IMAGE PROCESSING ON LINE

Published in Image Processing On Line on 2021–11–22.
Submitted on 2021–03–15, accepted on 2021–10–20.
ISSN 2105–1232 © 2021 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2021.340

Implementation of VBM3D and Some Variants

Thibaud Ehret, Pablo Arias

Université Paris-Saclay, CNRS, ENS Paris-Saclay, Centre Borelli, France
thibaud.ehret@ens-cachan.fr

Communicated by Julie Delon   *Demo edited by* Thibaud Ehret

Abstract

VBM3D is an extension to video of the well-known image denoising algorithm BM3D, which takes advantage of the sparse representation of stacks of similar patches in a transform domain. The extension is rather straightforward: the similar 2D patches are taken from a spatio-temporal neighborhood which includes neighboring frames. In spite of its simplicity, the algorithm offers a good trade-off between denoising performance and computational complexity. In this work we revisit this method, providing an open-source C++ implementation reproducing the results. A detailed description is given and the choice of parameters is thoroughly discussed. Furthermore, we study and compare several extensions of the original algorithm: (1) a multiscale implementation, (2) the use of 3D patches, (3) the use of optical flow to guide the patch search. These extensions yield results which are competitive with the most recent state of the art.

Source Code

The reviewed source code and documentation for this algorithm are available from the web page of this article[1]. Compilation and usage instruction are included in the README.txt file of the archive.

Keywords: video; denoising; sparsity

1 Introduction

VBM3D was proposed by [7] as an adaptation to video denoising of BM3D, a patch-based image denoising algorithm [9]. The algorithm is designed for additive white Gaussian noise (AWGN) with zero mean and standard deviation σ, i.e.

$$v(\rho) = u(\rho) + n(\rho), \quad n(\rho) \sim \mathcal{N}(0, \sigma^2),$$

where ρ is a spatio-temporal position in the video domain (a pixel), v is the noisy video and u the unknown clean video. The denoising principle of VBM3D (and BM3D) is based on the redundancy of similar patches. Groups of similar 2D patches are assembled in a 3D stack of patches. A separable 3D transform is applied to this stack. The stack is denoised by applying a shrinkage operator to the coefficients in the transformed domain. The algorithm follows four basic steps:

[1] https://doi.org/10.5201/ipol.2021.340

Thibaud Ehret, Pablo Arias, *Implementation of VBM3D and Some Variants*, Image Processing On Line, 11 (2021), pp. 374–395. https://doi.org/10.5201/ipol.2021.340

1. search for similar patches in the sequence, grouping them in 3D stacks,
2. apply a 3D linear domain transform to the 3D block,
3. shrink the transformed coefficients,
4. apply the inverse transform,
5. aggregate the resulting patches in the video.

The underlying idea is that, due to the high redundancy of a stack of similar patches, the energy will be concentrated in a few coefficients after the transform, while the noise is spread evenly among all coefficients. This allows to jointly denoise the patches of each stack. The reconstruction of the estimated video is obtained by aggregating for each pixel the estimated patches that contain it. This principle is applied twice. The first time the patches are denoised using a hard threshold in the transformed domain. In the second iteration a Wiener filter is used, with Wiener coefficients computed using the output of the first step as oracle.

There exist other adaptations to video and 3D images of this framework: BM4D [16] and VBM4D [15]. These methods stack similar 3D spatio-temporal patches in a 4D stack. The main difference between them is that VBM4D uses motion-compensated patches whereas BM4D aims at denoising volumetric images (such as those appearing in medical imaging), where the 3rd dimension is just another spatial dimension. In [1] however, it was shown that even non-motion-compensated 3D patches provide a very good denoising performance.

In this article, we revisit the VBM3D method, providing an open source implementation and we discuss some variants introduced in [1], such as 3D patches (as in BM4D [16]), optical flow guided patch search, and a multiscale implementation. In the next section, the algorithm itself is reviewed. In the following sections we consider three possible extensions: multiscale, spatio-temporal patches and motion-compensated patch search using an optical flow. Finally the performance of the algorithm is compared against other state of the art algorithms.

2 Description of the VBM3D Algorithm

VBM3D performs two steps, like its image counterpart BM3D. The first *hard-thresholding* step computes a basic estimate which is refined in the *Wiener filtering* step, yielding the final estimate. Figure 1 represents the global structure of the algorithm. The pseudocode of the core of the algorithm is presented in Algorithm 1.

Notation. For the parameters of the method we will use mostly the same notation as in [7]. The different parameters for the patch search are the number N of patches to return, the number of temporal frames N_f that are being searched, the search region for the reference frame N_s, the search region for the other frames N_{pr}, the maximum number N_b of patches to compute for each local search, a correcting factor d and a maximum distance threshold τ. The threshold parameter in the first step is λ_{3D}. We also write $\otimes$ for the element-wise product. Similar parameters will be used for the hard thresholding first step and for the Wiener filtering second step. Because they can have different values for the different steps, they'll be differentiated using the subscript *hard* and *wien* respectively.

2.1 The Patch Search

The groups of similar patches are built by selecting a reference patch and searching around it for its nearest neighbors. This patch search is the main difference between BM3D and VBM3D. The image

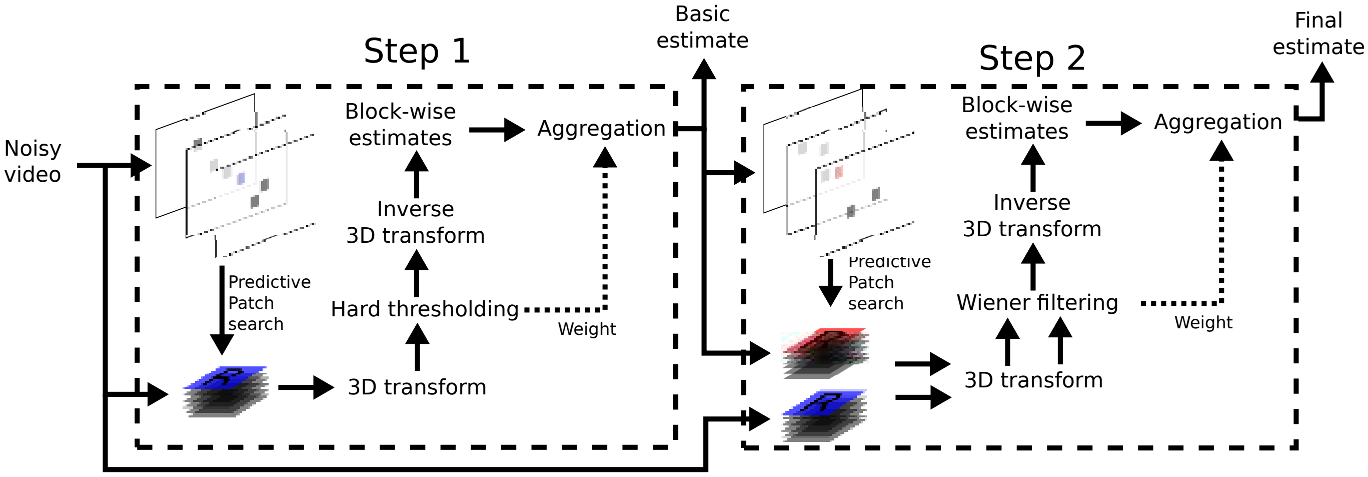

Figure 1: Scheme of the core of the VBM3D algorithm.

version of the algorithm searches a square 2D window centered at the reference patch. While the same could be done in a space-time volume for videos, Dabov et al. [7] propose a *predictive search* heuristic to reduce the size of the search window. The idea behind the proposed search is to track patches in the video. Let $\boldsymbol{p}$ and $\boldsymbol{q}$ be two patches at positions (x, y, t) and (x', y', t') respectively. The distance between $\boldsymbol{p}$ and $\boldsymbol{q}$ used to find the nearest neighbors is defined as follows

$$d(\boldsymbol{p}, \boldsymbol{q}) = \begin{cases} \|\boldsymbol{p} - \boldsymbol{q}\|_2^2 - \delta \text{ if } x = x' \text{ and } y = y' \\ \|\boldsymbol{p} - \boldsymbol{q}\|_2^2 \text{ otherwise} \end{cases}, \tag{1}$$

where δ is a positive parameter which biases the patch trajectories by assuming that the patch does not move.

Suppose that the reference patch is $\boldsymbol{p}$ located at (x, y, t). The goal of the proposed search is to find (at most) N patches in a window of $2N_f + 1$ frames around frame t. The steps are the following:

1. Find the N_b nearest neighbors to $\boldsymbol{p}$ in frame t in a square search region of size $N_s \times N_s$ centered at the location of the reference patch (x, y, t). Let L_t be the set of found patches.

2. Find the N_b nearest neighbors to $\boldsymbol{p}$ in frames $t' = t + 1, \dots, t + N_f$. The search region at t' is the union of $N_{pr} \times N_{pr}$ square regions centered at the positions of the candidates in $L_{t'-1}$, where $N_{pr} < N_s$. This is depicted in Figure 2.

3. Similarly, find N_b nearest neighbors to $\boldsymbol{p}$ in frames $t' = t - 1, t - 2, \dots, t - N_f$. This time the search region is the union of squares centered at the positions of the patches in $L_{t'+1}$.

4. Remove the candidates with distance (1) larger than τ from

$$L = \bigcup_{t'=t-N_f}^{t+N_f} L_{t'},$$

the set combining all the candidates computed in each frame.

5. Keep the N best candidates from L.

6. The Haar transform used for the domain transform requires a number of patches that is a power of 2. This is why only the best patches up to the largest power of 2 smaller or equal to N are kept. It is, however, possible to keep all patches when using a different domain transform than the recommended Haar transform.

These steps are summarized in Algorithms 2 and 3.

