Local LUT Upsampling for Acceleration of Edge-Preserving Filtering

Hiroshi Tajima, Teppei Tsubokawa, Yoshihiro Maeda, and Norishige Fukushima

Nagoya Institute of Technology, Japan

* https://fukushima.web.nitech.ac.jp/en/

Keywords: Edge-preserving filtering, local LUT upsampling, acceleration, bilateral filtering, $L_0$ smoothing

Abstract: Edge-preserving filters have been used in various applications in image processing. As the number of pixels of digital cameras has been increasing, the computational cost becomes higher, since the order of the filters depends on the image size. There are several acceleration approaches for the edge-preserving filtering; however, most approaches reduce the dependency of filtering kernel size to the processing time. In this paper, we propose a method to accelerate the edge-preserving filters for high-resolution images. The method subsamples an input image and then performs the edge-preserving filtering on the subsampled image. Our method then upsamples the subsampled image with the guidance, which is the high-resolution input images. For this upsampling, we generate per-pixel LUTs for high-precision upsampling. Experimental results show that the proposed method has higher performance than the conventional approaches.

1 INTRODUCTION

Edge-preserving filtering smooths images while maintaining the outline in the images. There are various edge-preserving filters for various purposes of image processing, such as bilateral filtering (Tomasi and Manduchi, 1998), non-local means filtering (Buades et al., 2005), DCT filtering (Yu and Sapiro, 2011), BM3D (Dabov et al., 2007), guided image filtering (He et al., 2010), domain transform filtering (Gastal and Oliveira, 2011), adaptive manifold filtering (Gastal and Oliveira, 2012), local Laplacian filtering (Paris et al., 2011), weighted least square filtering (Levin et al., 2004), and $L_0$ smoothing (Xu et al., 2011). The applications of the edge-preserving filters include noise removal (Buades et al., 2005), outline emphasis (Bae et al., 2006), high dynamic range imaging (Durand and Dorsey, 2002), hazy image removal (He et al., 2009), stereo matching (Hosni et al., 2013; Matsuo et al., 2015), free viewpoint imaging (Kodera et al., 2013), depth map enhancement (Matsuo et al., 2013).

The computational cost of edge-preserving filtering is the main issue. There are several acceleration approaches for each filter, such as bilateral filtering (Durand and Dorsey, 2002; Yang et al., 2009; Adams et al., 2010; Chaudhury et al., 2011; Chaudhury, 2011; Chaudhury, 2013; Sugimoto and Kamata, 2015; Sugimoto et al., 2016; Ghosh et al., 2018; Maeda et al., 2018b; Maeda et al., 2018a), non-local means filtering (Adams et al., 2010; Fukushima et al., 2015), local Laplacian filtering (Aubry et al., 2014), DCT filtering (Fujita et al., 2015), guided image filtering (Murooka et al., 2018; Fukushima et al., 2018b) and weighted least square filtering (Min et al., 2014). The computational time of each filter, however, depends on image resolution, and it is rapidly increasing, e.g., smartphone’s cameras even have 12M pixels. For such a high-resolution image, we require more acceleration techniques.

Processing with subsampling and then upsampling is the most straightforward approach to accelerate image processing. This approach dramatically reduces processing time; however, the accu-
racy of the approximation is also significantly decreased. Subsampling loses high-frequency and large edges in images; hence, the resulting images also lose the information. Different from super-resolution problems, subsampling/upsampling for acceleration can utilize a high-resolution input image as a guidance signal. Joint bilateral upsampling (Kopf et al., 2007) and guided image upsampling (He and Sun, 2015) utilize the high-resolution image for high-quality upsampling as extensions of joint bilateral filtering (Petschnigg et al., 2004; Eisemann and Durand, 2004). However, both upsampling are specific for accelerating bilateral filtering and guided image filtering. Note that More specific upsampling, e.g., depth map upsampling (Fukushima et al., 2016), improve the upsampling quality.

In this paper, we propose a method to accelerate arbitrary edge-preserving filters by using subsampling and upsampling with guidance high-resolution images. We named the proposed method local LUT upsampling. Figure 1 indicates the overview of the proposed method. In the proposed, edge-preserving filtering, which has the highest cost in the processing, is performed in the downsampled domain for acceleration. Then our method utilizes high-resolution information for accurate upsampling.

