1 BM4D GPU

BM4D is an extension of the BM3D denoising algorithm to volumetric images. It achieves state-of-the-art denoising both in terms of peak signal-to-noise ratio (PSNR) and subjective visual quality. The filter leverages sparse representation and a collaborative filtering paradigm where mutually similar three dimensional cubes are gathered together in a group on which a 4D transform is performed. Thresholding of the coefficients in the transform domain allows to reduce noise and enhance similarities inside one group. Following that, the inverse transform is performed and denoised cubes are placed back in space using specific aggregation procedure. This yield a denoised volume after the first step of the filter. The following steps are repeated for the second time using the denoised volume, but instead of thresholding in transformed domain the Wiener filtering is used which allows to improve the denoising results. In BM4D authors used 3D cubes of size 4x4x4 pixels in order to be able to reuse algorithm parameters from BM3D, which used 8x8 pixels 2D patches. In our work we introduce an implementation of the first step of the BM4D on a GPU. However, development of the second step of the algorithm can be a possible improvement to the current implementation.

1.1 Algorithm outline

BM4D consists of two main steps: Hard-thresholding and Wiener filtering. Each step can be further subdivided into approximately five smaller steps, which are:

- Cube-matching
- 4-D transform
- Thresholding/Wiener filtering
- Inverse 4-D transform
- Aggregation

The flow-diagram depicting all of the steps can be seen in Figure 1.

The most computationally intensive step is Cube-matching, but at the same time it is inherently parallel. It is carried out by computing distances between a reference 3D cube and all possible 3D cubes in its surrounding limited by the window size. The cubes are arranged in a group in the order of increasing distances where the reference cube will always be the first since it is the most similar to itself. The distance between cubes should satisfy the user-defined threshold (default 2500 for 8 bit images), otherwise a cube with higher distance from the reference cube will not be added to the group. The maximum size of the group is also limited and is equal to 16 in our implementation. On each cube a 3D Discrete Cosine Transform (DCT) is performed after which a Walsh-Hadamard transform is applied along 4-th dimension in a group. In order to improve performance, as the reference cubes are chosen only those that are 3 pixels apart (later denoted as step 3) from each other (1).

In Lebrun (2012) authors investigated different combinations of transforms and it has been shown that different transforms for each element of the group and between group elements generally produces better results. In our implementation we use combination of 3D DCT and Walsh-Hadamard transform in 4-th dimension. According to Lebrun (2012) using a bi-orthogonal spline wavelet, where the vanishing moments of the decomposing and reconstructing wavelet functions are 1 and 5 respectively (denoted as Bior1.5 Wavelets) instead of DCT might give slightly better results. This finding is entirely empirical and there is no known to us theoretical analysis of why a certain combinations of transforms perform better. We also additionally caried out an analysis of the performance of 3D Haar Wavelets for our implementation of the BM3D algorithm. The denoising procedure which used only Haar Wavelets produced the results that are inferiour to the DCT + Walsh-Hadamard transform combination. For more detailes please see the Appendix and for implementation of BM3D please see digital appendix.

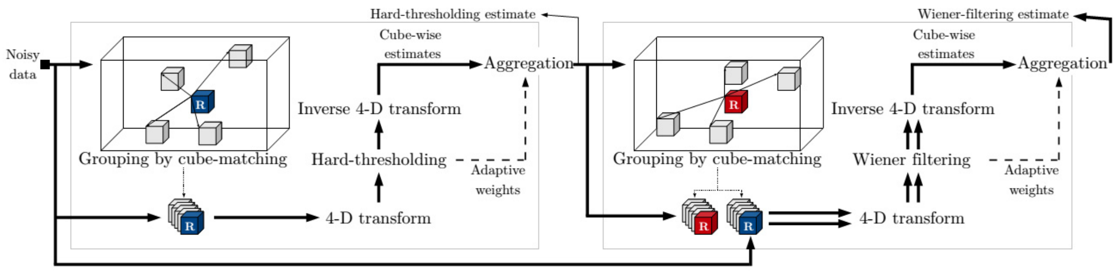

Figure 1: BM4D flow-diagram. On the left the first step (Hard-thresholding) and on the right the second step (Wiener filtering). Image source Maggioni et al. (2013)

