TURBO DENOISING FOR MOBILE PHOTOGRAPHIC APPLICATIONS

Tak-Shing Wong, Peyman Milanfar

Google Research
{wilwong, milanfar}@google.com

ABSTRACT

We propose a new denoising algorithm for camera pipelines and other photographic applications. We aim for a scheme that is (1) fast enough to be practical even for mobile devices, and (2) handles the realistic content dependent noise in real camera captures. Our scheme consists of a simple two-stage non-linear processing. We introduce a new form of boosting/blending which proves to be very effective in restoring the details lost in the first denoising stage. We also employ IIR filtering to significantly reduce the computation time. Further, we incorporate a novel noise model to address the content dependent noise. For realistic camera noise, our results are competitive with BM3D, but with nearly 400 times speedup.

Index Terms— denoise, boosting, camera pipeline, noise model

1. INTRODUCTION

Denoising is a key step in an imaging pipeline for improving image quality. Due to its importance, many works have addressed this problem in the past [1, 2, 3, 4, 5]. However, while these state-of-the-art algorithms produce very high quality denoising results, they usually suffer from high computational complexity and are unsuitable for many applications. In this paper, we propose a fast denoising algorithm which consists of a two-stage non-iterative, non-linear processing. Experimental results show that the denoising performance of our scheme is close in quality to one of the best quality denoising algorithms, BM3D [1], but it executes significantly faster.

2. TURBO DENOISING

The basic structure of our scheme is illustrated in Fig. 1, which consists of two processing stages. In the first smoothing stage, we apply a base denoising filter to suppress the noise. In the second stage (boosting), we compute the residual of the first stage and detect from it image details that are likely suppressed by the denoising filter. The detected details will then be blended back to form the final result.

2.1. Denoising Filter

The primary objective of the first stage is to suppress the image noise. While many image filters proposed in the past may fulfill this objective, we make our choice based on a few criteria. First, the selected filter must be effective in removing the noise. Second, it has to be computationally efficient so that the overall scheme is still practical. Third, a filter that is, to some extent, edge and detail aware is more desirable in order to avoid the need of relying excessively on the second stage to restore the lost details. The last two criteria essentially eliminate many of the options from existing literature. For example, fast approximation of bilateral filter, including bilateral grid [6] and permutohedral lattice [7], require more than a second to process a 1.5 megapixel (MP) color image on an 2.67 GHz Intel Core i7 920 processor single-threaded [7], and their complexities are dependent on the filter kernel size. We adopt the IIR version of Domain Transform filter [8] as our denoising filter, which is edge-aware and computationally efficient. Its IIR structure also makes the complexity independent of the effective kernel width of the filter, which is important when the noise level is high.

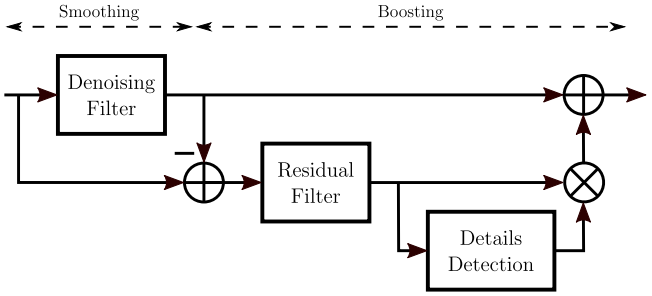


Fig. 1: Turbo denoising

Here we briefly summarize the operation of the IIR Domain Transform (IIR-DT) filter. The scheme decomposes the spatial filtering into alternating vertical (top-to-bottom, bottom-to-top) and horizontal (left-to-right, right-to-left) 1D IIR filters. Each pass of the IIR filter, left-to-right for example, is implemented according to the difference equation

$$J[n] = (1 - a^{d[n]})I[n] + a^{d[n]}J[n-1], \tag{1}$$

where $I[n]$ and $J[n]$ are the intensities of the n-th input pixel and output pixel. The constant a is determined from the spatial scale parameter σ_s as $a = \exp\left(-\sqrt{2}/\sigma_s\right)$. The quantity $d[n]$ approximates the local geodesic distance from the n-th pixel to its neighbors, and is given by

$$d[n] = 1 + \frac{\sigma_s}{\sigma_r}\sum_{j=1}^{C}\left|I_j'[n]\right|, \tag{2}$$

where $I_j'[n]$ is the derivative, along the horizontal or vertical direction (depending on the direction of the IIR filter), of the j-th channel of the image; and σ_r is the photometric scale parameter.