2 RELATED WORK

In this section, we review two edge-preserving filters used in the section of experimental results, i.e., bilateral filtering and $L_0$ smoothing. Also, we introduce guided image upsampling, which is an acceleration method for edge-preserving filtering.

2.1 Bilateral Filtering

Bilateral filtering is a standard edge-preserving filtering, and it is one of the finite impulse response (FIR) filters. The processing of the bilateral filter for an input image $I$ can be expressed as:

$$J_p = \frac{\sum_{q \in N_p} f_{st}(p,q)I_q}{\sum_{q \in N_p} f_{st}(p,q)},$$  

where $J$ is an output image of the bilateral filter, $p$ and $q$ are target and reference pixels, respectively. $N_p$ is a set of the neighboring pixel of the target pixel $p$. $f_{st}$ represents a weight function, and it is defined as:

$$f_{st}(p,q)=\exp\left(\frac{||p-q||^2}{\sigma^2} - \frac{||I_p-I_q||^2}{\sigma^2}\right),$$

where $||p-q||^2$ and $||I_p-I_q||^2$ show the difference of the spatial distance and luminance value between the target pixel and the reference pixel, respectively. $\sigma$ and $\sigma$ are smoothing parameters for the Gaussian distributions of the spatial distance and luminance difference, respectively. The bilateral filter smooths images with the range weight in addition to the spatial weight. As a result, a large weight is given to a reference pixel whose distance and luminance from target pixel is close.

2.2 $L_0$ Smoothing

$L_0$ smoothing can emphasize rough edges and can reduce fine edges. The $L_0$ smoothing smooths weak edge-parts in images more strongly than the bilateral filtering. Therefore, it is useful for extracting edges and generating non-photorealistic effects. An output is obtained by solving the following equation:

$$\min_{J,h,v} \left\{ \sum_p (J_p-I_p)^2 + \lambda C(h,v) + \beta ((\partial_J p - h_p)^2 + (\partial_J p - v_p)^2) \right\},$$  

where $I$ and $J$ are input and output images, respectively. $h_p$, $v_p$ are auxiliary variables corresponding to $\partial_J p$ and $\partial_J p$, where $p$ is a target pixel coordinate, respectively. $C(h,v)$ is the number of $p$ that $|h_p| + |v_p| \neq 0$, and $\beta$ is determined from the gradient of $(h,v)$. In order to solve Eq. (3), it is divided into two sub-problems. The first is computing $J$. Excluding the term not including $J$ from Eq. (3), the following equation is obtained:

$$E = \sum_p (J_p-I_p)^2 + \beta ((\partial_J p - h_p)^2 + (\partial_J p - v_p)^2).$$  

$J$ is obtained by minimizing equation (4). In this method, derivative operators are diagonalized after the fast Fourier transform (FFT) for speedup:

$$J = F \frac{1}{\beta} \left( F(I) + \beta (F(\partial_h p)^*F(h) + F(\partial_v p)^*F(v)) \right),$$

where $F$ is the FFT operator and $*$ denotes the complex conjugate. The second is computing $(h,v)$. The objective function for $(h,v)$ is defined as follows:

$$E^* = \sum_{p,h_p,v_p} \min \left\{ (h_p-\partial_h J_p)^2 + (v_p-\partial_v J_p)^2 + \frac{\lambda}{\beta} H(|h_p| + |v_p|) \right\},$$

where $H(|h_p| + |v_p|)$ is a binary function returning 1, if $|h_p| + |v_p| \neq 0$ and 0 otherwise. When Eq. (6) is rewritten to the function $E^*_p$ of each pixel $p,$ the following expression is obtained:

$$E^*_p = (h_p-\partial_h J_p)^2 + (v_p-\partial_v J_p)^2 + \frac{\lambda}{\beta} H(|h_p| + |v_p|),$$

which reaches its minimum $E^*_p$ under the condition:

$$(h_p,v_p) = \begin{cases} (0,0) & \text{if } (\partial_h J_p)^2 + (\partial_v J_p)^2 \leq \lambda/\beta \vspace{1em} \\
(\partial_h J_p, \partial_v J_p) & \text{otherwise.} \end{cases}$$
Algorithm 1 $L_0$ Gradient Minimization

**Input:** image $I$, smoothing weight $\lambda$, parameters $\beta_0$, $\beta_{\text{max}}$, and rate $\kappa$

**Initialization:** $J \leftarrow I$, $\beta \leftarrow \beta_0$, $i \leftarrow 0$

repeat

With $J(i)$, solve for $h_p(i)$ and $v_p(i)$ in Eq. (8)