1.1.1 Cube-matching

The first step in the algorithm is Cube-matching. It is realized by taking one cube from the volume and comparing it with all the surrounding cubes in its neighborhood. As a similarity measure between cubes the photometric distance is used,

$$d(C^z_{x_i}, C^z_{x_j}) = \frac{||C^z_{x_i} - C^z_{x_j}||^2_2}{L^3} \tag{1}$$

where C is a cube and $||\cdot||^2_2$ is a sum of squared differences between intensities of each pixel in the cube, L is the width of the cube. If the distances between two cubes are smaller than a certain threshold, they are added to one group. The value recommended by Lebrun (2012) for similarity threshold is equal to 2500. A higher value of the threshold tends to give slightly better results, but it also increases processing times and incurs additional risk of adding too different cubes in same group.

The algorithm operates on 8 bit images which have integer values between 0 and 255, the default cube size is 4x4x4 pixels. Therefore, the maximum distance between two cubes is equal to,

$$d(C^z_{x_i}, C^z_{x_j}) = \frac{\sum_1^{64}(255)^2}{4^3} = \frac{4161600}{64} = 65025 \tag{2}$$

Thus, the reasonable values for similarity threshold for 8 bit image with cube size 4x4x4 pixels lays in the range [0, 65024].

The maximum number of similar cubes in a group is limited and is equal to 16 in our implementation. This value must always be a power of 2 due to Walsh-Hadamard filtering in 4-th dimension.

1.1.2 Discrete Cosine Transform

Discrete Cosine Transform (DCT) is usually applied in lossy data compression due to the fact that most of the information tends to concentrate in few low-frequency components. In BM4D GPU we use 3D DCT to transform each cube in the group independently. We first apply 2D DCT for each slice of the cube and then 1D DCT in the third dimension. This approach was chosen over three 1D transforms since it permits to use a 2D transform in one parallel operation. Ideally, it would be better to use one 3D transform directly and it can be considered as a possible improvement.

DCT in the matrix form has the following form,

$$C = A^\top F A \tag{3}$$

where A is a coefficient matrix of size 4x4 for cube with edge length 4 (Equation 4) and F is the signal (one 4x4 slice of the cube).

$$A = \begin{bmatrix} 0.5000 & 0.5000 & 0.5000 & 0.5000 \\ 0.6532 & 0.2705 & -0.2705 & -0.6532 \\ 0.5000 & -0.5000 & -0.5000 & 0.5000 \\ 0.2705 & -0.6532 & 0.6532 & -0.2705 \end{bmatrix} \tag{4}$$

The inverse transform is similar and is written in Equation 5.

$$F = A C A^\top \tag{5}$$

1.1.3 Fast Walsh-Hadamard Transform

This is a very simple 1D transform that is applied in the 4-th dimension. It is a recursive process of combining sums and differences of a given vector, where the vector length is the power of 2. The recursive basis function has the following form,

$$H_N = (H_{N-1} \quad -H_{N-1}). \tag{6}$$

A simple example of the transform for a vector of size 4 will have the following form (Lebrun (2012)),

$$V = [a\, b\, c\, d] \tag{7}$$
$$V = [(a+b)\ (c+d)\ (a-b)\ (c-d)] \tag{8}$$
$$V = [(a+b+c+d)\ (a+b-c-d)\ (a-b+c-d)\ (a-b-c-d)] \tag{9}$$

After transform is performed each coefficient should be normalized by $\sqrt{N}$, then the inverse transform is achieved by applying the forward transform again.

1.1.4 Aggregation

After all of the groups have been denoised it is necessary to place them back into the volume. However, many of them will overlap and to resolve this overlapping a special weighting technique is used. The idea is to weight each cube in the group with the group weight, which is computed as follows (Lebrun (2012)),

$$w_{ht} = \frac{1}{N_{ht}} \tag{10}$$

where N_{ht} is the number of non-zero coefficients in the group after hard thresholding. Intuitively, it means that a cube from the group that contains less noise (many non-zeroed coefficients) will contribute more to the final image. This formula differs from original paper (Maggioni et al. (2013)) where N_{ht} is also multiplied by additional hyperparameter σ. Since aggregation techniques doesn't affect result significantly (Lebrun (2012)) we use simpler form without sigma. The weighted cubes are added to the final image volume and weights are accumulated to the weight volume to be later applied in normalization (Equation 11),

$$I(x) = I(x) + \omega_n * C_x^n \tag{11}$$
$$W(x) = W(x) + \omega_n \tag{12}$$
$$I(x) = I(x)/W(x) \tag{13}$$

where $I(x)$ is the final image, ω_n is the scalar value of the weight for the group n and C_x^n is a cube from the group n spatially placed in x.