To avoid stripe-like artifacts due to the 1D IIR filters [8], the IIR-DT filter further decomposes the filter into $N = 3$ or more iterations using successively decreasing spatial parameter $\sigma_{s,i} = \sigma_s \frac{2^{N-i}\sqrt{3}}{\sqrt{4^N - 1}}$, for $i = 1, \ldots, N$, where their squared sum across all iterations equals to σ_s^2. We used $\sigma_s = 2.5, \sigma_r = 6.5$ in our experiments.

2.2. Boosting

The idea of boosting or twicing [9, 10] makes use of the filtering residual to improve the quality of image denoising or reconstruction. In the basic formulation, the k-th iteration of boosting is obtained by

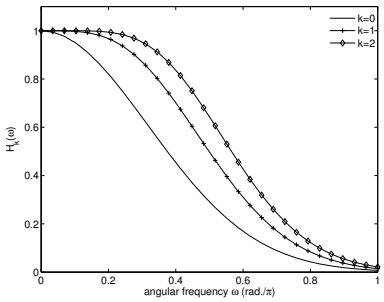


Fig. 2: Frequency responses of $\mathbf{H}_k = \sum_{j=0}^{k} \mathbf{W}(\mathbf{I}-\mathbf{W})^j$, for a linear Gaussian filter $\mathbf{W}$ with $\sigma = 1$.

adding the filtered residual to the result of the previous iteration. More specifically, given a signal $\mathbf{y}$, a filter operator $\mathbf{W}$, and $\mathbf{z}_0 = \mathbf{W}\mathbf{y}$, boosting is defined by the following recursion:

$$\mathbf{z}_k = \mathbf{z}_{k-1} + \mathbf{W}(\mathbf{y} - \mathbf{z}_{k-1}). \tag{3}$$

Here, the term $\mathbf{r}_k = \mathbf{y} - \mathbf{z}_{k-1}$ is the residual from the previous iteration. Applying Eq. (3) repeatedly, we can express $\mathbf{z}_k$ in terms of $\mathbf{y}$ directly:

$$\mathbf{z}_k = \sum_{j=0}^{k} \mathbf{W}(\mathbf{I}-\mathbf{W})^j \mathbf{y}. \tag{4}$$

As an illustration, Fig. 2 shows the frequency responses of a linear Gaussian filter $\mathbf{W}$ and after the first two boosting iterations. For this simple linear example, boosting successively adds different subbands back to the filter $\mathbf{W}$.

If the boosting components are selectively recovered based on the local signal and noise content, the boosting step would improve the overall quality by restoring the image details detected from the residual. Therefore, we apply a modified boosting step

$$\mathbf{z}_k = \mathbf{z}_{k-1} + \mathbf{S}_k\mathbf{W}(\mathbf{y} - \mathbf{z}_{k-1}), \tag{5}$$

where $\mathbf{S}_k$ is a diagonal matrix representing an image mask. We compute the matrix $\mathbf{S}_k$ separately from the filtered residual. Its diagonal elements, in the range of $[0, 1]$, will serve as a confidence score of whether the local region in the filtered residual contains image structure or noise.

To limit the computation, we extend the highly efficient census transform [11] to compute the structural mask $\mathbf{S}_k$. The census transform computes a 8-bit binary string at each pixel to summarize the structure in a 3×3 local window. Each neighbor pixel q_i is compared to the center pixel p,

$$c_i = \begin{cases} 0 & \text{if } q_i \le p + \delta \\ 1 & \text{if } q_i > p + \delta \end{cases}, \tag{6}$$

where δ is a small constant to improve the robustness of the transform against noise. The binary values c_i for the eight neighbors are then concatenated and typically encoded as an integer. For each of the 256 census transform values, we define a structural score based on the structure in the 3×3 binary pattern and create a look-up table.

