A PULL-PUSH METHOD FOR FAST NON-LOCAL MEANS FILTERING

John R. Isidoro, Peyman Milanfar

Google Research
isidoro@google.com, milanfar@google.com

ABSTRACT

Non-local means filtering (NLM), has garnered a large amount of interest in the image processing community due to its capability to exploit image patch self-similarity in order to effectively filter noisy images. However, the computational complexity of non-local means filtering is the product of three different factors; namely, $O(NDK)$, where K is the number of filter kernel taps (e.g. search window size), D is the number of patch taps, and N is number of pixels.

We propose a fast approximation of non-local means filtering using the multiscale methodology of the pull-push scattered data interpolation method. By using NLM with a small filter kernel to selectively propagate filtering results and noise variance estimates from fine to coarse scales and back, the process can be used to provide comparable filtering capability to brute force NLM but with algorithmic complexity that is linear in the number of image pixels and the patch comparison taps, $O(ND)$. In practical application, we demonstrate its denoising capability is comparable to NLM with much larger filter kernels, but at a fraction of the computational cost.

Index Terms— Non-local means, multiscale, pull-push, image denoising

1. INTRODUCTION

A classic way to improve the performance of a filtering algorithm is to use a multiscale technique in order to compute a wide filtering kernel by using smaller filters on a variety of scales. Using a coarse to fine strategy allows for a small amount of processing at each successive level to refine the solution.

Most multiscale approaches use a fine to coarse to fine approach and use two pyramids, resulting in 1.66 times the complexity of the direct approach. However, multiscale approaches allow the use of much simpler and smaller per-level filters, especially when a single scale filter being approximated is large and not separable.

There are two important contributions we present in this work. First, although multiscale techniques have been used for decades, many of these techniques use a fixed (non-data dependent) kernel in order to generate successively downsampled versions of the original image. Since these techniques do not alter the filtering kernel based on the data, they may decompose the data in a way that is not compatible with data dependent filtering, i.e. filtering across edges and propagating erroneous results through the solution. By contrast, instead of a fixed linear filter used in standard multiscale models, in our approach a nonlinear filter (NLM in this case) is used to construct the multiscale representation.

Second, the original pull-push approach [1] was designed for fast scattered data interpolation. Here, we extend its domain of use to a data dependent multiscale algorithm for filtering images effectively and efficiently. More specifically, we expand upon the ideas of pull-push to accelerate non-local means in a way that provides competitive results to large kernel (e.g. large search window) NLM filters, but requires a computation amount comparable to a much smaller kernel.

2. RELATED WORK

Non-local means [2] is a filtering technique that generalizes bilateral filtering [3, 4] by using a patch matching score in order to derive filtering weights. This patch matching allows for a better characterization of image pixel distributions in a way that allows for a higher degree of edge aware denoising than bilateral filtering.

One class of methods for accelerating edge aware filtering involves lifting the filtering space into a higher dimensional space in which fast non-data-dependent separable filtering can be applied to perform the bulk of the work [5, 6]. This can be used to decouple the computational complexity from the filter kernel size (e.g. search window size) K. As an example, the permutohedral lattice [7] is capable of NLM filtering, but the performance scales quadratically with the number of comparison patch samples D. i.e. $O(ND^2)$.

Other methods for fast edge-aware filtering involve approximating the bilateral filter as a series of 1D separable [15] or recursive [16, 17] passes. These approaches have not been extended to NLM.

Multiscale techniques have been used to accelerate NLM in the original paper [2], but were limited to optimizing the patch comparisons or weight computation [8].

[9] uses Laplacian pyramids and performs NLM filtering in each level as part of the reconstruction of the original image from the pyramid decomposition. [10] is conceptually similar but uses a wavelet decomposition. [11] proposes an alternative fixed kernel pyramid decomposition with bilateral filtering at each level. [12] also uses multiple image scales generated with non-edge-aware filtering, but uses NLM to directly fuse these pixels back into the original level. Other edge preserving multiscale decompositions exist throughout the literature, such as [13] and [14], but have not been extended to NLM filtering.

The main difference between our multiscale approach and these approaches is that by using NLM as the filter in the construction of the up and down pyramid processing, we avoid combining pixels together that would not be combined under the brute force NLM approach.