1.2 Implementation details

Default parameters used in our implementation are the recommended values from Lebrun (2012) shown in Table 1.

Table 1: BM4D GPU parameters

Parameter name	Values
Cube size	4
Group size	16
Step	3
Search-cube size	11
Similarity thr.	2500
Shrinkage thr.	2.7

Current GPU implementation is bounded by memory available on GPU. So it is necessary to resort to the overlapping strategy for longer sequences. The idea is to slice time series along the depth dimension into the overlapping chunks, Figure 2. The amount of the overlap required is dictated by the size of the neighborhood in which similar cubes are searched. In our case cube edge length is 4 and window size 11 (5 pixels from each side). Therefore, the overlap required for one side is,

$$\frac{cube\_edge\_length}{2} + window\_radius = 4/2 + 5 = 7. \tag{14}$$

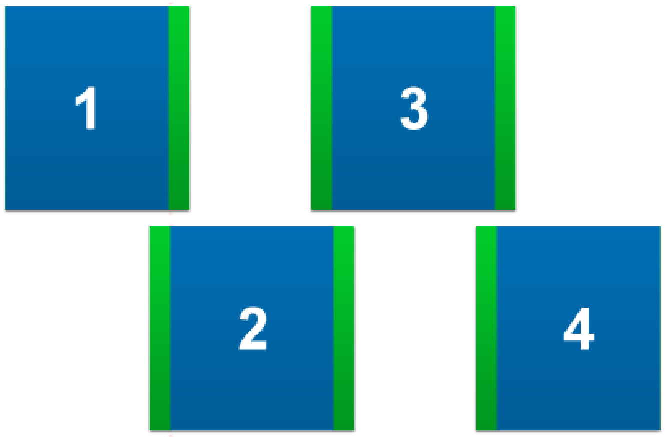

Figure 2: An overlapping strategy for denoising long sequences. The sequence is subdivided along depth dimension into chunks with overlap of 7 (green part), then blue part represents the denoised slices.

In Figure 3 it can be seen that the memory required for denoising grows linearly with the number of slices (blue line).

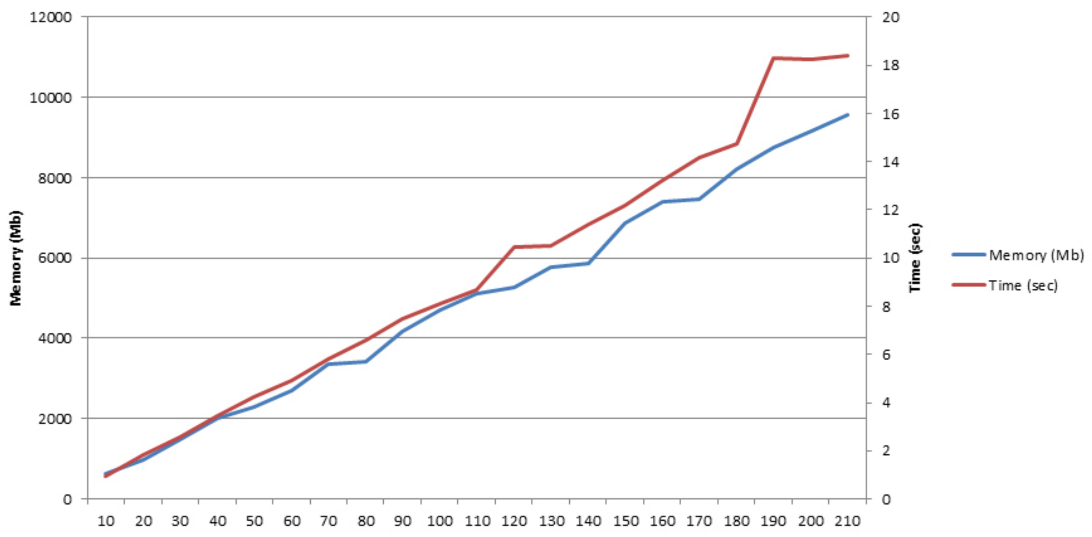

Figure 3: Memory required for computation grows linearly with respect to the input data size. X axis shows number of slices in depth dimension for a sequence of size 512x512 pixels. Y axis on the right shows memory required in Mb and Y axis on the left displays time required for processing the image volume in seconds. Blue line corresponds to the memory requirements and red to the time.