We compute the structural mask $\mathbf{S}_k$ as shown in Fig. 3. First, we compute the census transform at each pixel of the filtered residual. Next, the 8-bit census transform is used to look up a structural score, between 0 and 1, for the pixel. Finally, we apply a Gaussian filter to obtain the final structural mask. The Gaussian filter essentially implements a version of local majority vote by the structural score. Real structures in the residual image typically have size at least a few pixels so that neighboring pixels forming the structure are likely correlated and have large structural scores. If a noisy pixel is accidentally scored high, it will be more likely to be an isolated high score pixel, and will be blurred out by the Gaussian filter.

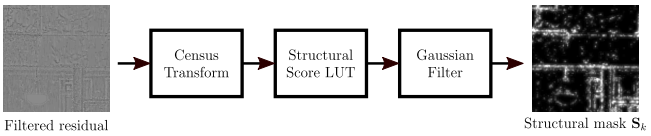


Fig. 3: Computation of the structural mask $\mathbf{S}_k$.

(a) Noisy image (b) Structural mask of Y channel

(c) Denoising filter output (d) Boosting output

Fig. 4: (a) Noisy image (Nexus 6P, ISO-1203), (b) structural mask $\mathbf{S}_k$, (c) denoising filter output, and (d) boosting output.

In addition to introducing the structural mask, our scheme also deviates from basic boosting in that it uses the edge-aware, nonlinear IIR-DT filter (Section 2.1) for the filter $\mathbf{W}$; that is, the residual filter in Fig. 1. Further, for the residual filter, we use the denoising filter output as the guiding image to compute $d[n]$ of Eq. (2), which improves the separation of structures from the noise.

Fig. 4 shows the effect of boosting visually. The noisy cropped image (Fig. 4a) was captured by Nexus 6P at ISO-1203. The denoising filter output (Fig. 4c) removes most of the noise but also causes too much blur. The final boosting output (Fig. 4d) restores the sharpness of the edges. Results in Table 1 also demonstrate the benefits of boosting (see Section 3 for experimental details). Columns 2-3 (PSNR) and columns 5-6 (SSIM) show the numerical results of Turbo Denoising after smoothing (no boosting) and after boosting. For all test images, boosting improves both PSNR and SSIM values.

2.3. Noise Model

We incorporate a noise model to accommodate the complex noise characteristics of camera captured images. In our experiment, the images are in the sRGB color space, having been processed by the camera pipeline. We apply our scheme in the YUV space to decorrelate the color channels. We employ a simple pipeline model and a sensor noise model to construct a noise model for the YUV data. Noise variance prediction consists of a table look up and scaling.

Our sensor model estimates the noise variance at a pixel as

$$\sigma_n^2 = g_1 I + g_2, \tag{7}$$

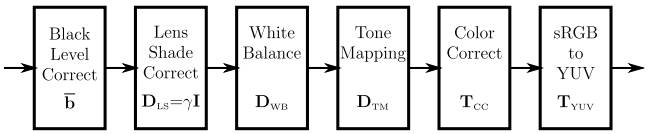


Fig. 5: A simplified camera pipeline model.

where I is the digital number, g_1 and g_2 are model parameters determined from sensor specification or calibration [12]. The first term $g_1 I$ accounts for the shot noise. The second term g_2 represents an additive Gaussian noise component to approximate the aggregate effect of thermal noise and read-out noise.

Fig. 5 shows a block diagram of our camera pipeline model. The model represents the different steps as linear and affine operations, where the parameters are obtained from the pipeline[1]. The matrices $\mathbf{D}_{\text{LS}}, \mathbf{D}_{\text{WB}}$, and $\mathbf{D}_{\text{TM}}$ defined in Fig. 5 are 3×3 diagonal matrices, and $\mathbf{T}_{\text{CC}}$ and $\mathbf{T}_{\text{YUV}}$ are 3×3 color transform matrices. The 3-vector $\overline{\mathbf{b}}$ contains the black-level correction values for the raw RGB data. We approximate lens shade correction by $\mathbf{D}_{\text{LS}} = \gamma\mathbf{I}$, where γ is the average gain across the RGB channels at a given pixel and its value is spatially varying. The tone mapping matrix[2] $\mathbf{D}_{\text{TM}}$ is also varying depending on the channel inputs and the tone mapping function f_{TM}. Our model does not explicitly account for the demosaic step and treats the raw data as if they were in full resolution. We add an extra scaling step to the final YUV noise model to accommodate the effects of demosaicking and other factors. Given this pipeline model, the raw RGB values $\overline{\mathbf{I}}_{\text{RAW}}$ and the YUV values $\overline{\mathbf{I}}_{\text{YUV}}$ are related by

