FAST BILATERAL FILTERING BY ADAPTING BLOCK SIZE

Wei Yu[1], *Franz Franchetti*[1], *James C. Hoe*[1] , *Yao-Jen Chang*[2], *Tsuhan Chen*[2]

[1]Carnegie Mellon University, [2] Cornell University

ABSTRACT

Direct implementations of bilateral filtering show $O(r^2)$ computational complexity per pixel, where r is the filter window radius. Several lower complexity methods have been developed. State-of-the-art low complexity algorithm is an $O(1)$ bilateral filtering, in which computational cost per pixel is nearly constant for large image size. Although the overall computational complexity does not go up with the window radius, it is linearly proportional to the number of quantization levels of bilateral filtering computed per pixel in the algorithm. In this paper, we show that overall runtime depends on two factors, computing time per pixel per level and average number of levels per pixel. We explain a fundamental trade-off between these two factors, which can be controlled by adjusting block size. We establish a model to estimate run time and search for the optimal block size. Using this model, we demonstrate an average speedup of 1.2–26.0x over the pervious method for typical bilateral filtering parameters.

***Index Terms*—** bilateral filtering, algorithm complexity, real time

1. INTRODUCTION

Bilateral filtering is a non-linear filter introduced by Tomasi et al. in 1998 [1]. It smoothes out an image by averaging neighborhood pixels like a Gaussian filter, but preserves sharp edges by decreasing weights of pixels when the intensity difference is large. Bilateral filtering is useful in many image processing and vision applications such as image denoising [1, 2], tone mapping [3], and stereo matching [4].

Direct implementation of bilateral filtering from definition is computationally expensive. There are generally three directions to make an algorithm run faster. First, design lower complexity algorithms without degrading accuracy; second, optimize code extensively for a given algorithm; third, optimize code on a more powerful hardware platform. In this paper, our focus is along the first path.

Related work. The computational complexity per pixel for direct implementation is $O(r^2)$, where r is the filter window radius. Recently, several methods have been proposed to reduce the arithmetic complexity of the algorithm. Porikli et al. [5] proposed a constant time bilateral filtering with respect to filter size for three types of bilateral filters. Quantization of image intensity in this method may significantly degrade the quality of the filtering output. Also, memory footprint requirement is large for storing the integral histograms. Yang et al. [6] propose a $O(1)$ bilateral filtering which extends Durand and Dorsey's piecewise-linear bilateral filtering method [3]. It can achieve $O(1)$ complexity with arbitrary spatial and arbitrary range kernels (assuming the exact or approximated spatial filtering is $O(1)$ complexity), with much better accuracy and less memory footprint than [5]. In [6], they discretize the image intensities. For each quantization value, they compute a linear spatial filtering, whose output is defined as Principle Bilateral Filtered Image Component (PBFIC). The final bilateral filtering output is a linear interpolation between the two closest PBFICs. For typical parameter settings (see section 4), processing time of [6] on a 2.67GHz CPU with 2GB RAM for image of size 600×800 is on the order of tens of milliseconds to several seconds.

Overview of proposed method. In this paper, we propose an extension of [6], to further reduce the run time based on an important trade-off we found. The overall computing time depends on two factors, the computational cost per pixel per PBFIC, and the average number of PBFICs computed per pixel. $O(1)$ cost per pixel only reflects the first factor. We will show there is a fundamental trade-off between these two factors, and changing the block size can control the trade-off. Block size of 1×1 corresponds to direct implementation, and block size up to the original image size corresponds to the implementation in [6]. The optimal block size should be somewhere in between for general cases. We build a model to predict run time given a fixed block size, and use this model to search for the best block size.

Synopsis. In the following, we briefly review the $O(1)$ bilateral filtering proposed in [6], and explains the fundamental trade-off in Section 2. Section 3 details a model to estimate the computing time. Section 4 shows experiment results and Section 5 concludes.

2. OBSERVATION OF A FUNDAMENTAL TRADE-OFF

From the definition of bilateral filtering, it is a compound of linear spatial filtering and non-linear range filtering. Spatial filtering kernel is usually a simple box filtering kernel or a Gaussian kernel, both having $O(1)$ (approximate) algorithms.