1.3 Results

In order to evaluate the results of the denoising algorithm we use the classic quantitative measure for image denoising - peak-signal-to-noise ratio (PSNR),

$$PSNR = 10 * log_{10}\left(\frac{MAX\_VALUE}{\sqrt{MSE}}\right). \tag{15}$$

It provides the ratio between the maximum possible value of a signal and the power of distorting noise that affects the quality of its representation. It is expressed in terms of the logarithmic decibel scale. In Equation 15 and 16 MSE stands for the Mean Squared Error and it is computed between the ideal image (ground truth) and the noisy or denoised image. Intuitively, the higher value the better, and the PSNR of the image to itself is the infinity.

$$MSE = \frac{1}{N}\sum_{x\in X}^{N}(u_g(x) - u_d(x))^2 \tag{16}$$

Therefore, to understand the relative improvement of the denoising algorithm we provide the baseline which is the PSNR between ground truth and the noisy image.

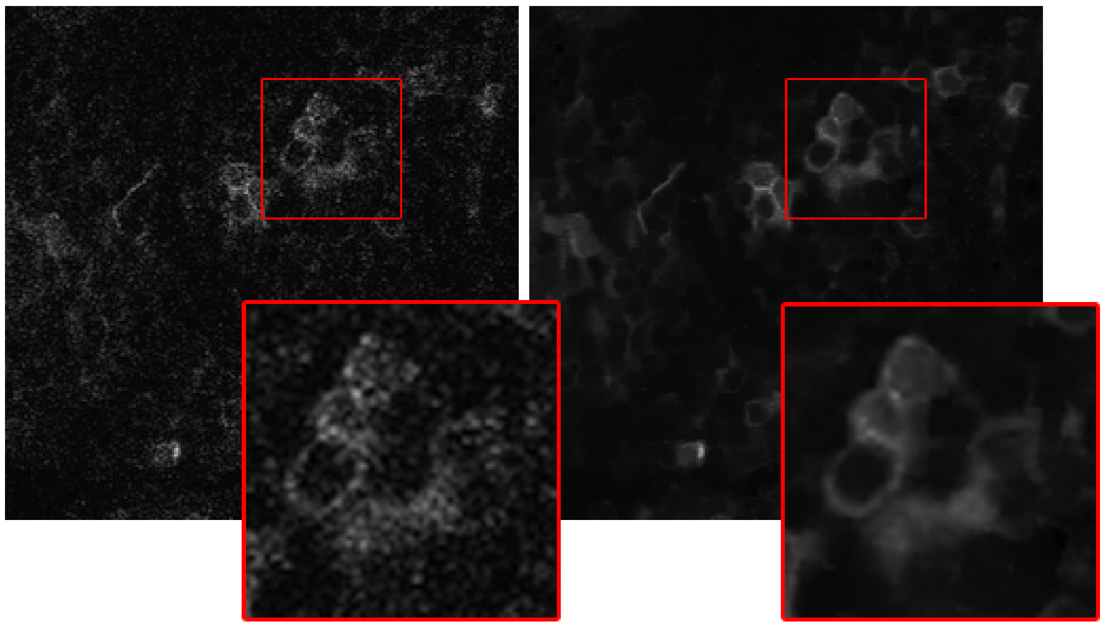

Figure 4: Noisy and denoised rec62. Due to the lack of ground truth, the mean image is used for computing the PSNR. PSNR noisy 29.262, PSNR denoised 38.4568.

Our implementation of BM4D takes 8 bit images as input. This representation uses integer values in the range from 0 to 255 as intensity. Therefore, in our case MAX_VALUE is equal to 255.

To compare how much faster the GPU implementation was than the original BM4D implementation, we tested in performance on three videos (“Tennis”, “Flowers”, “rec62”, see digital appendix), the last one of which was a calcium image time series of the larval zebrafish pretectum recorded at 2 fps in the Driever laboratory.