$$\overline{\mathbf{I}}_{\text{RAW}} = \left(\mathbf{T}_{\text{YUV}}\mathbf{T}_{\text{CC}}\mathbf{D}_{\text{TM}}\mathbf{D}_{\text{WB}}\mathbf{D}_{\text{LS}}\right)^{-1}\overline{\mathbf{I}}_{\text{YUV}} + \overline{\mathbf{b}}. \tag{8}$$

Directly applying Eqs. (7) and (8), we can compute the noise variances in the raw data and consequently the noise variances in the YUV data , but the results would depend on $\overline{\mathbf{I}}_{\text{YUV}}$ and the spatial location in a non-trivial manner. To avoid excessive storage and computation, our final YUV noise model decouples the dependence on $\overline{\mathbf{I}}_{\text{YUV}}$ and the spatial location, and approximates the noise standard deviation (std) in the YUV domain as

$$\hat{\sigma}_{\text{YUV}} = \gamma\tilde{\sigma} \tag{9}$$

$$\text{where} \quad \tilde{\sigma} = \mathbf{D}_{\text{NM}}\sqrt{(\tilde{\mathbf{K}}\circ\tilde{\mathbf{K}})\left[g_1(\mathbf{K}^{-1}\overline{\mathbf{I}}_{\text{YUV}} + \overline{\mathbf{b}}) + g_2\right]}, \tag{10}$$

$$\mathbf{K} = \mathbf{T}_{\text{YUV}}\mathbf{T}_{\text{CC}}\mathbf{D}_{\text{TM}}\mathbf{D}_{\text{WB}}, \tag{11}$$

$$\tilde{\mathbf{K}} = \mathbf{T}_{\text{YUV}}\mathbf{T}_{\text{CC}}\tilde{\mathbf{D}}_{\text{TM}}\mathbf{D}_{\text{WB}}, \tag{12}$$

$\tilde{\mathbf{K}}\circ\tilde{\mathbf{K}}$ is the element-wise square of $\tilde{\mathbf{K}}$, the diagonal of $\tilde{\mathbf{D}}_{\text{TM}}$ contains the mapped values of the tone mapping inputs by the derivative of the tone mapping function f'_{TM}, and $\mathbf{D}_{\text{NM}}$ is a diagonal calibration matrix to account for demosaic and other factors. In Eq. (10), $\left[g_1(\mathbf{K}^{-1}\overline{\mathbf{I}}_{\text{YUV}} + \overline{\mathbf{b}}) + g_2\right]$ computes the raw data noise variances as if there were no lens shading. Multiplying by $\tilde{\mathbf{K}}\circ\tilde{\mathbf{K}}$ accounts for the effect of propagating the noise through the pipeline.

We quantize each of the Y,U,V axes into 10 levels and apply Eq. (10) to construct a look-up table to map $\overline{\mathbf{I}}_{\text{YUV}}$ to $\tilde{\sigma}$. When applying the look-up table, we perform Gaussian filtering to smooth out the YUV data, use the look-up table to estimate $\tilde{\sigma}$, and scale $\tilde{\sigma}$ by the lens shading correction γ as in Eq. (9). To speed up computation further, table look up and scaling are performed with the sub-sampled (by 2) image. The noise map is then used in the denoising filter by modifying the geodesic distance $d[n]$ of Eq. (2) to the following:

$$d[n] = 1 + \frac{\sigma_s}{\sigma_r}\sum_{j=1}^{C}\max\left(\left|I'_j[n]\right| - \lambda\hat{\sigma}_j[n], 0\right), \tag{13}$$

where $\hat{\sigma}_j[n]$ is the noise std estimate for the j-th channel of the n-th pixel, and λ is a tuning parameter which we typically set to 1. Fig. 6b shows the Y channel noise map computed from our noise model, for the noisy image shown in Fig. 6a.