Algorithm 1: VBM3D algorithm

```
input  : A video $v$, the noise variance $\sigma^2$, the number of similar patches to compute $N$, a
         number of temporal frames $N_f$, the size of the search region in the reference frame
         $N_s$, the size of the search region in the other frames $N_{pr}$, the number of patches
         kept in each frame $N_b$, the size of the patch $k$, a correcting factor $\delta$, the distance
         threshold $\tau$, the thresholding parameter $\lambda_{3D}$, the domain transform $T_{3D}$, the
         coefficient for the Kaiser window $\beta$, the step of the grid on which the patches are
         taken $st$
output: A final estimate denoised video $\hat{v}^{(2)}$
$K_1 \leftarrow$ Kaiser window of size $k_{hard}$ and coefficient $\beta_{hard}$
$K_2 \leftarrow$ Kaiser window of size $k_{wien}$ and coefficient $\beta_{wien}$
// Step 1
for each $\boldsymbol{p}$ on the grid of step $st_{hard}$ do
    // Search for similar patches in the noisy video
    $L_{\boldsymbol{p}} \leftarrow$ compute_similar_patches($v, \boldsymbol{p}, N_{hard}, N_f, N_s, N_{pr}, N_b, k_{hard}, d_{hard}, \tau_{hard}$)
    // Filter the group of patches using a hard thresholding
    $(\hat{L_{\boldsymbol{p}}}, \omega) \leftarrow$ ht_filtering($L_{\boldsymbol{p}}, T_{3D,hard}, \sigma^2$)
    for $\boldsymbol{q} \in \hat{L_{\boldsymbol{p}}}$ do
        $a(\boldsymbol{q}) \leftarrow a(\boldsymbol{q}) + \omega K_1 \otimes \boldsymbol{q}$
        $w(\boldsymbol{q}) \leftarrow w(\boldsymbol{q}) + \omega K_1$
for each pixel $x$ do
    $\hat{v}^{(1)}(x) \leftarrow a(x)/w(x)$
// Step 2
for each $\boldsymbol{p}$ on the grid of step $st_{wien}$ do
    // Search for similar patches in the basic estimate
    $L'_{\boldsymbol{p}} \leftarrow$ compute_similar_patches($\hat{v}^{(1)}, \boldsymbol{p}, N_{wien}, N_f, N_s, N_{pr}, N_b, k_{wien}, d_{wien}, \tau_{wien}$)
    $L_{\boldsymbol{p}} \leftarrow$ patches from $v$ at the same position than the one from $L'_{\boldsymbol{p}}$
    // Filter the group of patches using a Wiener filtering
    $(\hat{L_{\boldsymbol{p}}}, \omega) \leftarrow$ wiener_filtering($L_{\boldsymbol{p}}, L'_{\boldsymbol{p}}, T_{3D,wien}, \sigma^2$)
    for $\boldsymbol{q} \in \hat{L_{\boldsymbol{p}}}$ do
        $a(\boldsymbol{q}) \leftarrow a(\boldsymbol{q}) + \omega K_2 \otimes \boldsymbol{q}$
        $w(\boldsymbol{q}) \leftarrow w(\boldsymbol{q}) + \omega K_2$
for each pixel $x$ do
    $\hat{v}^{(2)}(x) \leftarrow a(x)/w(x)$
return $\hat{v}^{(2)}$
```

2.2 Patch Stack Filtering: Hard-threshold Step

The first step processes each frame by filtering stacks of similar patches and aggregating them in an output frame. The reference patches for the stacks are all the patches on a sub-grid of step st_{hard}. For each group there is first an estimation, followed by an aggregation.

Estimation. For a given reference patch $\boldsymbol{p}$, we first find its similar patches L as described in Section 2.1. These patches are stacked in a 3D volume $\mathcal{P}(\boldsymbol{p})$, of size $k \times k \times N$. The first spatial slice of this stack is the reference patch, and the remaining ones are the similar patches in L ordered

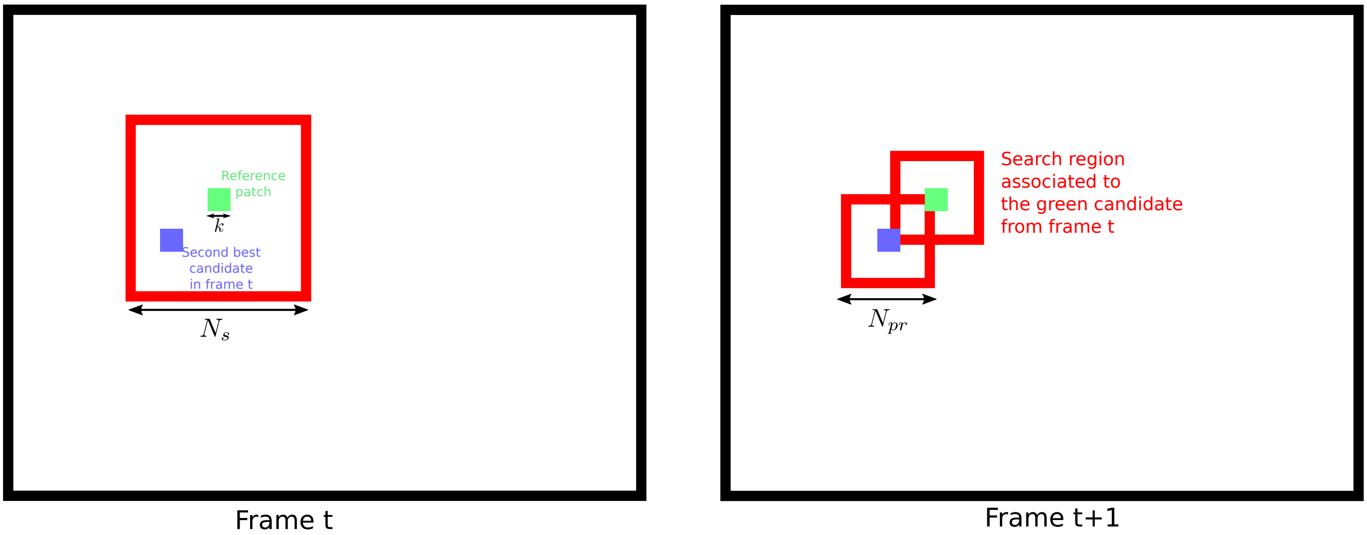
Reference patch
k
Second best candidate in frame t
N_s
Frame t
Search region associated to the green candidate from frame t
N_{pr}
Frame t+1

Figure 2: The patch search starts with a local spatial search in the current frame of size $N_s \times N_s$. Only the N_b best candidates are then used in the following frame to predict where to search. This leads to N_b local spatial searches (each centered at the candidates from the previous frame), of size $N_{pr} \times N_{pr}$. In practice, $N_s = 7$, $N_{pr} = 5$ and $N_b = 2$.

```
Algorithm 2: compute_similar_patches: VBM3D patch search
input : Noisy video $v = (v_0, \dots, v_f)$, a reference patch $\boldsymbol{p}$, the number of patches to return
        $N$, a number of neighboring frames $N_f$, a search region for the reference frame $N_s$,
        a search region for the other frames $N_{pr}$, the maximum number of similar patches
        within each local search $N_b$, the patch size $k$, a correcting factor $\delta$, a maximum
        distance threshold $\tau$
output: $L$ the list of the $N$ patches closest to $\boldsymbol{p}$ and their distance
$t \leftarrow$ frame at which $\boldsymbol{p}$ is located
$L_t \leftarrow$ local_search($\boldsymbol{p}, \boldsymbol{p}, N_s, k, N_b, d, u_t$) // Centered at $\boldsymbol{p}$ using $\boldsymbol{p}$ as a reference
// Search in the $N_f$ following frames
for $t_f = (t+1)$ to $\min(t+N_f, f-1)$ do
    for $\boldsymbol{q} \in L_{t_f - 1}$ do
        $L_{t_f} \leftarrow$ local_search($\boldsymbol{q}, \boldsymbol{p}, N_{pr}, k, N_b, d, u_{t_f}$) // Centered at $\boldsymbol{q}$ using $\boldsymbol{p}$ as reference
// Search in the $N_f$ previous frames
for $t_p = t-1$ to $\max(t-N_f, 0)$ do
    for $\boldsymbol{q} \in L_{t_p + 1}$ do
        $L_{t_p} \leftarrow$ local_search($\boldsymbol{q}, \boldsymbol{p}, N_{pr}, k, N_b, d, u_{t_p}$) // Centered at $\boldsymbol{q}$ using $\boldsymbol{p}$ as reference
$L \leftarrow \bigcup_{i \in [\![ t-N_f ; t+N_f ]\!]} L_i$
$L \leftarrow$ elements from $L$ with distance smaller than $\tau$
// the 3D transform requires a number of patches which is a power of 2 (see
   Section 2.1, step 6)
$N_l \leftarrow$closest power of 2 small or equal than $\min(\#L, N)$
if $L$ has more than $N_l$ elements then
    $L \leftarrow N_l$ best candidates from $L$
return $L$
```

by their distance to $\boldsymbol{p}$. A 3D domain transform is first applied to the group of patches followed by a thresholding of the resulting spectrum (excepting the DC component of every patch). The inverse

Algorithm 3: `local_search`: Local search

```
input : A patch $\boldsymbol{p}$ center of the search region, a reference patch $\boldsymbol{p}'$, the search region size $s$,
        the patch size $k$, the number of patches to return $N_b$, a correcting factor $\delta$, a frame
        $u$
output: list $L$ of the patches closest to $\boldsymbol{p}'$ in the region described by $\boldsymbol{p}$ and $s$ and their
        distance
// Compute the distances for all the patches in the local region
for each patch $\boldsymbol{q}$ of size $k \times k$ in the spatial region of size $s \times s$ centered at the position of $\boldsymbol{p}$
 do
    if $\boldsymbol{q}$ is at the same spatial position than $\boldsymbol{p}$ then
        Add $(\|\boldsymbol{q} - \boldsymbol{p}'\|_2^2 - d, \boldsymbol{q})$ to $L$
    else
        Add $(\|\boldsymbol{q} - \boldsymbol{p}'\|_2^2, \boldsymbol{q})$ to $L$
Sort $L$ according to the value of the distance
Keep the $N_b$ best elements in $L$
return $L$
```

3D domain transform is then applied to the thresholded coefficients to obtain the estimation, i.e.

$$\widehat{\mathcal{P}(\boldsymbol{p})} = T_{3D}^{-1}(HT_\lambda(T_{3D}(\mathcal{P}(\boldsymbol{p})))), \tag{2}$$

where HT is defined by

$$HT_{\lambda\sigma}(\nu) = \begin{cases} \nu \text{ if } \nu \text{ is a DC component or } |\nu| > \lambda\sigma \\ 0 \text{ otherwise.} \end{cases} \tag{3}$$

In practice the 3D transforms are chosen to be separable, consisting of a 2D spatial transform applied directly on the patches of $\mathcal{P}(\boldsymbol{p})$, typically a bi-orthogonal wavelet transform, followed by a 1D transform along the third dimension of the stack, typically a Haar transform. The pseudocode of the estimation for the first step is presented in Algorithm 4.