Range filtering kernel is usually a Gaussian kernel that assigns lower weight to pixels with large intensity difference. The filter output of a pixel **x** is

$$I^B(\mathbf{x}) = \frac{\sum_{\mathbf{y}\in N(\mathbf{x})} f_s(\mathbf{y},\mathbf{x}) \cdot f_r(I(\mathbf{y}), I(\mathbf{x})) \cdot I(\mathbf{y})}{\sum_{\mathbf{y}\in N(\mathbf{x})} f_s(\mathbf{y},\mathbf{x}) \cdot f_r(I(\mathbf{y}), I(\mathbf{x}))} \quad (1)$$

In Eq. (1), $I(\mathbf{x})$ is intensity of pixel **x**. $f_s(\mathbf{y},\mathbf{x})$ and $f_r(I(\mathbf{y}), I(\mathbf{x}))$ are spatial and range filter kernels. $N(\mathbf{x})$ denotes a neighborhood window around **x**. The key idea in [6] is to turn both dividend and divisor in Eq. (1) into linear spatial filters by fixing $I(\mathbf{x})$ to be constant. This is achieved by quantizing intensity value into multiple levels and computing so-called Principle Bilateral Filtered Image Component (PBFIC). Assuming intensity is quantized into L levels $I_0, I_1, ..., I_{L-1}$. The l-th PBFIC is computed as

$$I_l^B(\mathbf{x}) = \frac{\sum_{\mathbf{y}\in N(\mathbf{x})} f_s(\mathbf{y},\mathbf{x}) \cdot f_r(I(\mathbf{y}), I_l) \cdot I(\mathbf{y})}{\sum_{\mathbf{y}\in N(\mathbf{x})} f_s(\mathbf{y},\mathbf{x}) \cdot f_r(I(\mathbf{y}), I_l)} \quad (2)$$

The final output $I^B(\mathbf{x})$ is interpolated from two closest PBFICs. In the following, we assume spatial filtering to be simple box filtering to simplify the complexity analysis. But the idea in this paper is applicable to other $O(1)$ spatial filters as well. The range filter is assumed to be commonly used Gaussian filter $f_r(I_1, I_2) = \exp(-(I_1 - I_2)^2/(\sigma_r^2))$. Here a constant is omitted because it does not affect the final output.

Given $O(1)$ spatial filtering complexity for computing $I_l^B(\mathbf{x})$, the total complexity of computing bilateral filtering is $O(1)$ per pixel with respect to filter window radius r. However, the computational complexity also grows linearly with the number of quantization levels L, and L is roughly proportional to the intensity range of the image for a given accuracy requirement. We use e to control accuracy. We fix the lowest and highest quantization values to I_{min} and I_{max}, which are minimal and maximal intensity values in the image. The number of quantization levels

$$L = \max(\min(K, 2), \min(K, \text{round}(\frac{K}{256 \cdot \sigma_r \cdot e}))) \quad (3)$$

K is the intensity range, which equals $I_{max} - I_{min} + 1$. When $e \le \frac{1}{256 \cdot \sigma_r}$, L equals K, the bilateral filtering output is exact. When e increases, accuracy decreases. We fix e to 1.0, where we observe PSNR>50dB for almost all images. This is considered of the same visual quality to the exact output.

Let's partition the image into blocks of size $b_h \times b_w$, and apply the above method to each block. Here we do a rough estimation of the run time, without considering interpolation step. More accurate modeling can be found in Section 3. The computing time is roughly proportional to $T_a \cdot L_a$, where T_a is the average computing time per pixel per PBFIC level and L_a is the average number of PBFIC levels per block. T_a for box filtering can be roughly estimated as

$$T_a \approx \frac{C \cdot (b_h + 2r) \cdot (b_w + 2r)}{(b_h \cdot b_w)} \quad (4)$$

C is a constant depending on computing platforms. Generally speaking, most images contain a large portion of slowly varying regions, therefore intensity range within small blocks are smaller than the intensity range of the whole image.

Fig. 1 shows for an example image on the left side, how T_a and L_a changes with the block size $b_h \times b_w$. The trend is similar for other images. Here $L = K$ for each block. Image size is 512×512. We test square blocks, and $\log_2(b_w)$ varies from 0 to 9. As expected, T_a ($\Diamond$) decreases with the increasing block size, and L_a ($\Box$) increases with the increasing block size. For different r, $T_a \cdot L_a$ ($\bullet$) reaches minimum at different b_w. It is no surprise to see that for small r, L_a is the dominating factor and prefers small w_b; for large r, T_a is the dominating factor and prefers large w_b. When block size equals image size, it becomes exactly the $O(1)$ bilateral filtering in [6]. When block size equals 1×1, every block has only one intensity and $L_a = 1$. The filtering cost is $O(r^2)$ per pixel, which degenerates to the naive implementation of bilateral filtering.