[1]This is now possible with the Camera2 API on all devices running Android 5.0 Lollipop or later. See devCam [13] for details

[2]The tone mapping function f_{TM} is monotone increasing and non-linear. We define $g_{\text{TM}}(x) = f_{\text{TM}}(x)/x$ so that the diagonal of $\mathbf{D}_{\text{TM}}$ contains the mapped values of the tone mapping inputs by g_{TM}. This setup is only for simplifying the notation.

(a) Noisy image (b) Estimated noise map (Y)

Fig. 6: (a) Noisy image (Nexus 6P, 12 MP, ISO-1355). (b) Noise map computed from the LUT approach of Eqs. (9)-(12) (darker regions correspond to higher noise std.).

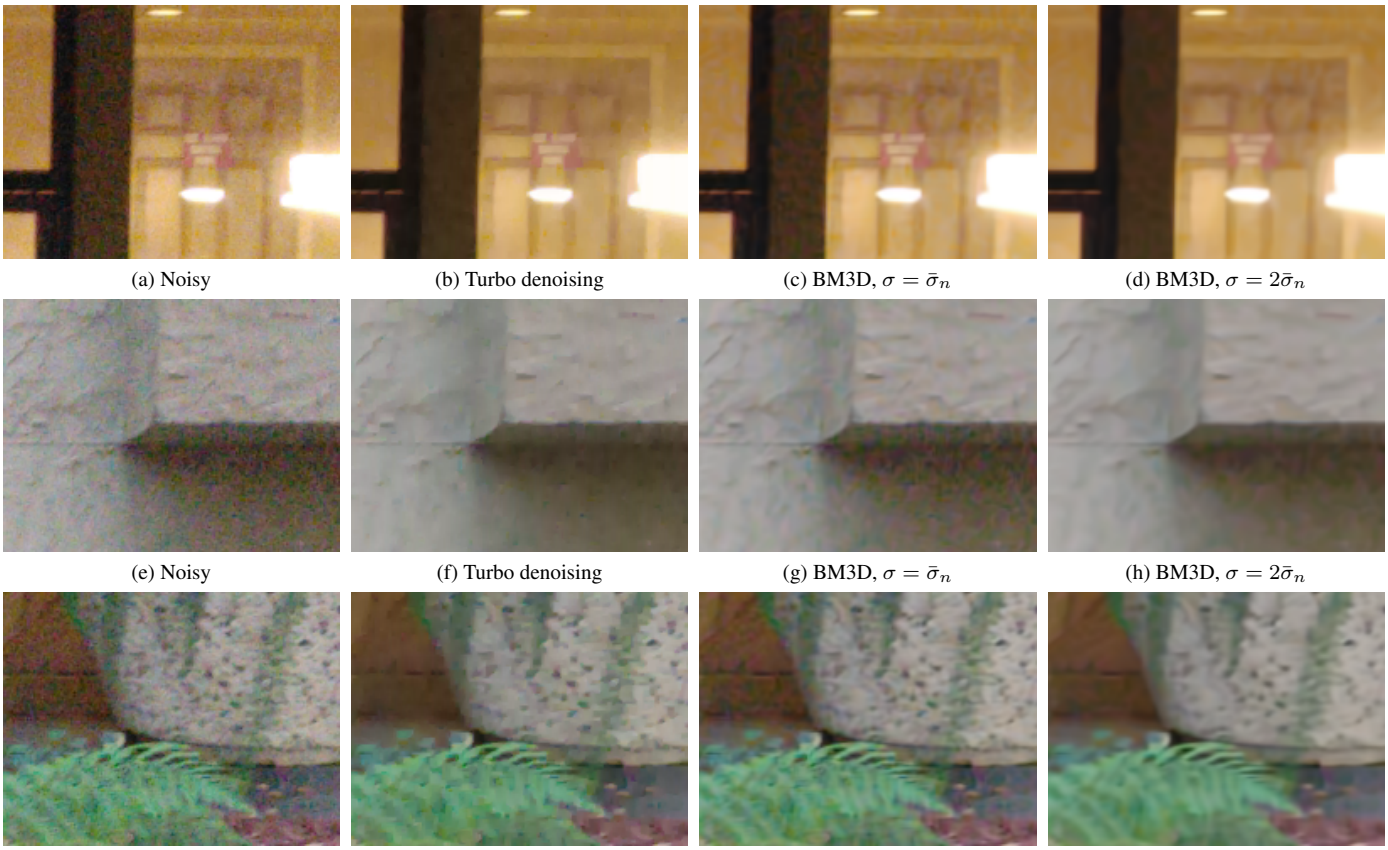
3. EXPERIMENT RESULTS

Fig. 7 shows the results for a few regions from one of our test images (Fig. 6a) captured by Nexus 6P at ISO-1355. The first column shows the noisy regions. Our scheme (column 2) cleans up the noise, even in the darker, more noisy regions, and the sharpness of the textures and edges are preserved properly. For comparison, we show the results of BM3D [1] in column 3 and column 4. We use the BM3D implementation provided by the authors [14], and set its parameter σ to 1× (column 3) and 2× (column 4) the average noise std $\bar{\sigma}_n$ estimated from our noise map.

We also compare our scheme with BM3D numerically in a simulation experiment. Six test images, shown in Fig. 8 were captured by a DSLR camera with a 24 MP full frame sensor at ISO 100-400, and down sized to 1.5 MP to further suppress the noise. For each image, we simulate the inverse of our pipeline model as in Eq. (8) to obtain the sensor data and simulate the sensor noise by Eq. (7) using Nexus 6P parameters at ISO-1355. We then simulate the pipeline model on the sensor data to obtain the ground truth and noisy sRGB images. We compute the PSNR and SSIM [15] values in Table 1, averaged over five noise realizations. The results for Turbo Denoising are shown in column 3 (PSNR) and column 6 (SSIM) in Table 1. For