Aggregation. An output pixel is estimated several times, since it belongs to several patches and each patch can be estimated multiple times (once for each group it belongs to). To compute the output frame $\widehat{u}$ these estimates are aggregated. For a pixel ρ of the frame, this aggregation is performed as

$$\widehat{u}(\rho) = \frac{\sum_{\text{patch } \boldsymbol{p} \text{ of } v} \sum_{\boldsymbol{q} \in \widehat{\mathcal{P}(\boldsymbol{p})}} \omega_{\boldsymbol{p}}(\rho)\boldsymbol{q}(\rho)}{\sum_{\text{patch } \boldsymbol{p} \text{ of } v} \omega_{\boldsymbol{p}}(\rho)}, \tag{4}$$

where a patch $\boldsymbol{p}$ has a support of the size of the image and is zero everywhere except on the actual location of the patch. The weights ω used during the aggregation have two components. A "per-stack" component w_{ht} computed during the estimation is $w_{ht}(\boldsymbol{p}) = \frac{1}{\sigma^2 N_{hard}(\boldsymbol{p})}$ where $N_{hard}(\boldsymbol{p})$ corresponds to the number of coefficients that have not been thresholded during the estimation of $\widehat{\mathcal{P}(\boldsymbol{p})}$. Each coefficient $w_{ht}(\boldsymbol{p})$ is properly defined because $N_{hard}(\boldsymbol{p})$ is always positive as the DC component is never thresholded. These weights penalize estimates with large variance, as proposed in [13]. In addition a "per-pixel" aggregation weight is used to smooth out the blocking artifacts that result from working with patches. This weight comes from a 2D Kaiser window $K(\boldsymbol{p})$ of size $k_{hard} \times k_{hard}$

(see Equation (6)) located on the position of $\boldsymbol{p}$. The Kaiser window of size $k_x \times k_y$ of parameter β is

$$K(\boldsymbol{p})(x,y) = I_0\left(\beta\sqrt{1-(2x/k_x)^2}\right) I_0\left(\beta\sqrt{1-(2y/k_y)^2}\right)/I_0(\beta)^2, \tag{5}$$

$$\text{with } 0 \leqslant x \leqslant k_x, 0 \leqslant y \leqslant k_y \text{ and } I_0 \text{ the zeroth-order modified Bessel function.} \tag{6}$$

Finally the (patch) weight is defined by $\omega_{\boldsymbol{p}} = w_{ht}(\boldsymbol{p})K(\boldsymbol{p})$. The pseudocode with the aggregation step can be found in Algorithm 1.

Algorithm 4: `ht_filtering`: Hard thresholding

```
input : A group of similar patches $L$, a 3D transform $T_{3D}$, the noise variance $\sigma^2$
output: A list of filtered patches $\hat{L}$, the aggregation weight $\omega$
$L \leftarrow T_{3D}(L)$
$n \leftarrow 0$
for each patch $\mathbf{p}$ in $L$ do
    for each coefficient $\rho$ of $\mathbf{p}$ do
        if $\mathbf{p}(\rho) > \lambda\sigma$ or $\rho$ is the DC component then
            $n \leftarrow n + 1$
            $\hat{\mathbf{p}}(\rho) \leftarrow \mathbf{p}(\rho)$
        else
            $\hat{\mathbf{p}}(\rho) \leftarrow 0$
$\omega = \frac{1}{\sigma^2 n}$
$\hat{L} \leftarrow T_{3D}^{-1}(L)$
return $\hat{L}, \omega$
```

2.3 Patch Stack Filtering: Wiener Filtering Step

The second step of the algorithm uses the basic estimate computed during the first step for the patch search to compute the coefficients of a Wiener shrinkage of the transformed coefficients. The video is processed by building groups of patches around reference patches from a coarse subgrid with step st_{wien}.

Estimation. Given a reference patch $\boldsymbol{p}$, a group of similar patches is selected as described in Section 2.1, but using patches extracted from the basic estimate for computing the patch distance. Two sets of patches are extracted: one from the noisy sequence and the other one from the basic estimate at the same locations. Each set of patches is stacked in a 3D volume: $\mathcal{P}(\boldsymbol{p})$ for the noisy patches and $\mathcal{P}(\hat{\boldsymbol{p}})'$ for patches of the basic sequence. The noisy stack is denoised with a Wiener filter. The filtering coefficients are estimated from $\mathcal{P}(\hat{\boldsymbol{p}})'$. Just like in the first step, a 3D domain transform is first applied to the group of patches before the application of the Wiener filter. This results in

$$\widehat{\mathcal{P}(\boldsymbol{p})} = T_{3D}^{-1}(WF(T_{3D}(\mathcal{P}(\boldsymbol{p})))), \tag{7}$$

where the Wiener attenuation coefficient WF applied to $\mathbf{p}$ on a frequency ν is equal to

$$\frac{\mathbf{p}'(\nu)^2}{\mathbf{p}'(\nu)^2 + \sigma^2}, \tag{8}$$

$\mathbf{p}'$ being here the basic estimate of $\mathbf{p}$.

In practice the 3D transforms are chosen as a 2D transform applied directly on the patches of $\mathcal{P}(\boldsymbol{p})$ and $\mathcal{P}(\hat{\boldsymbol{p}})$, typically a DCT, followed by a 1D transform along the third dimension of the group, typically a Haar transform. The pseudocode of the estimation for the first step is presented in Algorithm 5.

Algorithm 5: `wiener_filtering`: Wiener thresholding

```
input  : A group of similar patches $L$, a first estimate of $L$ called $L'$, a 3D transform $T_{3D}$,
         the noise variance $\sigma^2$
output: A list of filtered patches $\hat{L}$, the aggregation weight $\omega$
$L \leftarrow T_{3D}(L)$
$\omega \leftarrow 0$
for each patch $\boldsymbol{p}$ in $L$, $\mathbf{p}'$ the corresponding patch in $L'$ do
    for each coefficient $\rho$ of $\mathbf{p}$ do
        $\alpha \leftarrow \frac{\boldsymbol{p}'(\rho)^2}{\boldsymbol{p}'(\rho)^2+\sigma^2}$
        $\hat{\boldsymbol{p}}(\rho) \leftarrow \alpha \boldsymbol{p}(\rho)$
        $\omega \leftarrow \omega + \alpha^2$
$\omega \leftarrow \frac{1}{\sigma^2 \omega}$
$\hat{L} \leftarrow T_{3D}^{-1}(\hat{L})$
return $\hat{L}$, $\omega$
```

Aggregation. Just as with the first step, the estimation gives multiple estimates per pixel. Therefore the different estimates are aggregated using the same principle as in Section 2.2, defined by Equation (4). The only difference lies in the weights. The weights ω used during the aggregation are computed in part during the estimation. The part w_{wf} computed during the estimation is $w_{wf}(\boldsymbol{p}) = \frac{1}{\sigma^2}\left(\sum_\rho \left(\frac{\hat{\boldsymbol{p}}_i(\rho)^2}{\hat{\boldsymbol{p}}_i(\rho)^2+\sigma^2}\right)^2\right)^{-1}$ which is actually linked to the squared ℓ_2 norm of the vector of coefficients used for the filtering. The rest of the weight comes from a 2D Kaiser window $K(\boldsymbol{p})$ of size $k_{wien} \times k_{wien}$ applied to avoid boundary effects from the patches. Finally the pixel weight is defined by $\omega_{\boldsymbol{p}}(x) = w_{wf}(\boldsymbol{p})K(\boldsymbol{p})$. The pseudocode with the aggregation step can be found in Algorithm 1.

2.4 Reproducing the Original VBM3D

We now compare the results obtained with our implementation and with the binaries released by the authors of the original VBM3D [7][2]. Throughout this work we will use a test set of seven grayscale test sequences obtained from the *Derf's Test Media collection*[3]. The original sequences are of higher resolution and in RGB. They have been downscaled and converted to grayscale.

Table 1 compares the PSNR obtained by our implementation to that of the original binaries. Our results are slightly lower: The average gap starts at 0.27dB for $\sigma = 10$ and reduces to 0.18dB for $\sigma = 40$. *Station* and *sunflower* are the sequences where the gap is larger, reaching 0.68dB for the later at $\sigma = 10$. A visual comparison of both results is shown in Figure 3, for noise $\sigma = 40$. Both results are visually very similar, with the same type of artifacts. A closer inspection reveals that the result of the original VBM3D is slightly smoother, resulting in less noticeable DCT artifacts and on some textures with decreased contrast (see for instance the grass in the rightmost figure).

[2] Available at `http://www.cs.tut.fi/~foi/GCF-BM3D/`.

[3] `https://media.xiph.org/video/derf/`.

σ	Method	Crowd	Park	Pedestrians	Station	Sunflower	Touchdown	Tractor	Average
10	VBM3D (original)	35.65	34.75	40.83	38.93	40.49	39.04	37.01	38.10
	VBM3D (ours)	35.52	34.59	40.65	38.38	39.81	39.01	36.82	37.83
20	VBM3D (original)	32.25	31.25	36.94	35.45	36.46	36.08	33.07	34.50
	VBM3D (ours)	32.06	31.12	36.81	35.10	35.95	36.05	32.97	34.30
40	VBM3D (original)	28.65	27.68	32.81	32.02	32.65	33.52	29.41	30.96
	VBM3D (ours)	28.39	27.64	32.62	31.80	32.31	33.35	29.38	30.78

Table 1: Comparison of the denoising quality with the binary program provided by Dabov et al. with [8]. Results were both computed using the normal profile 'np' parameters.