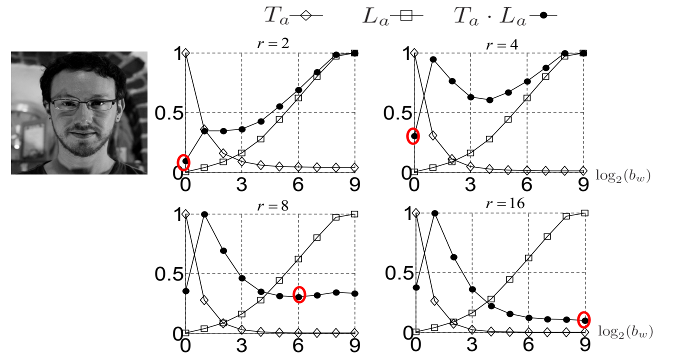

Fig. 1. Tradeoff between T_a and L_a for varying $r = 2, 4, 8, 16$. Here $L = K$ for each block. x-axis shows $\log_2(b_w)$. We only test square blocks of size $b_w \times b_w$. y-axis shows T_a, L_a and $T_a \cdot L_a$ normalized to their maximum values. Optimal b_w for $T_a \cdot L_a$ is circled.

The above analysis is an approximation of run time. Fig. 2 further demonstrates the relationship between run time and block size by measuring the real run time for varying block size. We use the same example image. Block size is the same as in Fig. 1. $256\sigma_r$ varies in $\{1, 16\}$. The code of [6] is publicly available on website. We simply modify the code such that bilateral filtering is looped on each block. When $256\sigma_r = 1$, $L = K$, the trend of normalized run time with respect to b_w is close to Fig. 1. When $256\sigma_r = 16$, which is a typical setting for image denoising, increase of L_a is much slower than K for small b_w, but T_a remains a decreasing func-

tion of increasing b_w. That is why we observe decreasing run time for small b_w values.

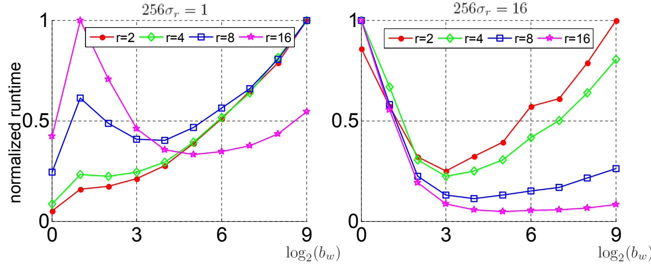

Fig. 2. Measured run time vs. block width, for varying $r = 2, 4, 8, 16$. x-axis shows $\log_2(b_w)$. y-axis shows normalized run time to the maximum values.(best view in color)

3. PROPOSED MODEL

In this section, we build a much more detailed model to estimate relationship between run time and block size. This model should be accurate enough to enable us to search for optimal or near optimal block size. It should also be simple so that modeling overhead is low.

The total run time of bilateral filtering for a block of size $b_h \times b_w$ includes four parts.

- **Part 1**: time to compute $f_r(I(\mathbf{y}), I_l))$ and $f_r(I(\mathbf{y}), I_l) \cdot I(\mathbf{y})$ in Eq. (2). $f_r(I(\mathbf{y}), I_l)$ can be pre-computed and stored in a table. For 8-bit intensities, only 256 table entries are needed. The computing time can be estimated as $T_1 = C_1(b_h + 2r)(b_w + 2r)L$.
- **Part 2**: time to compute dividend and divisor in Eq. (2). Both are box spatial filtering, which can be decomposed into horizontal sum filter followed by a vertical sum filter. For horizontal filter, each row takes $2(b_w + r - 1)$ additions/subtractions (after computing summation of neighbors for the first pixel, for consecutive pixels, summation is computed by adding a new pixel and subtracting an old pixel from the previous summation). Total number of rows is $b_h + 2r$. For vertical filter, each column takes $2(b_h + r - 1)$ additions/subtractions. Total number of columns is b_w. So the time is estimated as $T_2 = C_2((b_w + r - 1)(b_h + 2r) + (b_h + r - 1)b_w)L$.
- **Part 3**: time to compute division in Eq. (2), estimated as $T_3 = C_3 b_h b_w L$.
- **Part 4**: time to interpolate. The way we implement interpolation is that after computing the l-th level of PBFIC, we check all pixels in the block if $I(\mathbf{x})$ is in $[I_{l-1}, I_l)$. If so, then interpolate as Eq. (3). So roughly speaking, every pixel is checked for $L - 1$ times, and only one time its final bilater filtering output is interpolated. The time is estimated as $T_4 = C_4 b_h b_w (L - 2) + C_5 b_h b_w$.