Table 1: Numerical Comparison, Nexus 6P ISO-1355

	PSNR (dB)			SSIM		
	Turbo Denoising		BM3D	Turbo Denoising		BM3D
	Smoothing	Boosting		Smoothing	Boosting	
Img 1	26.27	**27.16**	27.10	0.73	**0.78**	0.76
Img 2	31.54	**31.87**	31.67	0.87	**0.88**	0.85
Img 3	29.39	30.33	**31.98**	0.89	0.90	**0.92**
Img 4	28.23	29.16	**29.27**	0.85	**0.87**	**0.87**
Img 5	30.70	**31.25**	**31.25**	0.89	**0.90**	0.89
Img 6	26.49	**27.97**	**27.97**	0.82	**0.87**	0.86



(a) Noisy (b) Turbo denoising (c) BM3D, $\sigma = \bar{\sigma}_n$ (d) BM3D, $\sigma = 2\bar{\sigma}_n$

(e) Noisy (f) Turbo denoising (g) BM3D, $\sigma = \bar{\sigma}_n$ (h) BM3D, $\sigma = 2\bar{\sigma}_n$

(i) Noisy (j) Turbo denoising (k) BM3D, $\sigma = \bar{\sigma}_n$ (l) BM3D, $\sigma = 2\bar{\sigma}_n$

Fig. 7: (a),(e),(i) Noisy regions, Nexus 6P, ISO-1355; (b),(f),(j) Turbo denoising; (c),(g),(k) BM3D, $\sigma = \bar{\sigma}_n$; (d),(h),(l) BM3D, $\sigma = 2\bar{\sigma}_n$.

(a) Img 1 (b) Img 2 (c) Img 3

(d) Img 4 (e) Img 5 (f) Img 6

Fig. 8: Thumbnails of the test images of Table 1.

all images, we fix the parameters of both IIR-DT filters at $\sigma_s = 2.5$ and $\sigma_r = 6.5$ for simplicity. The results for BM3D are shown in column 4 (PSNR) and column 7 (SSIM) of Table 1. For each image, we compute the mean absolute error of the noisy image $\bar{\sigma}_n$ and set the parameter of BM3D to $\sigma = \tau\bar{\sigma}_n$, with τ varying over the range 1 to 3 with step of 0.1, to give BM3D the best chance of success. The best PSNR and SSIM values are then reported in column 4 and column 7 of Tables 1. In terms of PSNR and SSIM, the results of Turbo Denoising are close to those of BM3D. In a few occasions, Turbo Denoising even achieves higher PSNR or SSIM values. We believe one can further improve the BM3D results by incorporating a noise map in its operation, but this will also add more computation to an already computationally intensive algorithm. Further, the proper way to implement it is not immediately clear.