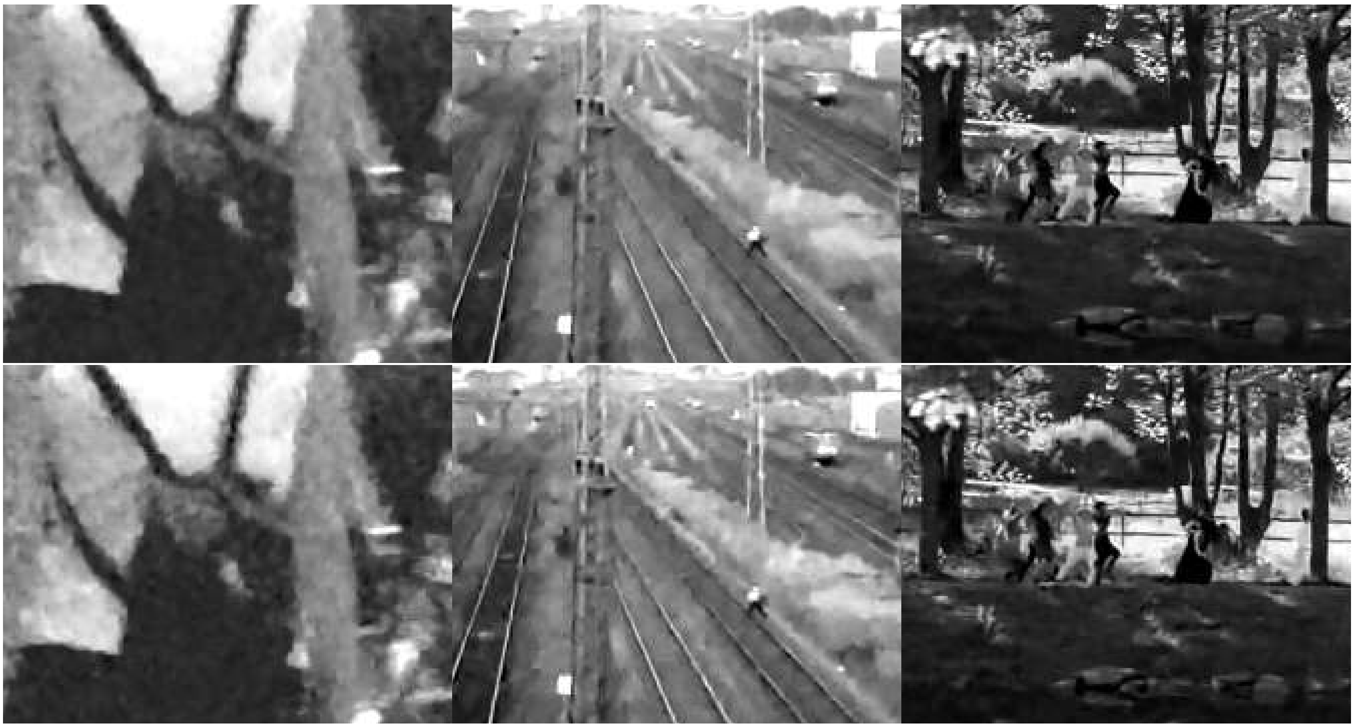

Figure 3: Top: results of VBM3D [7], with the original authors' implementation. Bottom: result obtained with our implementation. The noise level is $\sigma = 40$. The contrast has been linearly scaled for better visualization.

3 Extensions

3.1 Using Optical Flow to Guide the Search

Many video denoising methods take advantage of the optical flow as an estimate of the motion in the video. Typically, optical flow is used by video denoising methods that require aggregating information along motion trajectories [4, 14, 6, 11, 3]. These methods require a motion estimate as accurate as possible. Most patch-based methods, on the other hand, do not require such an accurate motion estimate, as they are based on finding similar patches in a 3D search region [5, 7, 15]. These methods either do not use any motion estimate at all, or use a very rough one (e.g. block matching) to guide the search region. However, it has been observed that using optical flow to shape the search region can still be beneficial for patch based methods [2, 1], as it allows to find better matches.

For a pair of two images A and B, the optical flow aims at finding a vector field $o(x, y) = (\delta_x, \delta_y)$

such that any point (x, y) of the image domain solves

$$A(x + \delta_x, y + \delta_y) = B(x, y). \tag{9}$$

The displacement vector can be sub-pixel, in which case the image A needs to be interpolated. We decided to use the TV-L1 optical flow method [22], in particular the implementation provided in [20]. We compute the optical flow in a downscaled version of the video by a factor of 4, and scale it back to the original resolution. This reduces the running time and the impact of the noise while still having a reasonable precision, as shown in [11].

We add the optical flow to VBM3D as a guide, the same way it is done for VNLB [2]. The spatio-temporal search region is defined as two sequences of N_f square windows of size $N_{pr} \times N_{pr}$ (plus the $N_s \times N_s$ window corresponding to the reference frame), whose centers follow the motion trajectory of the reference patch. The trajectory is estimated using the forward and backward optical flow. We use the center corresponding to the position of the reference patch propagated in previous, respectively following, frames using the backward, respectively forward, optical flow. The forward half of the trajectory $\varphi_{x,y,t}$ passing through (x, y, t) is computed by integrating the forward optical flow o^f (also indexed by time on top of the spatial position) as follows

$$\varphi_{x,y,t}(h) = o^f([\varphi_{x,y,t}(h-1)], h-1) + \varphi_{x,y,t}(h-1), \quad h = t+1, \ldots, t+N_f, \tag{10}$$

where $[\cdot]$ and $\lfloor\cdot\rfloor$ denote the round and floor operators. The backward half of the trajectory is defined analogously using the backward optical flow o^b.

This also means that only one center needs to be tracked, in contrast to the N_b required by a regular VBM3D search. Parameters are kept the same as in the original algorithm. The guided version of the search is summarized in Algorithm 6.

We also tested setting the parameter d to zero for both steps since one can assume that it would be redundant with the regularization of the patch search offered by the optical flow. PSNR results are shown in Table 2. Using the optical flow as a guide greatly improves the quality of the denoising. The non-zero d yields better results both with and without the optical flow guide, thus we decided to keep it in our final version.

Figure 4 shows results obtained with the different extensions that will be described in this section. The first column corresponds to VBM3D with the default parameters, and the 3rd column to the one using an optical flow guided search region (denoted "VBM3D OF"). Guiding the patch search enables the recovery of more details. This is because the tracking of patches is more robust to motion than the block-matching suggested for the original VBM3D and therefore provides better matches. Moreover, the optical flow is not required to be very precise, as it is used only as a guide for the center of the search region.

3.2 Spatio-temporal Patches

The patch similarity is determined based on the (squared) Euclidean distance between patches. Due to the AGWN noise, the Euclidean distance between two non-overlapping patches follows a non-central χ^2 distribution, with variance

$$\operatorname{var}\left\{\frac{1}{m}\|\boldsymbol{q}_1 - \boldsymbol{q}_2\|^2\right\} = \frac{8\sigma^2}{m}\left(\sigma^2 + \frac{1}{m}\|\boldsymbol{p}_1 - \boldsymbol{p}_2\|^2\right), \tag{11}$$

where $\boldsymbol{q}_1, \boldsymbol{q}_2$ are the noisy versions of the patches $\boldsymbol{p}_1, \boldsymbol{p}_2$, and m denotes the number of pixels in the patch (see [9] Equation (3)). The noise in the distance can be reduced by considering larger patches. However, increasing the spatial size of the patch also increases the distances between patches and

Algorithm 6: `compute_similar_patches`: VBM3D patch search guided by the optical flow

```
input  : Noisy video $v = (v_0, \dots, v_f)$, a reference patch $\boldsymbol{p}$, the number of patches to return
         $N$, a number of temporal frames $N_f$, a search region for the reference frame $N_s$, a
         search region for the other frames $N_{pr}$, the maximum number of patches to
         compute with each local search $N_b$, the size of the patch $k$, a correcting factor $\delta$, a
         maximum distance threshold $\tau$, the forward and backward optical flows $o_f$ and $o_b$
output: $L$ the list of the $N$ patches closest to $\boldsymbol{p}$ and their distance
$t \leftarrow$ frame at which $\boldsymbol{p}$ is located
$L_t \leftarrow$ local_search$(\boldsymbol{p}, \boldsymbol{p}, N_s, k, N_b, d, u_t)$
// Search in the $N_f$ following frames
$\boldsymbol{q} \leftarrow \boldsymbol{p}$
for $t_f = (t+1)$ to $\min(t+N_f, f-1)$ do
    $\boldsymbol{q} \leftarrow o_f(\boldsymbol{q}, t_f - 1)$          //Follow the center using the forward optical flow
    $L_{t_f} \leftarrow$ local_search$(\boldsymbol{q}, \boldsymbol{p}, N_{pr}, k, N_b, d, u_{t_f})$
// Search in the $N_f$ previous frames
$\boldsymbol{q} \leftarrow \boldsymbol{p}$
for $t_p = t-1$ to $\max(t-N_f, 0)$ do
    $\boldsymbol{q} \leftarrow o_b(\boldsymbol{q}, t_p + 1)$          //Follow the center using the forward optical flow
    $L_{t_p} \leftarrow$ local_search$(\boldsymbol{q}, \boldsymbol{p}, N_{pr}, k, N_b, d, u_{t_p})$
$L \leftarrow \bigcup_{i \in [\![t-N_f; t+N_f]\!]} L_i$
$L \leftarrow$ elements from $L$ with distance smaller than $\tau$
// the 3D transform requires a number of patch which is a power of 2 (see
   Section 2.1, step 6)
$N_l \leftarrow$closest power of 2 small or equal than $min(sizeof L, N)$
if $L$ has more than $N_l$ elements then
    $L \leftarrow N_l$ best candidate from $L$
return $L$
```

reduces the likelihood of finding similar ones. Adding an additional temporal dimension in a spatio-temporal patch allows to increase the number of pixels in the patch, without increasing its spatial size. Due to the high redundancy of the video in the temporal dimension, increasing the temporal size of the patch causes a much lower increase in the patch distances.

When the motion is known or can be estimated, then it is natural to consider motion-compensated spatio-temporal patches (see for instance [15, 6]). Alternatively, rectangular spatio-temporal patches with no motion compensation have been also used [19, 2]. For more complex types of motion (such as the shake in a video acquired with a hand-held camera), using rectangular spatio-temporal patches will result in a larger variability in the set of nearest neighbors of a given patch, due to the fact that both the spatial texture and the motion pattern may vary. At least in principle, better results should be obtained using motion compensation, albeit with a greater dependence on the quality of the motion estimate. In this work we do not pursue this option.

