Texture suppression. (b)∼(e) usually overly smooth some major edges in order to remove strong textures. Parameters are adjusted to prove their best performance. (f) result of our SGF ($r = 12, \sigma = 0.15, \tau = 30$, 3 iterations iterative filtering).

strong color differences and pick out the most relevant pixels to generate smoothing effects. As a result, it contributes to structure-preserving performance.

3) The external weight ω_2 properly balances and spreads the smoothing effects along the segment graph. It limits the effects of each superpixel on the target pixels, reduces the influence of the shape and size of the superpixels and prevents new false edges being generated at superpixel edges.

4) The local nature of our SGF further restricts the smoothing effects regions for each pixel to a rational local area and enables the SGF to avoid "leaks" and "false edges" appearing in some of the nonlocal approaches [2, 27].

Benefiting from the complementary effects of these properties, our SGF achieves impressive performance for structure-preserving smoothing. This also guarantees that in some extreme cases, for example, where the superpixel segment results used to build the segment graph are unreliable, the other three properties could still work together to generate satisfactory results. For example, in Fig.4, we use the superpixel segmentation results, a random segmentation results and the square grids to construct three kinds of segment graphs and then run the filtering. The results of them (Fig.4 (f)∼(b)) are slightly different and the major structures are all well preserved. This indicates that our SGF does not heavily rely on the superpixel segmentation. Furthermore, we test the performance of our SGF in extreme conditions where the superpixel segmentation is influenced by strong noises. As shown in Fig.6(b), the segmentation results degenerate greatly for image with strong noises. As a comparison, the output of the SGF are relatively satisfactory even though it could not compete with the state-of-the-art denoising algorithms [3, 5]. However, benefiting from the special design of our SGF, other strategies, for example, the effective patch strategy described in [5] could be introduced directly to improve the denoising performance without sacrificing the linear time complexity. Namely, we could replace Eq.(4) with the patch based distance to achieve the stronger patch-SGF for denoising. The result of patch-SGF is given in Fig.6(d) which is even better than the path-geodesic algorithm [5] with the same patch strategy for strong noise removal.

4.3. Parameters

Our SGF contains three parameters (r, σ, τ). r and σ can be adjusted as the BF [22]. While, for the threshold τ (Eq.11) which is used to cut off some of the smoothing aggregation on the segment graph, we recommend to choose the value among $(10/255, 60/255)$. We show the results with different parameters in the first row of Fig.5. As for SLIC superpixel algorithm,[2] we choose random values from the allowed range as parameters. Namely, we set the approximate size of the superpixels as random values in $((2r + 1)^2/3, (2r + 1)^2/2)$ with r as the radius of the

[2]SLIC parameters are further analyzed in supplementary material.

(a) Input (b) GF [13] (c) CLMF [16] (d) Domain Transform [10] (e) L_0-smoothing [23]

(f) TF [2] (g) WLS [7] (h) Xu *et al*. [24] (i) RGF [29] (j) Our SGF

Figure 9: Comparisons of high-contrast detail smoothing. The parameters of (b)∼(i) are adjusted with our best efforts. Results indicate that (b)∼(e) could not smooth out some of the high-contrast details, while (f)∼(h) will overly smooth (blur) some of the major edges. (i) RGF tends to round off the square corners. (Take a close look at the results in a high resolution display.) Only our SGF ($r = 20, \sigma = 0.1, \tau = 30/255$, 3 iterations) could successfully smooth out high-contrast trivial details while preserving image structures very well.