Table 2 presents comparison of the BM4D CPU and out BM4D GPU in terms of runtime and PSNR measure. The possible difference in the PSNR between our and original implementation can be due to the different aggregation weighting schema. In the latest version of BM4D authors introduced automatic evaluation of σ parameter while we always assume this parameter to be 1.

Table 2: BM4D Results.

Name	Size	PSNR (dB)			Time (sec)		Speedup
		Noisy	CPU	GPU	CPU	GPU	
Tennis	352x240x150	24.7890	30.6367	31.1713	461.3	3.983	115.8x
Flowers	352x240x150	24.8250	24.2875	27.7999	384.9	3.304	116.5x
rec62	512x512x897	29.2622	38.9373	38.4568	7137.1	98.21	72.7x

On the small sequences like Tennis and Flowers, when the whole sequence fit inside GPU memory, this speedup is about factor of 115 and for rec62, which is too big to fit in memory fully, the achieved speedup is 72x. The rec62, therefore, has been computed using overlapping strategy with the depth dimension equal to 200. All presented results were computed on Nvidia GeForce GTX TITAN X. A qualitative example can be seen in Figures 4 and 5.

1.4 Considerations for improvements

The most important improvement that can be made to the filter is to resolve the memory bottleneck. One possible solution is to re-implement existing slicing scheme in C++ which will already speed things up due to removed reads and writes. Apart from that, the algorithm is limited to the cube size 4 by DCT kernels, since coefficient matrices are of size 4x4 and hard-coded. It could be also worth adding matrix of size 8x8 since DCT is known to perform the best at this size of patches. Additionally, maximum group size is limited by 16 in Walsh-Hadamard kernel and can be extended to 32 or more.

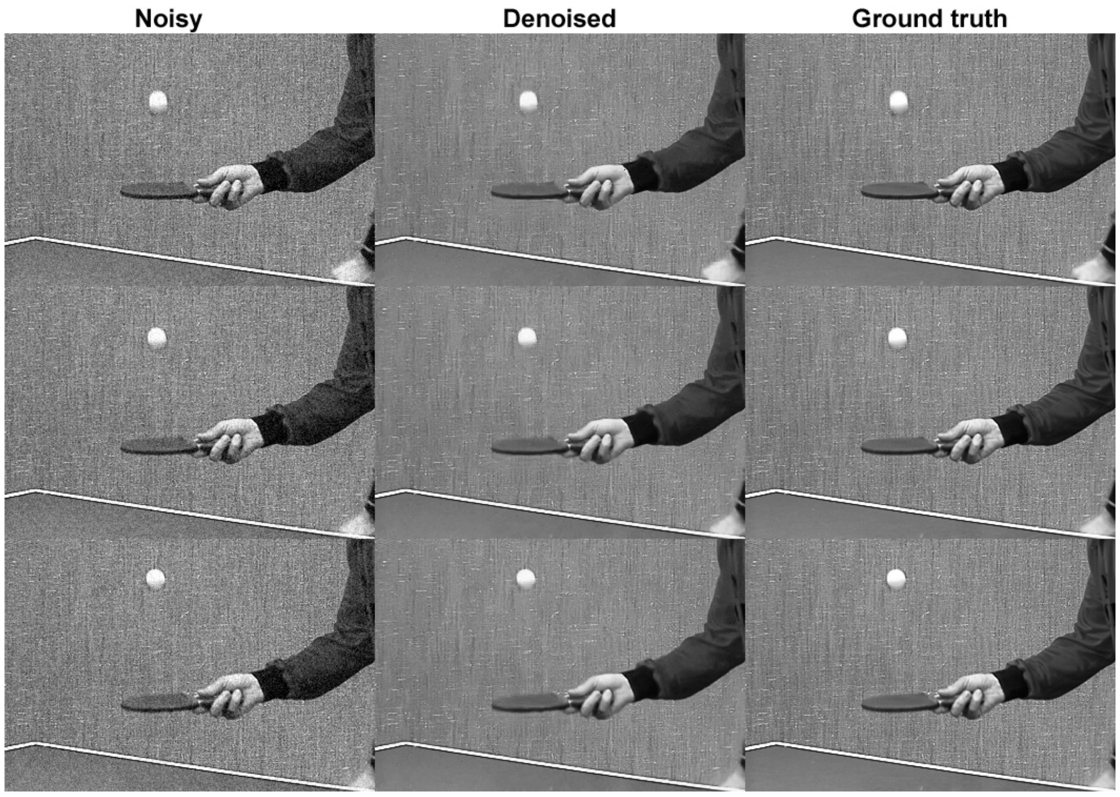

Figure 5: Comparison of BM4D on the Tennis sequence. Images were corrupted with Gaussian noise $\sigma = 15$. PSNR noisy 24.7253, PSNR denoised 31.1713.