All parameters C_1, C_2, C_3, C_4, and C_5 are platform dependent, and should be calibrated for each hardware platform. We use the example image in Fig. 1 and varying b_h, b_w to calibrate those parameters. Both $\log_2 b_h$ and $\log_2 b_w$ vary from 2 to $\log_2 \min(w, h)$, and they can be different. For each of the four parts, we use RDTSC( ) function (Intel time-stamp counter) to measure elapsed time. Fitting C_1, C_2, C_3 are simple, e.g. C_3 is average of $T_3/(b_h b_w L)$. C_4, C_5 are estimated using `robustfit()` in Matlab.

We do the experiment on a Dell XPS 720 desktop, with Intel Core2 Duo E6750 2.67GHz CPU and 2GB RAM. Estimated $C_1 = 28.56$, $C_2 = 17.96$, $C_3 = 32.45$, $C_4 = 32.52$, $C_5 = 140.48$. Fig. 3 shows for each of the four parts, the measured run time (T_m) and run time predicted from the model (T_p). Average prediction error ($|\frac{T_p}{T_m} - 1|$) of part 1 to 4 are 0.01, 0.05, 0.03, 0.10, standard deviations are 0.013, 0.055, 0.027, 0.113. Using the same C_1–C_5 on other images gives very similar results.

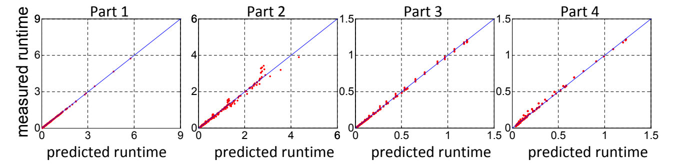

Fig. 3. Measured run time vs. predicted run time from model for part 1–4 (Unit for both run time is Giga-cycles).

4. EXPERIMENTAL RESULT

In this section, we show how well the model works, and how much speedup can be achieved compared to the method proposed in [6]. The dataset includes randomly downloaded 50 images from website, size ranging from 100×120 to 600×800. For bilateral filtering parameters r and σ_r, we test r in $\{2, 4, 8, 16\}$, $256\sigma_r$ in $\{1, 2, 4, 8, 16, 32\}$, which represent typical range of these parameters.

About the model, we are concerned about the following questions.

- How much extra cost does this model introduce?
- Is the optimal block size searched by this model matches the real optimal one?
- If answer to the second question is no, then how much worse is the searched block size from the real one?

For the first question, modeling time involves collecting number of quantization levels for a given block size, which needs the information of I_{min} and I_{max} for every block. We limit our search range to all square blocks ($b_w \times b_w$, $\log_2 b_w$ varies from 0 to $\log_2 \min(w, h)$), and all blocks whose height is twice of width ($2b_w \times b_w$, $\log_2 b_w$ varies from 0 to $\log_2 \min(w, h) - 1$). For these block sizes, we can

sort them in an increasing order, and quantization levels of a certain block size can be easily built from previous block size because current block contains two small previous blocks. The measured modeling time is low. We found for about 97% of all cases, modeling time is less than 10% of the measured run time for the optimal block size. The second and third question relates to modeling accuracy. Fig. 4 (a) shows predicted run time from modeling (T_p) vs. measured run time (T_m) for all images and parameter settings we tested. Average prediction error ($|\frac{T_p}{T_m} - 1|$) is 0.04, standard deviation is 0.038. Fig. 4 (b) shows measured run time of predicted optimal block size and the true optimal block size. For 80.2% of all cases, the optimal block size predicted by the model matches the true optimal block size. However, for only about 0.6% of all cases, measured run time of the predicted optimal block size is 10% greater than that of the true optimal block size.