The principle of BM3D has been applied to 3D patches with and without motion compensation. VBM4D, introduced in [15], uses motion-compensated spatio-temporal patches for video denoising (the “V” stands for video). The motion is estimated using block matching. BM4D uses cubic patches without motion compensation (of size $4 \times 4 \times 4$ or $5 \times 5 \times 5$), aiming at filtering volumetric images [16]. In [16] the authors compare the performance of both VBM4D and BM4D on video denoising concluding that VBM4D was the best video filtering strategy.

σ	Method	Crowd	Park	Pedestrians	station	Sunflower	Touchdown	Tractor	Average
10	VBM3D (without)	35.52	34.59	40.65	38.38	39.81	39.01	36.82	37.83
	VBM3D (without,d=0)	35.72	35.00	40.13	39.34	40.18	39.01	36.94	38.05
	VBM3D (with)	35.61	34.94	**41.04**	39.79	41.75	**39.89**	38.43	38.78
	VBM3D (with,d=0)	**35.78**	**35.19**	40.61	**40.27**	**41.85**	39.75	**38.51**	**38.85**
20	VBM3D (without)	32.06	31.12	36.81	35.10	35.95	36.05	32.97	34.30
	VBM3D (without,d=0)	32.00	31.24	36.13	35.38	36.09	35.83	33.03	34.24
	VBM3D (with)	**32.17**	31.44	**37.32**	36.26	**38.02**	**36.91**	**34.58**	**35.24**
	VBM3D (with,d=0)	32.10	**31.49**	36.72	**36.32**	37.98	36.61	**34.58**	35.11
40	VBM3D (without)	28.39	27.64	32.62	31.80	32.31	33.35	29.38	30.78
	VBM3D (without,d=0)	28.29	27.65	32.24	31.79	32.33	33.11	29.40	30.69
	VBM3D (with)	**28.55**	**27.99**	**33.29**	**32.80**	**34.46**	**34.00**	**30.79**	**31.70**
	VBM3D (with,d=0)	28.45	27.98	32.94	32.75	34.40	33.79	30.76	31.58

Table 2: Comparison of the denoising quality with and without guiding with an optical flow. Guiding the search leads to much better results. It is also better in general to keep the additional regularization given by d.

However, in [2, 1] it is shown that rectangular spatio-temporal patches with a temporal size of only two frames improve the denoising quality and still provide higher temporal consistency than a 2D patch. Based on those results, in this work we evaluate rectangular spatio-temporal patches of size $8 \times 8 \times 2$ in the first step and $7 \times 7 \times 2$ in the second (i.e. we use the same spatial patch size as in the original work but increase the temporal patch size).

In Table 3 we compare the quantitative results obtained by using spatial and spatio-temporal patches (denoted by "ST" in the table). We also consider the effect of guiding the patch search using an optical flow, indicated as "OF".

The first four columns of Figure 4 show results with/out motion-compensated search and spatio-temporal patches. From a qualitative point of view, using spatio-temporal patches provides better temporal consistency. In addition, the patch distance is more reliable since the number of pixels in the patches is doubled. This helps retrieve details and textures for regions with a simple motion (e.g. translational). For low noise levels, the effect of these 3D patches is mixed. While the results seem more consistent temporally, they are blurrier for sequences with complex motions. This explains the drop in PSNR observed for *pedestrians, sunflower* and *touchdown* between VBM3D and VBM3D ST for $\sigma = 10$. As the noise level increases this detail loss is out-weighted by the increased robustness to noise of the patch matching.

When spatio-temporal patches are used in conjunction with the optical-flow-guided search, their positive impact is magnified. Although the patches are not themselves motion-compensated, having a motion-compensated search region helps find better matches, even in sequences with more complex motion patterns. The motion-compensated search region also improves the temporal consistency, although to a lesser extent than the spatio-temporal patches. The best result, both in terms of PSNR and temporal consistency, is obtained when both strategies are used together (VBM3D ST+OF).

3.3 Multiscale Video Denoising

Most denoising algorithms work by processing local regions of the image/video (a patch, a group of patches, the receptive field of a neuron, etc). As a result, these methods fail to remove the lower frequencies of the noise. This results in smooth bumps mostly noticeable in large non-textured areas. Multiscale approaches are able to reduce this artifact by applying the denoising algorithm at different

σ	Method	Crowd	Park	Pedestrians	Station	Sunflower	Touchdown	Tractor	average
10	VBM3D [7]	35.65	34.75	40.83	38.93	40.49	39.04	37.01	38.10
	VBM3D (ours)	35.52	34.59	40.65	38.38	39.81	39.01	36.82	37.83
	VBM3D ST	**35.65**	34.66	40.41	38.55	39.65	38.91	36.90	37.82
	VBM3D OF	35.61	**34.94**	**41.04**	**39.79**	**41.75**	**39.89**	**38.43**	**38.78**
	VBM3D ST+OF	**35.74**	**35.04**	**41.01**	**40.41**	**41.91**	**39.98**	**38.71**	**38.97**
20	VBM3D [7]	32.25	31.25	36.94	35.45	36.46	36.08	33.07	34.50
	VBM3D (ours)	32.06	31.12	36.81	35.10	35.95	36.05	32.97	34.30
	VBM3D ST	**32.39**	31.36	36.97	35.57	36.14	36.16	33.23	34.55
	VBM3D OF	32.17	**31.44**	**37.32**	**36.26**	**38.02**	**36.91**	**34.58**	**35.24**
	VBM3D ST+OF	**32.48**	**31.71**	**37.61**	**37.02**	**38.45**	**37.19**	**35.18**	**35.66**
40	VBM3D [7]	28.65	27.68	32.81	32.02	32.65	33.52	29.41	30.96
	VBM3D (ours)	28.39	27.64	32.62	31.80	32.31	33.35	29.38	30.78
	VBM3D ST	**29.18**	**28.13**	**33.35**	32.50	32.70	33.65	29.66	31.31
	VBM3D OF	28.55	27.99	33.29	**32.80**	**34.46**	**34.00**	**30.79**	**31.70**
	VBM3D ST+OF	**29.30**	**28.50**	**34.21**	**33.68**	**35.06**	**34.47**	**31.46**	**32.38**

Table 3: Quantitative denoising results (PSNR and SSIM) for seven grayscale test sequences of size 960 × 540 from the *Derf's Test Media collection* for several variants of VBM3D. We highlighted the best performance in black and the second best in brown.

scales.

There are two main approaches for multiscale denoising in the literature. The first one consists in modifying the denoising algorithm to consider several scales of the image/video. See for instance the multiscale version of the EPLL method [23] proposed by Papyan and Elad [17]. Another approach proposed in [12, 18] considers the denoising algorithm as a black box, applying it as is at multiple scales. The result is then obtained by merging the results of each scale. This has the benefit that it can be applied to any denoising method, without any adaptation needed.

The multiscaler of [12] first creates a pyramid from the noisy input image $v^0 = v$ by applying a downscaling operator $\mathcal{D}$

$$v^{s+1} = \mathcal{D}(v^s), \text{ for } s = 0, \dots, S-1. \tag{12}$$

At each scale s, the denoising algorithm is applied with the same parameters except for the noise level set at $\sigma/2^s$ (see [12] for more details). This yields a denoised image $\hat{u}^s$. These images are then recomposed into a single multiscale output $\hat{u}_{\text{ms}}$. The recomposition is recursive. The recursion is initialized at the coarsest level by defining $\hat{u}^S_{\text{ms}} = \hat{u}^S$, and then proceeds as follows

$$\hat{u}^s_{\text{ms}} = \hat{u}^s - \mathcal{U}(\mathcal{L}(\mathcal{D}(\hat{u}^s), f_{\text{rec}})) + \mathcal{U}(\mathcal{L}(\hat{u}^{s+1}_{\text{ms}}, f_{\text{rec}})), \tag{13}$$

where $\mathcal{U}$ is the upscaling operator, and $\mathcal{L}(\,\cdot\,, f_{\text{rec}})$ is a low-pass filter with cutoff frequency parameterized by f_{rec} (f_{rec} does not depend on the scale). This equation substitutes the low frequencies of the single-scale result $\hat{u}^s$ with the multiscale solution $\hat{u}^{s+1}_{\text{ms}}$, computed using the coarser scales $s+1, \dots, S$. The recomposition parameter f_{rec} determines which low frequencies are substituted. In one extreme, the low-pass filter lets all frequencies pass, in which case the whole coarse solution is used. At the other end, f_{rec} filters out all frequencies: the solution at the coarser level $\hat{u}^{s+1}_{\text{ms}}$ is discarded, and the output of the recomposition is the single scale denoising $\hat{u}^0$. In [12] it is found that the optimum is to filter out some of the high frequencies of the coarser level $\hat{u}^{s+1}_{\text{ms}}$. However, the exact amount depends on the denoiser.