[18] performs successive iterations of NLM filtering and updates per pixel variance values based on the estimated variance reduction from the sum of squared normalized weights. Much of the variance propagation machinery is similar in our approach, but their approach is not multiscale and thus processes the full resolution image multiple times. Also, in [18] per-iteration the filter size grows arithmetically instead of geometrically per-level as multiscale approaches do.

A common optimization for NLM is to reorder the loops in the algortihm and use separable filtering for the weight computation [19, 20]. This technique can also be applied to the pull-push algorithm presented in this paper as an additional optimization.

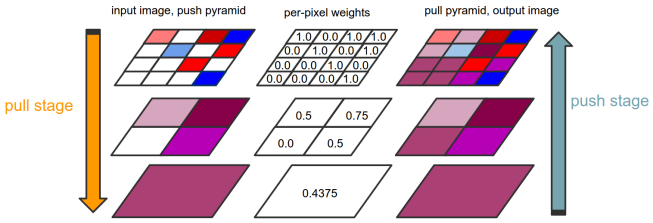

Fig. 1. Diagram of how the pull-push algorithm keeps track of per-pixel weights to represent the pixel coverage each coarser level has. These weights are used to combine samples in successively coarser or finer levels.

2.1. Overview of Pull-Push

The pull-push algorithm was originally designed as an efficient, GPU amenable, scattered data interpolation method for Lumigraph rendering [1]. The basic problem addressed was to fill in the missing pixels in an image where only a subset of the pixels are specified. Figure 1 shows the data flow for pull-push. For the initial finest level, a weight of 1 is used for pixels that are present and a weight of 0 for pixels that are missing.

The first step in pull-push is to build an image pyramid. Each successive level is generated recursively by a 'pull' (analogous to downsampling) filter. This filter uses normalized per-pixel weighting, to selectively incorporate present pixels. In addition to this, for each pixel generated, a weight is computed which is proportional to the number of present pixels used to generate that pixel. This weight can be thought of as a coverage fraction.

After the pull stage is evaluated to the coarsest desired level, the push stage blends in missing pixels starting from the coarsest level and blending in finer levels iteratively using the computed weights.

A few key concepts about pull-push should be noted here. Like other multiscale frameworks such as Laplacian pyramids, pull-push still maintains the capability to much more efficiently filter over large regions than a single scale method. Because of this, and since a small fixed size kernel is used per level, the overall complexity of the pull-push approach is not dependent on the effective filter size.

Each pull-stage down-fused pixel can be considered as a distribution of the pixel values in the finer levels above it, where the pixel value functions as a mean, and the weight function as an encoding of the strength or reliability of the mean (e.g. how many pixels were combined to represent it). Regions which have less reliable, or no information incorporate more data from the wider region around them in order to form an estimate. Most crucially, this selective filtering and reliability (i.e. information) propagation is what makes push-pull different than existing multiscale frameworks.

2.2. Overview of NLM

Non-local means [2] is a filtering technique that uses a patch matching score in order to derive filtering weights. Given an image x to filter, the kernel weight $w_{i,k}$ per-pixel location i and tap location k is based on a patch neighborhood with patch taps p. Consider σ_s to be a smoothing parameter based on the amount of sensor noise. The weights are designed as follows

$$\Delta_{i,k} = \sum_p \left(\frac{x_{i+p} - x_{i+k+p}}{\sigma_s} \right)^2 \tag{1}$$

$$w_{i,k} = \exp(-\Delta_{i,k}) \tag{2}$$

Normalizing by the sum of weights:

$$\widehat{w}_{i,k} = \frac{w_{i,k}}{\sum_k w_{i,k}} \tag{3}$$

Each new filtered pixel is a weighted average of its kernel neighborhood:

$$\widehat{x}_i = \sum_k \widehat{w}_{i,k}\ x_{i+k}. \tag{4}$$

Patch matching comes at additional computational expense. The additional work in the per-pixel inner loop over the patch taps is multiplicative and increases the computational complexity of the algorithm from the $O(NK)$ of a brute force bilateral implementation to $O(NDK)$, where K is the number of filter kernel taps, D is the number of patch samples, and N is number of pixels. Note that in the original NLM paper [2], K included all pixels in the image, but in practice this is often limited to a relatively small local window with a fixed number of kernel taps. By combining NLM with a pull-push approach we show how the algorithmic complexity can be reduced to $O(ND)$.