Table 1: 1-pixel error results on Middleybury stereo benchmark

Methods	Rank	Average Error (%)	Time (s)
Our SGF	**27**	**4.91**	4.9
CLMF-1 [16]	44	5.13	9.1
GF [19]	49	5.55	5.2
NonLocal [25]	55	5.48	**3.1**
BF [22]	96	6.67	109.5

Table 2: RMSE evaluations for $8\times$ upsampling results

Test Views	BF	GF	CLMF-1	[8]	**SGF**
Art	7.91	8.67	7.79	4.61	**4.23**
Laundry	4.01	5.39	4.57	3.12	**2.97**
Moebius	3.19	3.24	3.44	**1.96**	2.00
Teddy	3.57	3.07	2.34	**1.86**	1.91
Time (s)	5.1	0.20	0.44	5.9	0.26

Table 3: Time consumption comparisons of different filters

Local Filters	Time (s)	Global Filters	Time (s)
BF [22]	7.1	WLS [7]	7.9
GF [13]	0.10	L_0-smooth [23]	9.7
CLMF-1 [16]	0.31	Xu *et al*. [24]	17.3
Karacan *et al*. [14]	23.4	TF [2]	3.1
BTF [6]	21.6		
Our SGF	0.15		

smoothing window, and we also use random values among $(10, 30)$ to control the compactness of the superpixels.

Iterative Filtering: We extend our SGF for iterative filtering. For highly textured images, instead of using large r and σ, we find the iterative smoothing is more effective. Namely, we use the output as input to smooth the image more than once. For most of the cases, $2 \sim 4$ iterations are enough to remove the textures and high-contrast details. As illustrated in the second row of Fig.5, even after many iterations, the major edges of results are preserved very well. Moreover, the iterative SGF converges rapidly. We plot the difference between two successive filtering results in Fig.7. Unlike the other filters, the iterative SGF could converge to an image with only major structures left (Fig.5(j)).

Table 4: Average endpoint error evaluations for optical flow

Test Views	NonLocal	GF	CLMF-1	**Our SGF**
Venus	0.2347	0.2177	0.2374	**0.1571**
Grove2	0.2453	0.1937	0.2016	**0.1593**
Grove3	0.9324	0.7851	0.7910	**0.5442**
Urban2	0.3391	0.3901	0.3321	**0.3034**
Urban3	**0.2963**	0.3759	0.3598	0.3276
Hydrangea	0.2803	**0.2539**	0.2606	0.2654
Dimetrodon	0.2531	0.2106	0.2310	**0.1910**
RubberWhale	0.2956	0.2164	0.2073	**0.1687**
Average	0.3596	0.3304	0.3276	**0.2646**
Time (s)	**178**	326	579	279

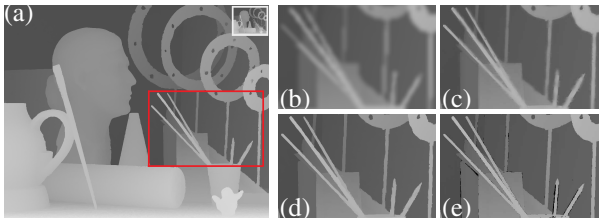


Figure 10: Depth map upsampling results. (a) low-resolution input (upper right) and $8\times$ upsampling result of our SGF. (b)∼(e) close-up of the red window. (b) GF [13], (c) state-of-the-art, David *et al*. [8], (d) our SGF (σ=0.05, r=16, τ =20/255), (e) ground truth.

5. Experiments

This section presents various applications of the proposed SGF. To shorten the paper, only 3 applications (stereo, optical and depth map upsampling) are presented. Others including texture editing, image abstraction, image/depth

map denosing, scene simplification/edges detection along with more test results appear in the supplementary material.

5.1. Edge-Preserving Smoothing

Our SGF is designed for structure-preserving smoothing and aims to smooth out high-contrast details and textures effectively. Using natural image with high-contrast details as input, we demonstrate the performance of our SGF in Fig.9 which indicates that our SGF could keep the major edges better than other 8 filters. We also prove the superiorities of our SGF when used for texture smoothing by comparing with other 4 state-of-the-art texture smoothing methods in Fig.8. Unlike other approaches, our method will not overly smooth edges during the texture removing. What's more, the flexibility of our SGF has also been proved. For example, we could use a small window for structure-preserving denoising (Fig.2(e)). While, by increasing the size of smoothing window, our SGF could remove undesirable trivial contents in the images (Fig.2(f)).