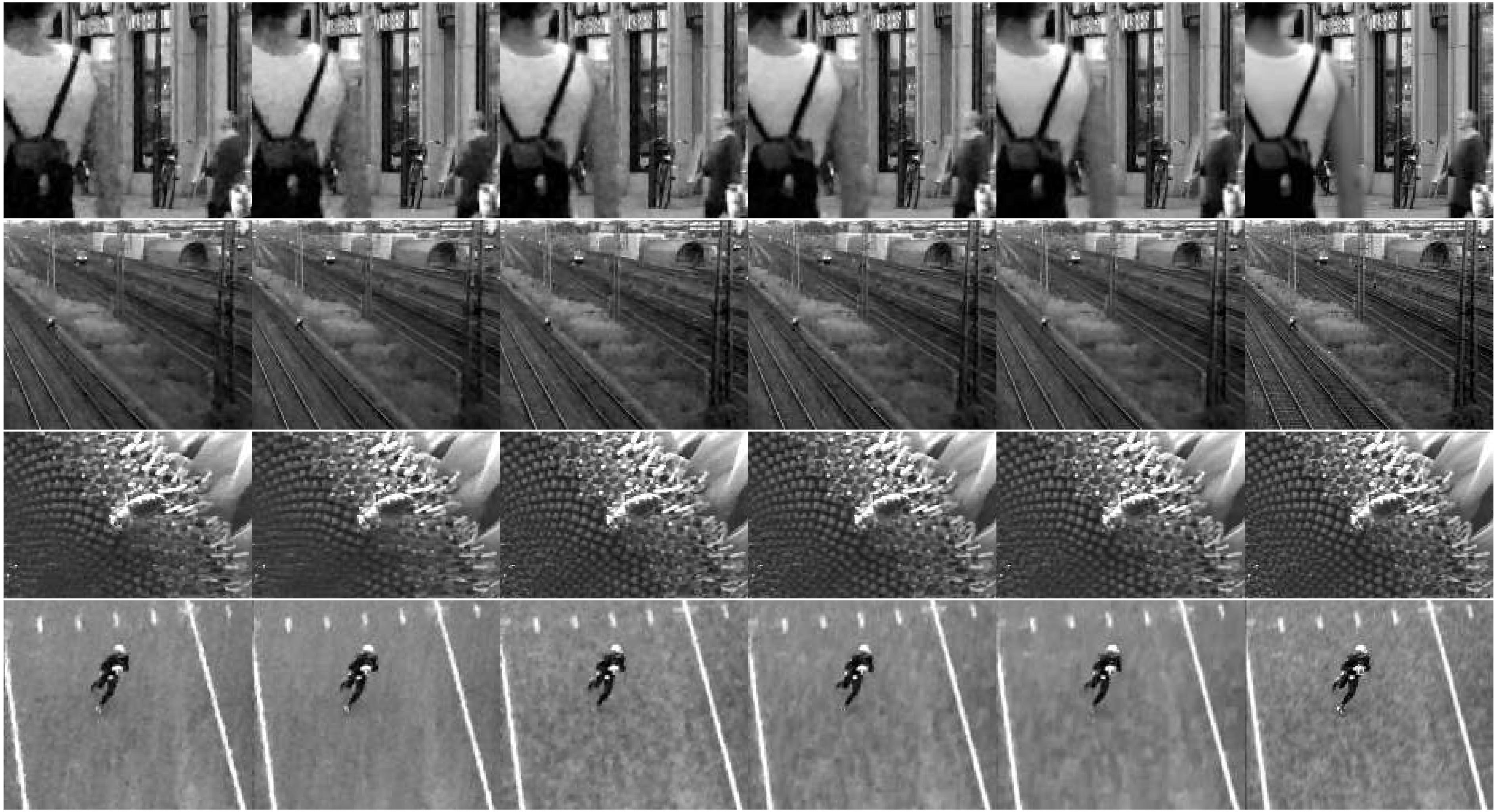

Figure 4: From left to right: result of VBM3D (our implementation with default parameters), VBM3D ST, VBM3D OF, VBM3D ST+OF, VBM3D ST+OF+MS (Lanczos multiscaler) and the original clean video. The noise level is $\sigma = 40$. Contrast has been linearly scaled for better visualization.

To apply the multiscaler on a video, we apply it spatially, i.e. the temporal dimension is not downscaled. We first create a spatial pyramid of the entire video by creating a pyramid of each frame. We then denoise these videos, and recompose them by applying Equation (13) to each frame. This is summarized in Algorithm 7.

Algorithm 7: Multiscale processing

```
input  : A video $v$, a number of scales $S$, parameters for VBM3D $params$, the downscaling
         operator $\mathcal{D}$, the upscaling operator $\mathcal{U}$, the low-pass filter $\mathcal{L}(\cdot, f_{\text{rec}})$, the noise level $\sigma$
output: The denoised video $\hat{v}_{\text{ms}}$
// Denoise the video at the original scale
$\hat{v}^0 \leftarrow \text{VBM3D}(v, \sigma, params)$
$v^0 \leftarrow v$
for scale $s$ from $1$ to $S-1$ do
    // Compute the video $v^s$ at the scale $s$ using the video at scale $s-1$ and
       Equation 12
    for each frame $v_i^{s-1}$ in $v^{s-1}$ do
        $v_i^s \leftarrow \mathcal{D}(v_i^{s-1})$
    // Denoise the video $v^s$
    $\hat{v}^s \leftarrow \text{VBM3D}(v^s, \sigma/2^s, params)$
for each frame $\hat{v}_i^{S-1}$ in $\hat{v}^{S-1}$ do
    $\hat{v}_{i,\text{ms}}^{S-1} \leftarrow \hat{v}_i^{S-1}$
for scale $s$ from $S-2$ to $0$ do
    // Compute the video $v_{\text{ms}}^s$ at the scale $s$ using the video at scale $s+1$ and
       Equation 13
    for each pair of frames $(v_i^{s-1}, \hat{v}_{i,ms}^{s+1})$ in $(v^{s-1}, \hat{v}_{ms}^{s+1})$ do
        $\hat{v}_{i,\text{ms}}^s = \hat{v}_i^s - \mathcal{U}(\mathcal{L}(\mathcal{D}(\hat{v}_i^s), f_{\text{rec}})) + \mathcal{U}(\mathcal{L}(\hat{v}_{i,\text{ms}}^{s+1}, f_{\text{rec}}))$
return $\hat{v}_{\text{ms}}^0$
```

The parameters of the multiscaler are the number of scales, the downsampling ratio and the recomposition parameter f_{rec}. We shall set the downscaling ratio to 2 (the default), and try different values for the number of scales and the recomposition factor. There are different possibilities for the down/upscaling operators and the low-pass filter.

DCT pyramid. The DCT multiscaler uses a DCT pyramid which guarantees white Gaussian noise at all scales. The downscaling is performed by computing the DCT transform of the image and keeping only the quadrant of the image corresponding to the lowest frequencies. The upscaling is done by zero-padding in the DCT domain, and the low-pass filtering zeroes out the highest frequencies in the DCT domain. In this case f_{rec} represents the ratio of frequencies that are left: $f_{\text{rec}} = 1$ corresponds to an all-pass filter, where as $f_{\text{rec}} = 0$ filters out all the image.

Laplacian pyramid. The downscaling and upscaling operators samples the image using a Lanczos kernel

$$k_a(x) = \begin{cases} \text{sinc}(x)\text{sinc}(x/a) & \text{if } |x| < a \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

We set $a = 3$. For the downscaling, to reduce aliasing, the image is downsampled using a scaled version of the kernel $k_3(\cdot/2)$ (as described in [21][4]). The low-pass filter used is the Gaussian

[4] This corresponds to Matlab's `imresize` scaling function using the `lanczos3` interpolation kernel.

filter of width f_{rec}. When $f_{\mathrm{rec}} \to \infty$ we obtain the single scale output, and if $f_{\mathrm{rec}} = 0$ no frequencies from the coarser solutions are discarded.

Figures 5 and 6 show plots of the average PSNR for our seven test sequences obtained with the two multiscalers varying the number of scales and the recomposition cutoff parameter f_{rec}. In each figure, we show the results of the multiscaler applied to the standard VBM3D, and to the one with statio-temporal patches and guided patch search (VBM3D ST+OF).

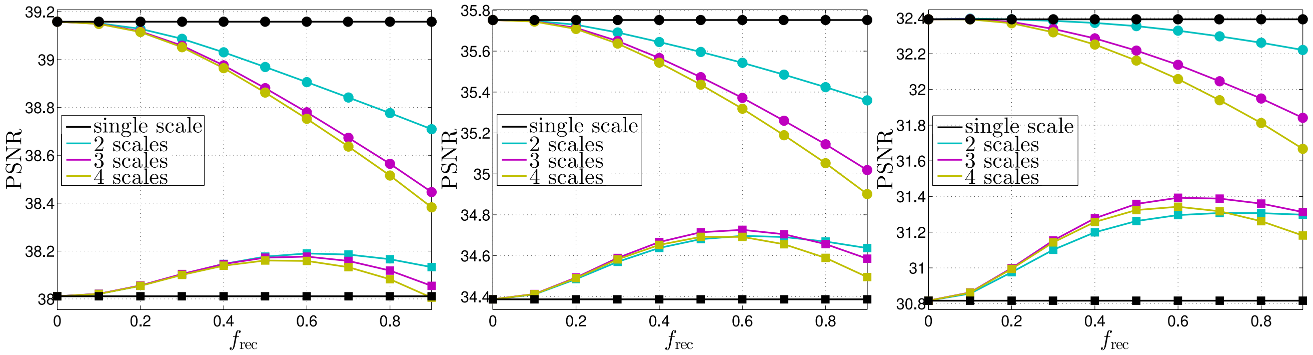

Figure 5: Effect of the DCT multiscaler for VBM3D without extensions (square markers) and VBM3D ST+OF (round markers). Each plot shows the average PSNR over our seven test sequences obtained when varying the recomposition factor f_{rec} for different number of scales. From left to right, $\sigma = 10, 20, 40$. The multiscaler has a positive effect on the original VBM3D, but not on the improved VBM3D ST+OF. In this case $f_{\mathrm{rec}} \in [0, 1]$ represents the ratio of frequencies that are left: $f_{\mathrm{rec}} = 1$ corresponds to an all-pass filter, where as $f_{\mathrm{rec}} = 0$ filters out all the image.

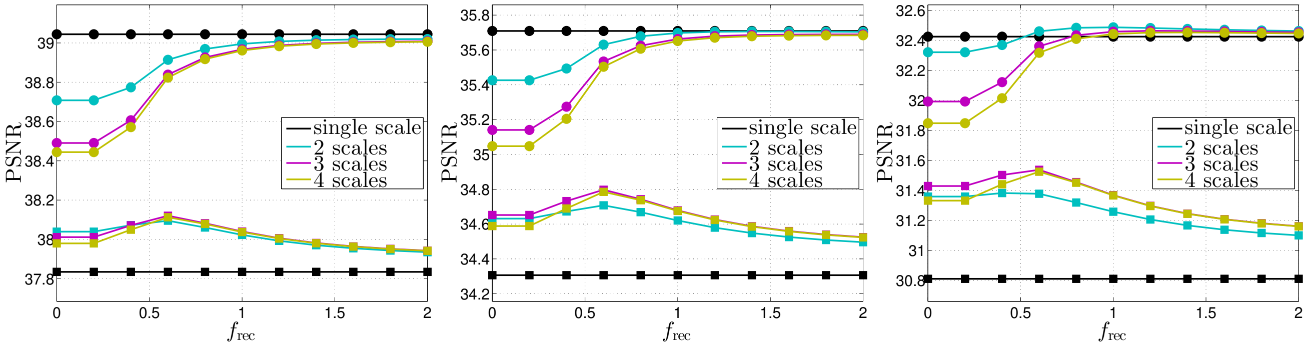

Figure 6: Effect of the Lanczos multiscaler for VBM3D without extensions (square markers) and VBM3D ST+OF (round markers). Each plot shows the average PSNR over our seven test sequences obtained when varying the recomposition factor f_{rec} for different number of scales. From left to right, $\sigma = 10, 20, 40$. The multiscaler has a positive effect on the original VBM3D, and on the improved VBM3D ST+OF for $\sigma = 40$. In this case, $f_{\mathrm{rec}} \in [0, +\infty[$ is the width of a Gaussian low-pass filter $\mathcal{L}$. When $f_{\mathrm{rec}} \to \infty$ we obtain the single scale output, and if $f_{\mathrm{rec}} = 0$ no frequencies from the coarser solutions are discarded.