With $h_p(i)$ and $v_p(i)$, solve for $J(i+1)$ with Eq. (5)

$\beta \leftarrow \kappa \beta, i \leftarrow i + 1$

until $\beta \geq \beta_{\text{max}}$

**Output:** result image $J$

After computing the minimum energy $E_p$ for each pixel $p$, calculating $\sum_p E_p$ becomes the global optimum for Eq. (6). The alternating minimization algorithm is sketched in Algorithm 1. Parameter $\beta$ is automatically adapted for each iteration. $\beta$ is starting from a small value $\beta_0$, and then it is multiplied by $\kappa$ each time.

2.3 Guided Image Upsampling

We now summarize guided image upsampling. The upsampling is an extension of the guided image filtering. The filtering converts local patches in an input image by a linear transformation of a guidance image. Let the guide signal be $G$, and it is possible to be $G = I$, where $I$ is an input image. The output patch $J_p$ should be satisfied for the following relation:

$$ J_p = a_k G_p + b_k, \forall p \in 0_k, $$

where $k$ indicates a center position of a rectangular patch $0_k$, and $p$ indicates a position of a pixel in the patch. $a_k$ and $b_k$ are coefficients for the linear transformation. The equation represents the coefficients linearly convert the guide signals in a patch.

The coefficients are calculated by a linear regression of the input signal $I$ and Eq. (9),

$$ \arg \min_{a_k, b_k} \sum_{p \in 0_k} ((a_k J_p + b_k - I_p)^2 + \varepsilon a_k^2). $$

(10)

The coefficients are estimated as follows:

$$ a_k = \frac{\text{cov}_k (G, I)}{\text{var}_k (G) + \varepsilon}, \quad b_k = I_k - a_k G_k, $$

(11)

where $\varepsilon$ indicates a parameter of smoothing degree. $\text{cov}_k$ and $\text{var}_k$ indicate the mean, variance, and covariance values of the patch $k$. The coefficients are overlapping in the output signals; thus, these coefficients are averaged;

$$ a_i = \frac{1}{|\omega|} \sum_{k \in \omega \rho} a_k, \quad b_i = \frac{1}{|\omega|} \sum_{k \in \omega \rho} b_k, $$

(12)

where $|\cdot|$ indicates the number of elements in the set.

Finally, the output is calculated as follows:

$$ J_i = a_i G_i + b_i. $$

(13)

For color filtering, let input, output, and guidance signals be $I = \{I', I^2, I^3\}$, $J_n (n = 1, 2, 3)$, and $G$, respectively. The per-channel output is defined as:

$$ J_p^n = a_p^n T G_p + b_p^n. $$

(14)

The coefficients $a_p^n, b_p^n$ are obtained as follows:

$$ a_p^n = \frac{\text{cov}_k (G, I^n)}{\text{var}_k (G) + \varepsilon E}, \quad b_p^n = b_k^n - a_k^n G_k, $$

(16)

where $E$ is an identity matrix. When the output signal is a color image, $\text{cov}_k$ is the covariance matrix of the patch in $I$ and $G$. Also, $\text{var}_k$ is the variance of the R, G, and B components, which will be covariance matrix, in the patch of $G$. The division of the matrix is calculated by multiplying the inverse matrix of the denominator from the left. We use box filtering for the calculation of per pixel mean, variance, and covariance. The fast implementation of the box filtering (Fukushima et al., 2018a) accelerates the filtering.

The guided image upsampling is an extension of guided image filtering and the upsampling is expressed as:

$$ J_p^n = S_1^{-1} (S_2 (a_p^n) T G_p + S_3 (b_p^n)), $$

(17)

where $S(\cdot)$ and $S^{-1}(\cdot)$ indicate subsampling and upsampling operators, respectively. The upsampling computes coefficients of $a_p$ and $b_p$ in the subsampled domain, and then upsamples the coefficients. Notice that we should keep high-resolution signal of $G$.