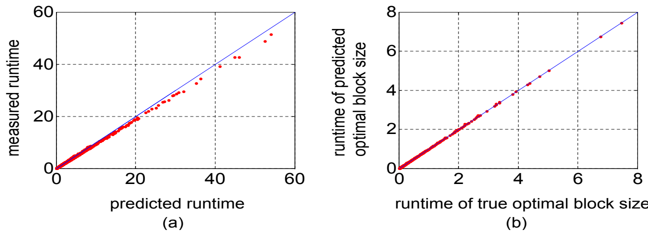

Fig. 4. (a) Measured run time vs. predicted run time from model; (b) run time of the predicted optimal block size vs. run time of the true optimal block size. (Unit for all run time is Giga-cycles).

Next we will show the speedup of using this model over using full image size as block size, which is exactly the $O(1)$ bilateral filtering in [6]. Fig. 5 shows for varying σ_r, average speedup over all images vs. r. In each subfigure, the line (with □) shows the upper bound of speedup, which is the speedup by using the true optimal block size. The line (with $\times$) shows the achieved speedup by using the optimal block size predicted by the model. Here we take into account the model overhead. The speedup decreases when r increases. This is consistent with Fig. 1. When r is large, T_a becomes the dominating factor in run time, and encourages large block size. For large block size, both T_a and L_a changes slowly with respect to the block size, so the observed speedup is small. The speedup also decreases when σ_r increases, especially for small r. When r is small, L_a becomes the dominating factor. However the difference of L_a for small and large block sizes becomes smaller when σ_r gets larger. For example, $L_a = 1$ for 1×1 block, but $L_a \approx \frac{256}{256\sigma_r}$ for block equaling image size. That is why the observed speedup decreases with increasing σ_r. In summary, we observe average speedup of 1.2–26.0x on all images for typical parameter settings.

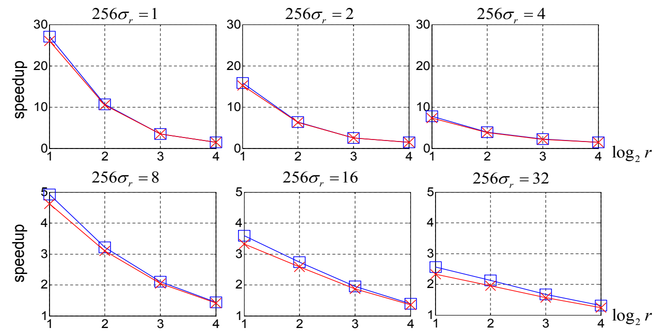

Fig. 5. Average speedup vs. r over the $O(1)$ bilateral filtering in [6] for varying $256\sigma_r$. The line (with □) shows upper bound of speedup by using the true optimal block size, and the line (with $\times$) shows real speedup of the proposed model.

5. CONCLUSION

In this paper, we show a fundamental trade-off between two factors in the $O(1)$ bilateral filtering method in[6]. The two factors are the computing time per pixel per quantization level and the average number of quantization levels per pixel. The trade-off can be controlled by varying block size. We build a timing model to estimate run time for a given block size. Experiments show the model gives reasonably accurate estimation of the run time with negligible overhead. More importantly, run time of the optimal block size predicted by the model is very close to the run time of the true optimal block size for most cases. Experiments also demonstrates an average speedup of 1.2–26.0x on all images for typical parameter settings. We expect to see even more significant speedup for HDR (high dynamic range) images because intensity ranges of HDR images are usually much larger than 256, which is a typical range of 8-bit digital images.

6. REFERENCES

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

[2] A. Buades, B. Coll, and Morel J.M., "A review of image denoising algorithms, with a new one," in *Multiscale Modeling and Simulation*, 2005.

[3] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," in *Proc. of SIGGRAPH*, 2002.

[4] K. J. Yoon and I. S. Kweon, "Adaptive support-weight approach for correspondence search," in *IEEE Trans on PAMI*, 2006.

[5] Fatih Porikli, "Constant time $o(1)$ bilateral filtering," in *Proc. of CVPR*, 2008.

[6] Qingxiong Yang, Kar-Han Tan, and N. Ahuja, "Real-time $o(1)$ bilateral filtering," in *Proc. of CVPR*, 2009.