Both multiscalers have a positive impact when applied to the standard VBM3D. The gain can be up to 0.3dB for noise 10, 0.5dB for noise 20 and 0.8dB for noise 40. However, when applied to the VBM3D ST+OF version of the algorithm, the multiscaler does not improve the result. In fact, for noise 10 and 20, the best PSNR is attained by the single scale algorithm and the PSNR deteriorates as the cutoff frequency of the low-pass filter is increased (i.e. as more frequency components from the coarse solution are used).

The multiscaler achieves a better denoising of large objects with smooth textures by removing low-frequency noise left by the denoiser. This low-frequency noise is much stronger for the VBM3D denoiser than for the VBM3D ST+OF version. Hence, the improvement provided by the multiscaler is smaller for the latter. This also depends on the characteristics of the sequence. Frames with smooth objects occupying larger areas will benefit from the multiscaler. Yet, the multiscaler introduces artifacts penalizing the PSNR (particularly additional ringing for the DCT multiscaler). These artifacts are not temporally coherent and can therefore be quite noticeable.

Based on these results we select the Lanczos3 multiscaler. We use 2 scales and a recomposition factor $f_{\text{rec}} = 1$ when applied to VBM3D ST+OF (i.e. VBM3D ST+OF+MS) and 3 scales with recomposition factor $f_{\text{rec}} = 0.6$ when applied to the standard VBM3D (VBM3D MS). Table 4 shows the obtained PSNRs. The visual results of VBM3D ST+OF+MS are shown in Figure 4. The impact of the multiscaler can be noticed in the top row (results for *pedestrian*) as a reduction of the low-frequency noise in the smooth areas in the image. For the other sequences, since they are highly textured, the effect of the multiscaler is subtle. A careful examination reveals some texture loss.

σ	Method	Crowd	Park	Pedestrians	Station	Sunflower	Touchdown	Tractor	average
10	VBM3D (ours)	35.52	34.59	40.65	38.38	39.81	39.01	36.82	37.83
	VBM3D MS (DCT)	35.28	34.42	40.73	38.62	40.51	39.23	36.93	37.96
	VBM3D MS (Lanczos)	35.48	34.57	40.79	38.59	40.35	39.19	36.93	37.99
	VBM3D ST+OF	**35.74**	**35.04**	**41.01**	**40.41**	**41.91**	**39.98**	**38.71**	**38.97**
	VBM3D ST+OF+MS (DCT)	35.48	34.74	40.82	39.56	41.16	39.62	38.06	38.49
	VBM3D ST+OF+MS (Lanczos)	**35.72**	**35.01**	**41.05**	**40.34**	**41.88**	**39.97**	**38.66**	**38.95**
20	VBM3D (ours)	32.06	31.12	36.81	35.10	35.95	36.05	32.97	34.30
	VBM3D MS (DCT)	31.75	30.92	37.31	35.49	37.22	36.33	33.49	34.64
	VBM3D MS (Lanczos)	32.04	31.13	37.29	35.45	36.95	36.31	33.35	34.65
	VBM3D ST+OF	**32.48**	**31.71**	**37.61**	**37.02**	**38.45**	**37.19**	**35.18**	**35.66**
	VBM3D ST+OF+MS (DCT)	32.08	31.32	37.47	36.27	**37.71**	36.71	**34.51**	35.15
	VBM3D ST+OF+MS (Lanczos)	**32.46**	**31.68**	**37.74**	**36.99**	**38.45**	**37.16**	**35.18**	**35.67**
40	VBM3D (ours)	28.39	27.64	32.62	31.80	32.31	33.35	29.38	30.78
	VBM3D MS (DCT)	28.31	27.68	33.75	32.42	33.90	33.60	30.27	31.42
	VBM3D MS (Lanczos)	28.51	27.78	33.54	32.31	33.55	33.64	30.01	31.33
	VBM3D ST+OF	**29.30**	**28.50**	34.21	**33.68**	**35.06**	**34.47**	**31.46**	**32.38**
	VBM3D ST+OF+MS (DCT)	**28.90**	28.18	**34.25**	33.19	**34.41**	34.04	31.00	32.03
	VBM3D ST+OF+MS (Lanczos)	**29.30**	**28.51**	**34.46**	**33.70**	**35.06**	**34.50**	**31.56**	**32.44**

Table 4: Quantitative denoising results (PSNR and SSIM) for seven grayscale test sequences of size 960 × 540 from the *Derf's Test Media collection* for several variants of VBM3D. We used two scales and $f_{\text{rec}} = 1$ for ST+OF and three scales and $f_{\text{rec}} = 0.6$ for ST+OF+MS. We highlighted the best performance in black and the second best in brown.

4 Comparison with the State of the Art and Conclusion

In Table 5, we compare the PSNR of different recent denoising methods with VBM3D as well as with versions of VBM3D modified using different combinations of the improvements suggested in Sections 3.3, 3.2, 3.1. We included the patch-based methods SPTWO [6] and VNLB [2] and VNLnet [10], a convolutional neural network (CNN). SPTWO and VNLB are based on grouping similar patches and filtering them on a transformed domain. However, instead of a pre-defined transform, an optimal transform is estimated for each group (the principal directions of the group of patches). While this gives good results, it requires computing an SVD of the matrix formed by the similar patches, resulting in very slow algorithms. VNLB groups 3D patches 10 × 10 × 2, similar to our spatio-temporal patches. SPTWO instead uses long 3D spatio-temporal patches which are motion compensated, producing highly temporally consistent results. VNLnet applies a CNN to a stack of frames. The frames are all registered using block matching with a large block size. It produces good results with high temporal consistency except when the block matching fails to find good matches.

Since it is implemented on GPU it runs much faster than VNLB and SPTWO, although it also has a high computational cost. The computational cost of each variant is shown in Table 6.

VBM3D combined with spatio-temporal patches and a patch-search guided with an optical flow gives very competitive results even compared to the latest state of the art, and especially for higher noises. When combined with the other improvements, it seems that multiscaling does not increase the quality of the denoising and can even degrade it in terms of PSNR. Visual examples are shown in Figure 7, where we compare results obtained with different denoising methods.

In conclusion the proposed implementation of VBM3D and its variants, provide a video denoising framework with an interesting trade-off between denoising quality and computational complexity.

σ	Method	Crowd	Park	Pedestrians	Station	Sunflower	Touchdown	Tractor	average
10	VBM3D [7]	35.65	34.75	40.83	38.93	40.49	39.04	37.01	38.10
	BM4D [16]	35.84	34.45	41.15	40.23	40.97	39.78	37.33	38.54
	VBM4D [15]	36.05	35.31	40.61	40.85	41.88	39.79	37.73	38.88
	SPTWO [6]	36.57	35.87	41.02	41.24	42.84	40.45	38.92	39.56
	VNLnet [10]	37.00	36.39	41.96	**42.44**	**43.76**	41.05	38.89	40.21
	VNLB [2]	**37.24**	**36.48**	**42.23**	42.14	43.70	**41.23**	**40.20**	**40.57**
	VBM3D ST+OF+MS	35.72	35.01	41.05	40.34	41.88	39.97	38.66	38.95
20	VBM3D [7]	32.25	31.25	36.94	35.45	36.46	36.08	33.07	34.50
	BM4D [16]	32.37	30.96	37.43	36.71	37.13	36.54	33.53	34.95
	VBM4D [15]	32.40	31.60	36.72	36.84	37.78	36.44	33.95	35.10
	SPTWO [6]	32.94	32.35	37.01	38.09	38.83	**37.55**	35.15	35.99
	VNLnet [10]	33.40	**32.94**	38.32	38.49	**39.88**	37.11	35.23	36.47
	VNLB [2]	**33.49**	32.80	**38.61**	**38.78**	39.82	37.47	**36.67**	**36.81**
	VBM3D ST+OF+MS	32.46	31.68	37.74	36.99	38.45	37.16	35.18	35.67
40	VBM3D [7]	28.65	27.68	32.81	32.02	32.65	33.52	29.41	30.96
	BM4D [16]	29.10	27.82	33.44	32.98	33.06	33.68	29.84	31.42
	VBM4D [15]	28.72	27.99	32.62	32.93	33.66	33.68	30.20	31.40
	SPTWO [6]	29.02	28.79	31.32	32.37	32.61	31.80	30.61	30.93
	VNLnet [10]	29.69	**28.29**	34.21	33.96	35.12	33.88	31.41	32.51
	VNLB [2]	**29.88**	29.28	**34.68**	**34.65**	**35.44**	34.18	**32.58**	**32.95**
	VBM3D ST+OF+MS	29.30	28.51	34.46	33.70	35.06	**34.50**	31.56	32.44

Table 5: Quantitative denoising results (PSNR and SSIM) for seven grayscale test sequences of size 960 × 540 from the *Derf's Test Media collection* on several state-of-the-art video denoising algorithms. We highlighted the best performance in black and the second best in brown.