The coefficients in the downsampling domain is computed as follows:

$$ a_{p'_i}^{n'} := S (a_{p_i}^n), \quad b_{p'_i}^{n'} := S (b_{p_i}^n), $$

(18)

$$ a_{p'_i}^{n'} = \frac{1}{|\omega'|} \sum_{k \in \omega' \rho} a_k^n, \quad b_{p'_i}^{n'} = \frac{1}{|\omega'|} \sum_{k \in \omega' \rho} b_k^n, $$

(19)

$$ a_{k'_i}^{n'} = \frac{\text{cov}_k (G', I'^n)}{\text{var}_k (G') + \varepsilon E}, \quad b_{k'_i}^{n'} = b_k'^n - a_k'^n G_k', $$

(20)

where $p'_i$ indicates pixel coordinate in the downsampling domain. Also, $a'$ and $b'$ are coefficients in the downsampling domain, respectively $G' := S(G)$ and $I' := S(I)$ represent downsampling input and guidance images, respectively. The filtering radius is reduced according to the rate of subsampling; thus, $\rho'$ indicates a small rectangular patch, which is reshaped to fit the rate of subsampling.
3 PROPOSED METHOD

We propose an upsampling method guided by a high-resolution image for various edge-preserving filtering. The proposed method generates a per-pixel look-up table (LUT), which maps pixel values of the input image to that of the output image of the edge-preserving filtering. If we have the result of edge-preserving filtering, we can easily transform the input image into the edge-preserving result by one-to-one mapping. We generate the mapping function in the subsampled image domain for acceleration. The proposed method generates per-pixel LUTs for the function from the correspondence between a subsampled input and its filtering result.

Luminance values are highly correlated between pixels and its neighboring region in the image. Also, the correlation between the output of edge-preserving filtering and the input image becomes high. Therefore, in this method, a LUT for each pixel is created from a pair of low-resolution input and output images. Then, we refer the subsampled LUT to generate the high-resolution edge-preserving image. Initially, an input image $I$ is subsampled to generate a low-resolution input image $I_s$:

$$I_s = S_s(I),$$

where $S_s$ is a subsampling operator. Then, edge-preserving filter is applied for $I_s$ and the output $J_s$ is obtained as follows:

$$J_s = F(I_s),$$

where $F$ is an operator of arbitrary edge-preserving filtering. Then, we count frequency of one-to-one mapping between the images $J_s$ and $I_s$. Let $f_p(s, t)$ be a frequency map on a pixel $p$, where $s$ and $t$ are an intensity value on image $I_s$ and $J_s$, respectively. The frequency is defined by gathering around a pixel. The map $f$ is defined as follows:

$$f_p(s, t) = \sum_{q \in W_p} \delta(S_c(I_{qs}), s) \delta(S_c(J_{qs}), t),$$

where $\delta(a, b)$ indicates the Kronecker delta function. $p = (x, y)$ indicates a pixel position in the images of $I_s$ and $J_s$. $q$ is a neighboring pixel around $p$. $W_p$ indicates the set including $q$. $S_c(s)$ is a quantization function, which is divided by $l$. The output values have $256/l$ candidates. The redaction also accelerates the processing. We call the number of intensity candidates as the number of bins.

Next, we determine a local LUT on a pixel $p$. With the simplest solution, we select the maximum frequency argument from the $f$ by the winner-takes-all approach (WTA). The LUT under the downsampled domain is defined as follows:

$$T_p[s] = \arg\max_t f(s, t)$$

The operator $[\cdot]$ is an array operator in programming languages, such as C programming language. We cannot determine the value where $f(s, t) = 0$ for $f$. In this case, we linearly interpolate the values by the nearest non-zero values around $s$. For boundary condition, we set $T_p[0] = 0$ and $T_p[255] = 255$, when $f(s, t) = 0$. The solution cannot keep monotonicity for the LUT. The fact sometimes generates negative edges.

To keep the monotonicity, we can solve the problem by using dynamic programming (DP). We assume that $T_p[0] = 0$ and $T_p[n] > T_p[n]$. We define a cost function $C(s, t)$ for the DP under this condition:

$$C(s, t) = \max\left(C(s-1, t-1) + O, C(s-1, t), C(s, t-1)\right) + f(s, t)$$

where $O = f(0, 0)$, $C(s, t) = 0$ for $s < 0$ and $t < 0$. The parameter $O$ represent an offset value to enforce diagonal matching. The cost function can be recursively updated. After filling the cost function, then
we trace back the function to determine a solid pass.

$$T_{pp} = \arg \text{DP} C(s,t),$$

where \textit{pass} shows the tracing back pass of the cost matrix \( C \) by using DP. Figure 2 shows the frequency map and the resulting LUTs obtained by WTA and DP. After WTA and DP, the matrix indicates one-to-one mapping. We call this per-pixel LUT \( T_{pp} \) as local LUT.