5.2. Applications

Stereo Matching and Optical Flow: The accuracy of some correspondence field estimation algorithms (stereo matching and optical flow) is dependent on the cost aggregation scheme. As prior methods employ BF [22] and GF [13] for cost aggregation, our SGF can also be applied to filter the cost volume while preserving edges. We embedded our SGF into the CostFilter framework [19] and used it for stereo matching and optical flow. We also compare our SGF with several local filters and Yang's non-local aggregation algorithm [25] using Middlebury stereo benchmark [20]. Results in Table 1 show that our method ranked 27 among more than 150 stereo approaches on the benchmark for 1-pixel threshold errors, which is far better than other filtering based methods. Also for optical flow, we compare their performance using 8 test views from Middlebury optical data set and adopt the average endpoint errors for evaluations. The results shown in Table 4 indicate that our SGF performs better in 6 of the 8 test views.

Joint Depth Map Upsampling: Given a low-resolution depth map and a registered high-resolution, color image, the resolution of the depth map can be enhanced by joint filtering with the color image as the guidance. Table 2 compares the performance of different filters when applied to upsample a low resolution depth map by a scaling factor of 8. Compared with BF, GF and CLMF, our SGF significantly improves the depth map accuracy, particularly for challenging depth discontinuities (Fig.10). Our SGF achieves the results close to (or even better than) those of a state-of-the-art method [8], moreover, it runs more than 10 times faster.

5.3. Time and Complexity Analysis

With N as the pixel number of the image, we begin to analyze the time complexity of our SGF. For the internal aggregation (step 5$\sim$8 in algorithm 1), we need to run the linear cost aggregation [25] for all the superpixels once. This can be finished in $O(N)$. For neighbor aggregation (step 9$\sim$12 in Algorithm 1), assuming there are K neighborhoods for each superpixel (average $K \approx 7$ in practice), the neighbor aggregation will need $O(KN)$. Finally, for the weighted average calculation (step 13$\sim$17), we must run boxfilter for the minimum circumscribed rectangles $R(S)$ of each superpixel S to calculate the external weights ω_2 for all pixels. As a result, $\sum |R(S)| \approx \sum |S| = N$ times calculations are necessary.

For the segment graph construction, we use the linear SLIC [1] for superpixel segmentation. We find that the original SLIC runs 10 times iterative k-means which is a significant increase in computational cost in exchange for a small increase in accuracy of the segmentation results. So we just use 2 iterations for our SGF implementation. We also find that even though we use the fast SLIC for implementation, more than 2/3 of the running time is cost on the pre-segmentation. This means our SGF could run even faster if new faster superpixel approaches are employed.

We compare the time consumption of different local and global filters using 1-megapixel 24-bit color image as input. All the algorithms are executed using C++ on a 3.40 GHz Intel Core i7-3770 processor with 16 GB RAM (only one core is used). The running time data are given in Table 3. As shown in the test results, our SGF runs more than 10 times faster than the global counterparts and it could even compete with the linear GF [13] in efficiency. In the experiments, for some applications (like stereo matching and optical flow [19, 25]) which need hundreds or thousands of times of filtering processes, our SGF runs much faster than the GF [13] (Table 1 and 4) because our SGF only need to construct the segment graph once and the normalizing term K_p in Eq.(11) is actually not needed in these applications.

6. Discussion and Future Work

We present in this paper our segment graph based double weighted average image filter (SGF) and show its superiorities in many computer vision and graphics tasks. Our SGF achieves competitive performance when used for smoothing out high-contrast details and textures. Moreover, it possesses a linear running time. Our future work will focus on finding faster superpixel approaches to improve the performance. We will also try to implement the GPU version of our SGF which may improve the efficiency significantly.