We implement our scheme in C++ with Halide [16]. We benchmark our scheme on a Intel Xeon E5 (6 cores, 3.5GHz) Linux computer with 32GB of memory. The single-threaded run time of Turbo Denoising on color images, excluding noise look up, is 133.64 msec/MP. The single-threaded run time for noise look up is 13.77 msec/MP. The multi-threaded run times of Turbo Denoising and noise look up are 32.77 msec/MP and 5.68 msec/MP respectively. The BM3D implementation [14] consists of compiled Matlab code executed by a front-end Matlab function. The implementation details and language used are unknown to us. Its run time is 15.23 sec/MP, which is about 103 times slower and 396 times slower than our single-threaded and multi-threaded implementations, respectively. All reported run times are averaged over 10 executions.

4. CONCLUSION

We presented a new denoising algorithm based on a simple two step processing. We also proposed a novel YUV noise model to estimate the complex intensity and location dependent image noise. Both the denoising algorithm and the noise model can be implemented very efficiently. Our scheme demonstrates denoising results matching those of BM3D, but requires significantly less computation.

5. REFERENCES

[1] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *Image Processing, IEEE Transactions on*, vol. 16, no. 8, pp. 2080–2095, Aug 2007.

[2] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, June 2005, vol. 2, pp. 60–65 vol. 2.

[3] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *Image Processing, IEEE Transactions on*, vol. 15, no. 12, pp. 3736–3745, Dec 2006.

[4] P. Chatterjee and P. Milanfar, "Patch-based near-optimal image denoising," *Image Processing, IEEE Transactions on*, vol. 21, no. 4, pp. 1635–1649, April 2012.

[5] H. Talebi, X. Zhu, and P. Milanfar, "How to saif-ly boost denoising performance," *Trans. Img. Proc.*, vol. 22, no. 4, pp. 1470–1485, Apr. 2013.

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

[7] A. Adams, J. Baek, and M. A. Davis, "Fast high-dimensional filtering using the permutohedral lattice," in *Computer Graphics Forum*, 2010, vol. 29, pp. 753–762.

[8] Eduardo S. L. Gastal and Manuel M. Oliveira, "Domain transform for edge-aware image and video processing," in *ACM SIGGRAPH 2011 Papers*, New York, NY, USA, 2011, SIGGRAPH '11, pp. 69:1–69:12, ACM.

[9] J.W. Tukey, *Exploratory Data Analysis*, Addison-Wesley series in behavioral science. Addison-Wesley Publishing Company, 1977.

[10] P. Milanfar, "A tour of modern image filtering: New insights and methods, both practical and theoretical," *Signal Processing Magazine, IEEE*, vol. 30, no. 1, pp. 106–128, Jan 2013.

[11] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *Proceedings of the Third European Conference on Computer Vision (Vol. II)*, Secaucus, NJ, USA, 1994, ECCV '94, pp. 151–158, Springer-Verlag New York, Inc.

[12] Y. Tsin, V. Ramesh, and T. Kanade, "Statistical calibration of CCD imaging process," in *Proceedings IEEE International Conference on Computer Vision*, 2001, vol. 1, pp. 480–487.

[13] R. Sumner, "devCam – parameterized image capture for algorithm development and testing," `http://devcamera.org/`.

[14] K. Dabov, Danieyan A., and A. Foi, "Bm3d demo software for image/video restoration and enhancement," `http://www.cs.tut.fi/~foi/GCF-BM3D/BM3D.zip`, 2006–2014.

[15] Z. Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *Image Processing, IEEE Transactions on*, vol. 13, no. 4, pp. 600–612, April 2004.

[16] J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. Amarasinghe, and F. Durand, "Decoupling algorithms from schedules for easy optimization of image processing pipelines," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 32:1–32:12, July 2012.