The LUT has three-dimensional information, such as \( n, p \equiv (x, y) \); however, all elements are subsampled. It has not enough to map an input image into a high-resolution image. Therefore, we upsample the LUT by using tri-linear upsampling. The upsampling is defined as follows:

$$T_p = S_{c}^{-1}(S_{s}^{-1}(T_{pp})),$$

where \( S_{c}^{-1} \) and \( S_{s}^{-1} \) is upsampling operator for the intensity and spatial domain, respectively. \( p \) indicates the image coordinate of pixel in the upsampled domain.

Finally, the output image is referred from the local LUT and input intensity \( I_p \). The output is defined by:

$$J_p = T_p[I_p].$$

### 4 EXPERIMENTAL RESULTS

We approximated two edge-preserving filters, such as iterative bilateral filtering and \( L_0 \) smoothing by subsampling based acceleration methods. These filters can mostly smooth images; however, the computational cost is high. We compared the proposed method of the local LUT upsampling with the conventional method of the cubic upsampling and guided image upsampling by approximation accuracy and computational time. Also, we compared the proposed method with naïve implementation, which does not subsample images, in computational time. We utilized high-resolution test images, which are shown in Fig. 3.\(^1\) We used two metrics of PSNR and SSIM (Wang et al., 2004) for accuracy evaluation, and we regarded the results of the naïve filtering as ground truth results. We implemented the proposed method by C++ with OpenMP parallelization. We also used Intel IPP for the cubic upsampling, which is optimized by AVX/AVX2. For guided image upsampling, we also optimized by AVX/AVX2 with OpenCV functions. For downsampling in the pro-

\(^1\)http://imagecompression.info
posed method, we use the nearest neighbor downsampling. This downsampling has better performance than the cubic interpolation for our method. We used OpenCV and Intel IPP for the operation with cv::INTER_NN option. The used computer was Intel Core i7 6700 (3.40 GHz) and compiled by Visual Studio 2017. We used \( r = 3 \) for local LUT upsampling, \( r = 10 \), \( \text{iteration} = 10 \), \( \sigma_r = 10, \sigma_c = 20 \) for iterative bilateral filtering, and \( \lambda = 0.01 \) and \( \kappa = 1.5 \) for \( L_0 \) smoothing.

Figures 4, 5, and 6 show the PSNR/SSIM accuracy and computational time of iterative bilateral filtering and \( L_0 \) smoothing for large images. The parameters of the proposed method are \( l = 2 \) and downsampling ratio is \( 8 \times 8 \). The computational time is an average of 10 trails. The local LUT upsampling has drastically higher PSNR than the cubic interpolation for each image. Guided image upsampling is slower and lower PSNR than the proposed method. The computation time of the local LUT is \( \times 100 \) faster than the naïve implementation; however, the cubic upsampling is \( \times 10000 \) faster. Notice that the code of the proposed method is not vectorized, such as AVX/AVX2; thus, the proposed can speed up by programming, while Intel’s code fully optimizes the cubic upsampling. We now optimize the code for the proposed method while reviewing period.

Figures 7 and 8 show the relationship between spatial/range subsampling ratio and PSNR for iterative bilateral filtering and \( L_0 \) smoothing. The result shows that 1/4 range downsampling (64 bin) and \( 8 \times 8 \) spatial downsampling do not notable decrease the PSNR quality.

Figures 9 and 10 show the trade-off between the PSNR accuracy and computational time by changing the resizing ratio from 1/16 to 1/1024. We also change the number of bins for local LUT upsampling. These results show that the proposed method has the best trade-off than the cubic interpolation and guided image upsampling.

Figures 11 and 12 depict input image and results of each upsampling method for iterative bilateral filtering and \( L_0 \) smoothing, respectively.

5 CONCLUSIONS

In this paper, we proposed an acceleration method for edge-preserving filtering with image upsampling. We call the upsampling as local LUT upsampling. The local LUT upsampling has higher approximation accuracy than the conventional approaches of cubic and guided image upsampling. Also, the local LUT upsampling accelerates \( \times 100 \) faster than the naïve implementation.

ACKNOWLEDGEMENTS

This work was supported by JSPS KAKENHI (JP17H01764, 18K18076).

REFERENCES


Figure 11: Results of iterative bilateral filtering.

Figure 12: Results of $L_0$ smoothing.