Acknowledgements

This work was supported by China 863 program with No. 2015AA016402, and by NSFC with Nos. 61331018, 61332017, 91338202, 61271430.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, 2012. 3, 8

[2] L. Bao, Y. Song, Q. Yang, H. Yuan, and G. Wang. Tree filtering: Efficient structure-preserving smoothing with a minimum spanning tree. *IEEE Transactions on Image Processing*, 23(2):555–569, 2014. 1, 2, 3, 6, 7

[3] A. Buades, B. Coll, and J.-M. Morel. A non-local algorithm for image denoising. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 60–65 vol. 2, June 2005. 5, 6

[4] K. N. Chaudhury, D. Sage, and M. Unser. Fast bilateral filtering using trigonometric range kernels. *IEEE Transactions on Image Processing*, 20(12):3376–3382, 2011. 2

[5] X. Chen, S. B. Kang, J. Yang, and J. Yu. Fast patch-based denoising using approximated patch geodesic paths. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1211–1218, 2013. 2, 5, 6

[6] H. Cho, H. Lee, H. Kang, and S. Lee. Bilateral texture filtering. *ACM Trans. Graphics*, 33(4):128, 2014. 2, 6, 7

[7] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Trans. Graphics*, 27(3):67, 2008. 1, 2, 7

[8] D. Ferstl, C. Reinbacher, R. Ranftl, M. Rüther, and H. Bischof. Image guided depth upsampling using anisotropic total generalized variation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 993–1000, 2013. 7, 8

[9] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*, 48(3):533–551, 1994. 2

[10] E. S. Gastal and M. M. Oliveira. Domain transform for edge-aware image and video processing. *ACM Trans. Graphics*, 30(4):69, 2011. 7

[11] J. Grazzini and P. Soille. Edge-preserving smoothing using a similarity measure in adaptive geodesic neighbourhoods. *Pattern Recognition*, 42(10):2306–2316, 2009. 2, 5

[12] B. K. Gunturk. Fast bilateral filter with arbitrary range and domain kernels. *IEEE Transactions on Image Processing*, 20(9):2690–2696, 2011. 2

[13] K. He, J. Sun, and X. Tang. Guided image filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1397–1409, 2013. 1, 2, 3, 5, 7, 8

[14] L. Karacan, E. Erdem, and A. Erdem. Structure-preserving image smoothing via region covariances. *ACM Trans. Graphics*, 32(6):176, 2013. 2, 6, 7

[15] D. R. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM (JACM)*, 42(2):321–328, 1995. 2

[16] J. Lu, K. Shi, D. Min, L. Lin, and M. N. Do. Cross-based local multipoint filtering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 430–437, 2012. 1, 2, 5, 7

[17] F. Porikli. Integral histogram: A fast way to extract histograms in cartesian spaces. In *IEEE Conference Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 829–836, 2005. 5

[18] F. Porikli. Constant time o(1) bilateral filtering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2008. 5

[19] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3017–3024, 2011. 2, 7, 8

[20] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002. http://vision.middlebury.edu/stereo/eval/. 8

[21] X. Tan, C. Sun, and T. D. Pham. Multipoint filtering with local polynomial approximation and range guidance. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2941–2948, 2014. 5

[22] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *International Conference on Computer Vision*, pages 839–846, 1998. 6, 7, 8

[23] L. Xu, C. Lu, Y. Xu, and J. Jia. Image smoothing via l0 gradient minimization. *ACM Trans. Graphics*, 30(6):174, 2011. 1, 2, 7

[24] L. Xu, Q. Yan, Y. Xia, and J. Jia. Structure extraction from texture via natural variation measure. *ACM Trans. Graphics*, 2012. 2, 6, 7

[25] Q. Yang. A non-local cost aggregation method for stereo matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1402–1409, 2012. 7, 8

[26] Q. Yang. Recursive approximation of the bilateral filter. *IEEE transactions on image processing*, 2015. 2

[27] Q. Yang. Stereo matching using tree filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(4):834–846, April 2015. 2, 3, 4, 6

[28] Q. Yang, K.-H. Tan, and N. Ahuja. Real-time o (1) bilateral filtering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 557–564, 2009. 2

[29] Q. Zhang, X. Shen, L. Xu, and J. Jia. Rolling guidance filter. In *Computer Vision–ECCV 2014*, pages 815–830. Springer, 2014. 2, 7