3. PULL-PUSH NLM

Using the standard NLM formulation with patch matching based on a Gaussian noise model, on a per-pixel basis, a variable number of pixels are averaged together based on their patch similarities. Considering the sensor noise for a pixel as uncorrelated Gaussian with mean μ_i and standard deviation σ_n, the estimated distribution of the filtered sample (a weighted average of Gaussian samples) can be computed from the sum of the squared normalized kernel weights [21]:

$$\widehat{x}_i \sim \mathcal{N}\left(\sum_k \widehat{w}_{i,k}\ x_{i+k}\ ,\ \sigma_n^2 \sum_k \widehat{w}_{i,k}^2 \right) \tag{5}$$

Considering the variance reduction from Eq. 5, the NLM kernel weights can be used to determine of the reliability of a filtered sample. By outputting the sum of squared normalized weights along with the NLM filtered pixel values, we are in effect providing a simple characterization of a neighborhood distribution of samples used to create the sample. By propagating this information through the layers, we enable tracking a simplified model of the neighborhood statistics for successively larger regions in a multiscale manner.

One last point is that eventually we will need to combine estimates with different variances during the pull stage. Inverse variance weighting is a common method to aggregate multiple independent observations y_k each with different variances $\sigma_{y_k}^2$. Given this input, inverse variance weighting is known to be the minimum variance estimate using all the uncorrelated observations [22][23]. This takes the form:

$$\widehat{y} = \frac{\sum_k \frac{y_k}{\sigma_{y_k}^2}}{\sum_k \frac{1}{\sigma_{y_k}^2}} \tag{6}$$

In Subsection 3.3, we show how inverse variance weighting is used to derive our reliability score.

3.1. Pull stage: Downfuse

In order to aggregate filtering results for a neighborhood of pixels without filtering across edges NLM filtering is used to generate subsequent coarser pyramid levels during the pull stage. Because it is not performing a scaling operation but rather performing a selective

combination of pixels, we call this a downfuse pass instead of downscaling. A basic approach is to perform NLM every other pixel in level $[r]$, and store the results in the next coarsest level $[r+1]$. Superscripts are used to denote pyramid level.

$$\Delta_{i,k}^{[r]} = \sum_p \left(\frac{x_{i+p}^{[r]} - x_{i+k+p}^{[r]}}{\sigma_s^{[r]}} \right)^2 \tag{7}$$

$$w_{i,k}^{[r]} = \exp(-\Delta_{i,k}^{[r]}) \tag{8}$$

$$\widehat{w}_{i,k}^{[r]} = \frac{w_{i,k}^{[r]}}{\sum_k w_{i,k}^{[r]}} \tag{9}$$

$$x_i^{[r]} = \sum_k \widehat{w}_{2i,k}^{[r-1]} \; x_{2i+k}^{[r-1]} \tag{10}$$

Different samples will fuse together differing numbers of samples. However, since the power of the pull-push approach relies on propagating reliability information in order to selectively combine data from level to level we also compute reliability per-pixel for each downfused sample. One statistically motivated way to do this is the following:

$$\rho_i^{[r]} = \frac{1}{\sum_k (\widehat{w}_{2i,k}^{[r-1]})^2} \tag{11}$$

This reliability measure is the inverse of the variance scale factor for weighted sum of random variables from the previous subsection. This reliability is analogous to the way weights are propagated in the pull-push technique. Incorporating the reliability score into NLM can be performed as weighted NLM for $r \geq 1$:

$$w_{i,k}^{[r]} = \rho_{i,k}^{[r-1]} \exp(-\Delta_{i,k}^{[r]}) \tag{12}$$

3.2. Push stage: Upfuse

Now that a pyramid has been constructed containing filtering results, each with a reliability measure, a push stage will be run in order to up-integrate the results. Just as in the pull stage, the push stage also uses NLM, but also incorporates samples from lower pyramid levels. Analogous to downfuse we call the push stage the upfuse stage because of the selective filtering and upintegration of samples.

However, in order to determine the patch comparison weights to incorporate samples from a coarser level, we perform patch comparisons from the level upsampled into. Note that during the downfuse stage each NLM result sample was generated from one level finer, including the patch comparisons to generate that sample. Therefore, it makes sense that a center tap patch used by the downfuse stage to generate a coarser sample serves as a signature for the neighborhood, and can be used for patch comparison to determine weights for the coarser sample. For upfusing, the tap offsets k come from two different locations. Tap offsets from the current level are denoted as k_f, and the tap offsets for samples from the coarser level will be denoted as k_c. The set of taps containing both k_f and k_c is k.