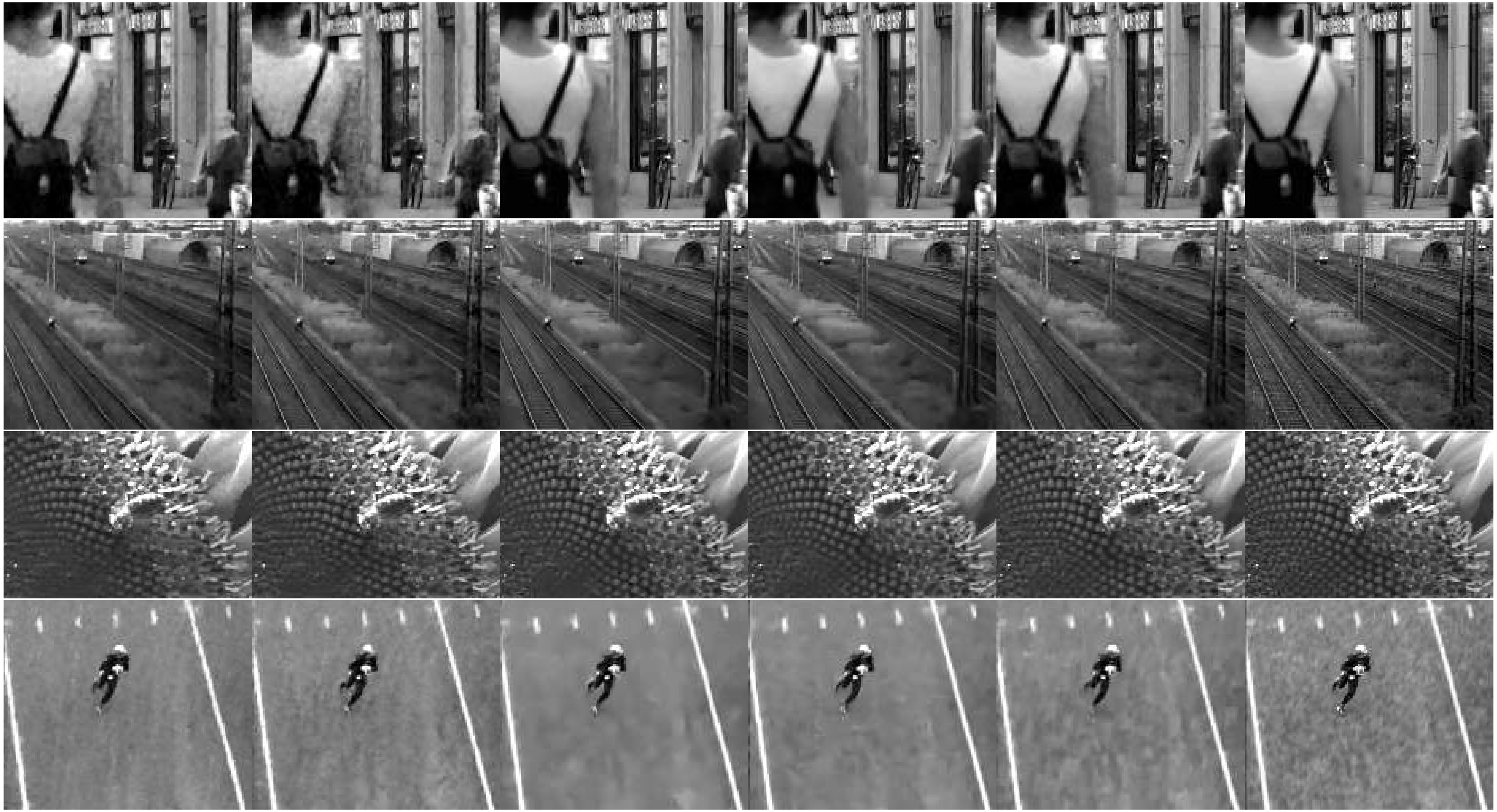

Figure 7: Comparison with state-of-the-art video denoising methods ($\sigma = 40$). From left to right: result of VBM3D [7], VBM4D [15], VNLB [2], VNLnet [10], VBM3D SF+OF+MS (Lanczos) and ground truth. Contrast has been linearly scaled for better visualization.

Default	OF	ST	MS	ST+OF	ST+MS	OF+MS	ST+OF+MS
175	245	295	269	351	390	351	481

Table 6: Computation time for each combination of the proposed variants of VBM3D (in seconds). The 'Crowd' sequence was used for the estimation of the computation time.

Acknowledgment

Work partly financed by IDEX Paris-Saclay IDI 2016, ANR-11-IDEX-0003-02, Office of Naval research grant N00014-17-1-2552, DGA Astrid project «filmer la Terre» n^o ANR-17-ASTR-0013-01, MENRT.

Image Credits

The Derf's Test Media collection[5].

References

[1] P. Arias, G. Facciolo, and J-M. Morel, *A comparison of patch-based models in video denoising*, in Image, Video, and Multidimensional Signal Processing Workshop (IVMSP), June 2018, pp. 1–5. https://doi.org/10.1109/IVMSPW.2018.8448824.

[2] P. Arias and J-M. Morel, *Video Denoising via Empirical Bayesian Estimation of Space-Time Patches*, Journal of Mathematical Imaging and Vision, 60 (2018), pp. 70–93. https://doi.org/10.1007/s10851-017-0742-4.

[3] P. Arias and J-M. Morel, *Kalman filtering of patches for frame recursive video denoising*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops (NTIRE), 2019. https://doi.org/10.1109/CVPRW.2019.00243.

[4] J.C. Brailean and A.K. Katsaggelos, *Simultaneous recursive displacement estimation and restoration of noisy-blurred image sequences*, IEEE Transactions on Image Processing, 4 (1995), pp. 1236–1251. https://doi.org/10.1109/83.413168.

[5] A. Buades, B. Coll, and J-M. Morel, *Denoising image sequences does not require motion estimation*, in IEEE Conference on Advanced Video and Signal Based Surveillance, 2005, pp. 70–74. https://doi.org/10.1109/AVSS.2005.1577245.

[6] A. Buades, J-L. Lisani, and M. Miladinović, *Patch-based video denoising with optical flow estimation*, IEEE Transactions on Image Processing, 25 (2016), pp. 2573–2586. https://doi.org/10.1109/TIP.2016.2551639.

[7] K. Dabov, A. Foi, and K. Egiazarian, *Video denoising by sparse 3D transform-domain collaborative filtering*, in European Signal Processing Conference (EUSIPCO), 2007.

[5] https://media.xiph.org/video/derf/

[8] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, *Color image denoising via sparse 3D collaborative filtering with grouping constraint in luminance-chrominance space*, in IEEE International Conference on Image Processing (ICIP), 2007. https://doi.org/10.1109/ICIP.2007.4378954.

[9] ——, *Image denoising by sparse 3D transform-domain collaborative filtering*, IEEE Transactions on Image Processing, 16 (2007), pp. 2080–2095. https://doi.org/10.1109/TIP.2007.901238.

[10] Axel Davy, Thibaud Ehret, Jean-Michel Morel, Pablo Arias, and Gabriele Facciolo, *Video denoising by combining patch search and CNNs*, Journal of Mathematical Imaging and Vision, 63 (2021), pp. 73–88.

[11] T. Ehret, J-M. Morel, and P. Arias, *Non-Local Kalman: A Recursive Video Denoising Algorithm*, in IEEE International Conference on Image Processing (ICIP), IEEE, 2018, pp. 3204–3208. https://doi.org/10.1109/ICIP.2018.8451421.

[12] G. Facciolo, N. Pierazzo, and J-M. Morel, *Conservative Scale Recomposition for Multiscale Denoising (The Devil is in the High Frequency Detail)*, SIAM Journal on Imaging Sciences, 10 (2017), pp. 1603–1626. https://doi.org/10.1137/17M1111826.

[13] O. G. Guleryuz, *Weighted overcomplete denoising*, in Asilomar Conference on Signals, Systems Computers, vol. 2, Nov 2003, pp. 1992–1996 Vol.2. https://doi.org/10.1109/ACSSC.2003.1292330.

[14] C. Liu and W.T. Freeman, *A high-quality video denoising algorithm based on reliable motion estimation*, in European Conference on Computer Vision (ECCV), 2010, pp. 706–719. https://doi.org/10.1007/978-3-642-15558-1_51.

[15] M. Maggioni, G. Boracchi, A. Foi, and K. Egiazarian, *Video denoising, deblocking, and enhancement through separable 4-D nonlocal spatiotemporal transforms*, IEEE Transactions on Image Processing, (2012). https://doi.org/10.1109/TIP.2012.2199324.

[16] M. Maggioni, V. Katkovnok, K. Egiazarian, and A. Foi, *Nonlocal transform-domain filter for volumetric data denoising and reconstruction*, IEEE Transactions on Image Processing, 22 (2013), pp. 119–133. https://doi.org/10.1109/TIP.2012.2210725.

[17] V. Papyan and M. Elad, *Multi-scale patch-based image restoration*, IEEE Transactions on Image Processing, 25 (2016), pp. 249–261. https://doi.org/10.1109/TIP.2015.2499698.

[18] N. Pierazzo, J-M. Morel, and G. Facciolo, *Multi-Scale DCT Denoising*, Image Processing On Line, 7 (2017), pp. 288–308. https://doi.org/10.5201/ipol.2017.201.

[19] M. Protter and M. Elad, *Image sequence denoising via sparse and redundant representations*, IEEE Transactions on Image Processing, 18 (2009), pp. 27–35. https://doi.org/10.1109/TIP.2008.2008065.

[20] J. Sánchez Pérez, E. Meinhardt-Llopis, and G. Facciolo, *TV-L1 Optical Flow Estimation*, Image Processing On Line, 3 (2013), pp. 137–150. https://doi.org/10.5201/ipol.2013.26.

[21] D. Schumacher, *General filtered image rescaling*, in Graphics Gems III (IBM Version), D. Kirk, ed., Morgan Kaufmann, San Francisco, 1992, pp. 8 – 16. https://doi.org/10.1016/B978-0-08-050755-2.50012-9.

[22] C. Zach, T. Pock, and H. Bischof, *A duality based approach for realtime TV-L 1 optical flow*, in Joint Pattern Recognition Symposium, Springer, 2007. https://doi.org/10.1007/978-3-540-74936-3_22.

[23] D. Zoran and Y. Weiss, *From learning models of natural image patches to whole image restoration*, in IEEE International Conference on Computer Vision (ICCV), Nov 2011, pp. 479–486. https://doi.org/10.1109/ICCV.2011.6126278.