The weights for sample offsets for the different levels are:

$$w_{i,k_f}^{[r]} = \exp(-\Delta_{i,k_f}^{[r]}) \tag{13}$$

$$w_{i,k_c}^{[r]} = \rho_{g(i+k_c)/2}^{[r+1]} \exp(-\Delta_{i,g(k_c)}^{[r]}) \tag{14}$$

where $g(j)$ is the tap location in the finer level that generated the downfused sample in the coarser level for location j.

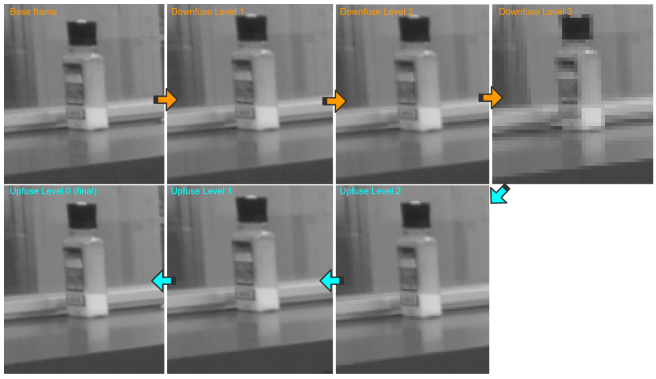

Fig. 2. This series of images shows a zoomed in portion of an image and how filtering progresses through the downfuse and upfuse process.

Again the weights are normalized using the sum over all weights (e.g. from both the coarse and fine levels),

$$\widehat{w}_{i,k}^{[r]} = \frac{w_{i,k}^{[r]}}{\sum_k w_{i,k}^{[r]}} \tag{15}$$

$$y_i^{[r]} = (\sum_{k_f} \widehat{w}_{i,k_f}^{[r]} \; x_{i+k_f}^{[r]}) + \sum_{k_c} \widehat{w}_{i,k_c}^{[r]} \; y_{g(i+k_c)/2}^{[r+1]} \tag{16}$$

with the exception of the coarsest level which is just a direct copy of the coarsest downfused level. Figure 2 shows a zoomed in portion of an image being filtered and how filtering progresses through the downfuse and upfuse process.

3.3. Selective Up-Integration using Reliability

During the upfuse process it is important to prevent incorporating unreliable data from lower pyramid levels. Due to local image structure, filtered results may not be reliable or useful if too few samples were combined to create the result. Furthermore, but if few samples were combined, it signifies the region may contain some form of local structure, and directly using the coarser results may introduce undersampling artifacts. In order to prevent this, we only combine samples for which the reliability score is above a given threshold. In practice, we found a threshold corresponding to least 3 samples being combined $\rho_{thresh} = 3$ is effective to eliminate artifacts. Figure 3 shows how reliability downweighting alleviates artifacts. In order to prevent artifacts from a sharp thresholding and to provide a smooth transition based on the reliability, we subtract and clamp in order to implement this threshold.

$$\widehat{\rho}_i = min(\rho_i - \rho_{thresh}, 0) \tag{17}$$

4. EXPERIMENTAL RESULTS

The experiments were run on a 6 Core Xeon CPU running at 2.4GHz using Halide [24] for vectorization and parallelization support. The implementation was run on 4032x3024 YUV imagery with 8-bit full resolution luma and half resolution chroma. In order to best compare algorithm performance, both the pull-push NLM and standard NLM implementations use floating point math and the same 3x3 patch comparison size. Other than using Halide vectorization and parallelization and an optimizing compiler, no other optimizations were used for this comparison.

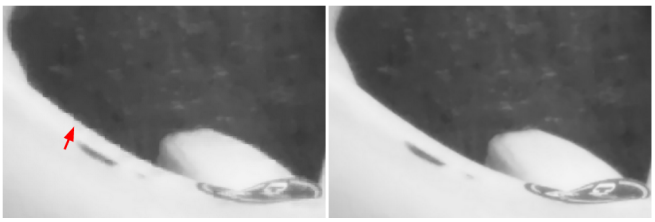

Fig. 3. The left image shows the effect of pull-push filtering without reliability downweighting. Near the edge of the banana, the upfused samples contained a high amount of structure and resulted in aliasing artifacts. The right side image shows the same region filtered using the reliability adjustment to mitigate the artifacts.

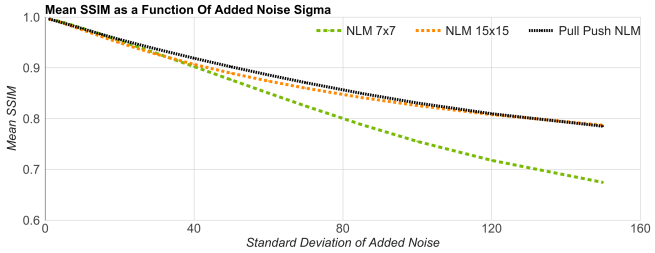

Fig. 4. Graph comparing denoising performance of pull-push NLM vs standard NLM for a range of synthetic noise levels.

We compared the algorithms using a suite of 143 different clean real world images by adding synthetic Gaussian noise, and performing filtering. For comparison we use SSIM [25] to measure the difference between the noiseless base image, and the processed denoised image. The standard recommended values from [25] were used for all parameters. For each algorithm, the input sigma parameters resulting in the highest SSIM score were used on a per-image basis.

We found that the pull-push NLM approach runs in a little less time than NLM using a 7x7 filter kernel (e.g. search window), (320ms vs 350ms per megapixel). Comparison with the 7x7 NLM can be considered as a quality comparison for equivalent runtime. In order to show an output quality comparison of pull-push NLM vs large kernel NLM we compare against NLM with a 15x15 filter kernel (1660ms per megapixel). Note that all of these timings can be improved by an order of magnitude with standard NLM optimizations such as those described in [26].

The results are summarized in Figure 4. Trials were run on a wide range of synthetic noise levels ranging in standard deviation from mild noise (σ = 1) to heavy noise (σ = 150). Each data point on the graph represents the mean best-case SSIM for the given algorithm across the 143 images. For noise $\sigma \leq 15$, the three approaches resulted in very similar SSIM scores. However as σ increased, the pull-push NLM approach performed better than 7x7 NLM with the difference widening as noise increased. Even against NLM using a 15x15 kernel (and taking 5 times as much time to process the same image) the pull-push approach provided better quality results from σ ranging from 20 to 125.

We found the pull push NLM algorithm performed optimally using 5 pyramid levels. Additional pyramid levels were comparatively free, but offered little additional denoising, due to the fact that a 5 level pyramid offers up to a 31 pixel filter radius (63x63) and can potentially combine thousands of samples. Experimentally we found that halving the patch comparison sigma for each level down the pyramid gave the best results.

Figures 5, 6, 7, and 8 show a visual comparison of the results and the associated SSIM scores.

Fig. 5. From left to right, image with additive Gaussian noise with σ=85 (SSIM:0.065), 7x7 NLM (SSIM:0.799), 15x15 NLM (SSIM:0.918), pull-push NLM (SSIM:0.935). It is recommended to view these figures zoomed in on a high resolution screen.

Fig. 6. From left to right, image with additive Gaussian noise with σ=85 (SSIM:0.078), 7x7 NLM (SSIM:0.748), 15x15 NLM (SSIM:0.833), pull-push NLM (SSIM:0.840)

Fig. 7. From left to right, synthetic image with text with additive Gaussian noise with σ=85 (SSIM:0.089), 7x7 NLM (SSIM:0.794), 15x15 NLM (SSIM:0.864), pull-push NLM (SSIM:0.874)

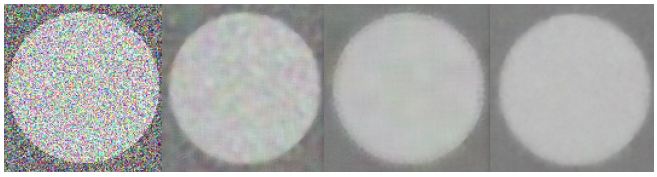

Fig. 8. From left to right, synthetic image of a solid circle with additive Gaussian noise with σ=60 (SSIM:0.082), 7x7 NLM (SSIM:0.834), 15x15 NLM (SSIM:0.945), pull-push NLM (SSIM:0.973)

5. CONCLUSION

We have presented a framework to efficiently compute an approximation to NLM filtering with large spatial kernels. In practice we are able to achieve much better filtering results than the brute force NLM approach taking roughly the same amount of time.

One minor limitation of the proposed algorithm is that repetitive patterns within an image may not receive as much filtering as in a brute force NLM algorithm with a large kernel size. However, pull-push achieves better noise reduction in smooth regions which is where the sensor noise tends to be the most noticeable by human eyes.

6. REFERENCES

[1] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The lumigraph," in *In Proceedings of SIGGRAPH 96*. 1996, pp. 43–54, ACM.

[2] A. Buades, B Coll, and J. M. Morel, "A review of image denoising algorithms, with a new one," *Multiscale Modeling and Simulation (SIAM interdisciplinary journal)*, vol. 4, no. 2, pp. 490–530, 2005.

[3] S. M. Smith and J. M. Brady, "Susan – a new approach to low level image processing," *International Journal of Computer Vision*, vol. 23, no. 1, pp. 45–78, 1997.

[4] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," *ICCV, Bombay, India*, pp. 836–846, January 1998.

[5] J. Chen, S. Paris, and F. Durand, "Real-time edge-aware image processing with the bilateral grid," *ACM Trans. Graph.*, vol. 26, no. 3, 2007.

[6] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," *Int. J. Comput. Vision*, vol. 81, no. 1, pp. 24–52, Jan. 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] X. Liu, X. Feng, and Y. Han, "Multiscale nonlocal means for image denoising," in *Wavelet Analysis and Pattern Recognition (ICWAPR), 2013 International Conference on*, 2013.

[9] S. Nercessian, K. A. Panetta, and S. S. Agaian, "A multi-scale non-local means algorithm for image de-noising," *Proc. SPIE*, vol. 8406, pp. 84060J–84060J–10, 2012.

[10] M. Zhang and B. K. Gunturk, "Multiresolution bilateral filtering for image denoising," *IEEE Trans. On Image Processing*, vol. 17, no. 12, pp. 2324–2333, Dec 2008.

[11] Q. She, Z.Lu, W. Li, and Q. Liao, "Multigrid bilateral filtering," in *IEICE Trans. Inf. Syst.*, 2014, pp. 2748–2759.

[12] M. Zontak, I. Mosseri, and M. Irani, "Separating signal from noise using patch recurrence across scales," *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 1195–1202, 2013.

[13] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski, "Edge-preserving decompositions for multi-scale tone and detail manipulation," *ACM Trans. Graph.*, vol. 27, no. 3, August 2008.

[14] R. Fattal, "Edge-avoiding wavelets and their applications," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 1–10, 2009.

[15] T. Q. Pham and L. J. Vliet, "Separable bilateral filtering for fast video preprocessing," in *In IEEE Internat. Conf. on Multimedia & Expo, CD14*. 2005, pp. 1–4, IEEE.

[16] E. S. L. Gastal and M. M. Oliveira, "Domain transform for edge-aware image and video processing," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 69:1–69:12, July 2011.

[17] Q. Yang, "Recursive bilateral filtering," in *ECCV*, 2012, pp. 399–413.

[18] C. Kervrann and J. Boulanger, "Local adaptivity to variable smoothness for exemplar-based image regularization and representation," *International Journal of Computer Vision*, vol. 79, no. 1, pp. 45–69, 2008.

[19] J. Darbon, A. Cunha, T. F. Chan, S. Osher, and G. J. Jensen, "Fast nonlocal filtering applied to electron cryomicroscopy.," in *International Symposium on Biomedical Imaging (ISBI)*. 2008, pp. 1331–1334, IEEE.

[20] L. Condat, "A simple trick to speed up and improve the non-local means," Research Report hal-00512801, Aug. 2010.

[21] P. Milanfar, "A tour of modern image filtering," *IEEE Signal Processing Magazine*, vol. 30, no. 1, pp. 106–128, 2013.

[22] S. M. Kay, *Fundamentals of Statistical Signal Processing - Estimation Theory -*, Signal Processing Series. PTR Prentice-Hall, Englewood Cliffs, N.J., 1993.

[23] J. Hartung, G. Knapp, and B. K. Sinha, *Statistical meta-analysis with applications*, John Wiley and Sons. ISBN 978-0-470-29089-7, 2008.

[24] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," *SIGPLAN Not.*, vol. 48, no. 6, pp. 519–530, June 2013.

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

[26] M. Protter, M. Elad, H. Takeda, and P. Milanfar, "Generalizing the non-local-means to super-resolution reconstruction," in *IEEE Trans. On Image Processing*, 2009, p. 36.