References

Lebrun, M. (2012). An Analysis and Implementation of the BM3D Image Denoising Method. *Image Processing On Line*, 2:175–213.

Maggioni, M., Katkovnik, V., Egiazarian, K., and Foi, A. (2013). Nonlocal transform-domain filter for volumetric data denoising and reconstruction. *IEEE transactions on image processing*, 22(1):119–133.

Appendix

Case study of BM3D

To understand how BM4D works we first implemented BM3D CPU version of the algorithm. We supply the source code for this algorithm in the digital appendix. The whole algorithm is in single file that was tested on Linux.

Additionally, we have investigated how a 3D Haar Wavelets are working and implemented three 1D Haar Wavelet transforms on the group. In Figure 6 on the left you can see the test image ”Barbara” that was corrupted by Gaussian noise with $\sigma = 15$, and on the right is the denoised image with our implementation of BM4D CPU using 3D Haar Wavelets with hard shrinkage. The images that were denoised with Haar Wavelets used the following parameters: similarity threshold 5500, hard threshold 30 and other parameters equal to the default values (please see the provided code for the details). It can be seen that even though the PSNR and visual appearance of the image is generally improved with Haar Wavelets using hard shrinkage or Garotte shrinkage, they are both inferior to the implementation with Discrete Cosine Transform and Walsh-Hadamard transform.

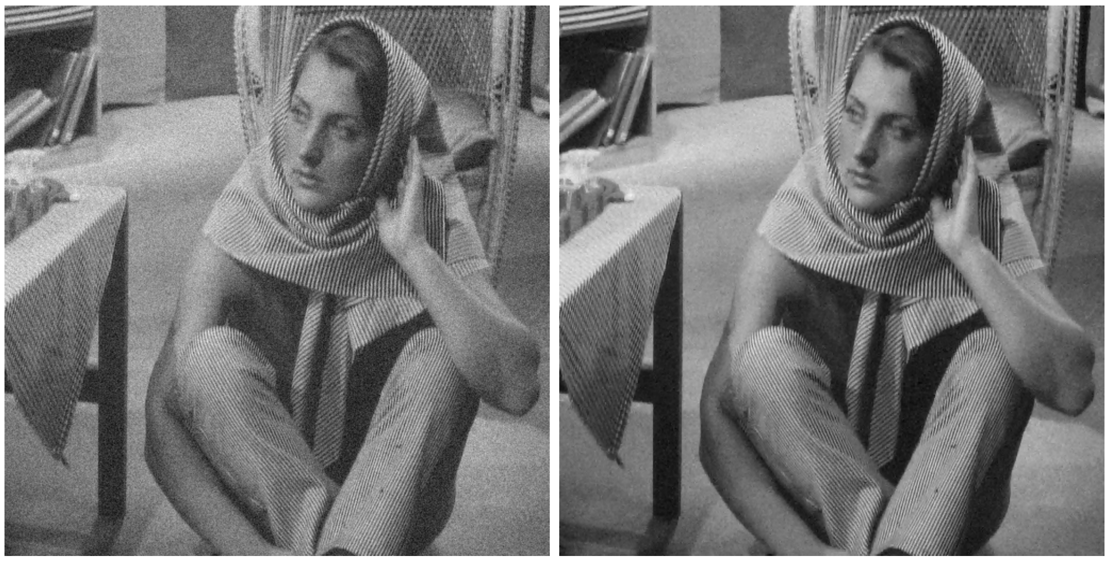
Figure 6: Denoised image barbara.ppm with BM3D using 3D Haar Wavelet + Hard shrinkage. PSNR noisy 19.668, PSNR denoised 20.422.

The 1D Haar Wavelet has the following form,

$$\phi(x) = \phi 2x + \phi(2x - 1) \tag{17}$$

where x is an index of the element in 1D vector. In our implementation we applied transform first along x dimension, then y and z. The inverse transform should follow the inverse order.

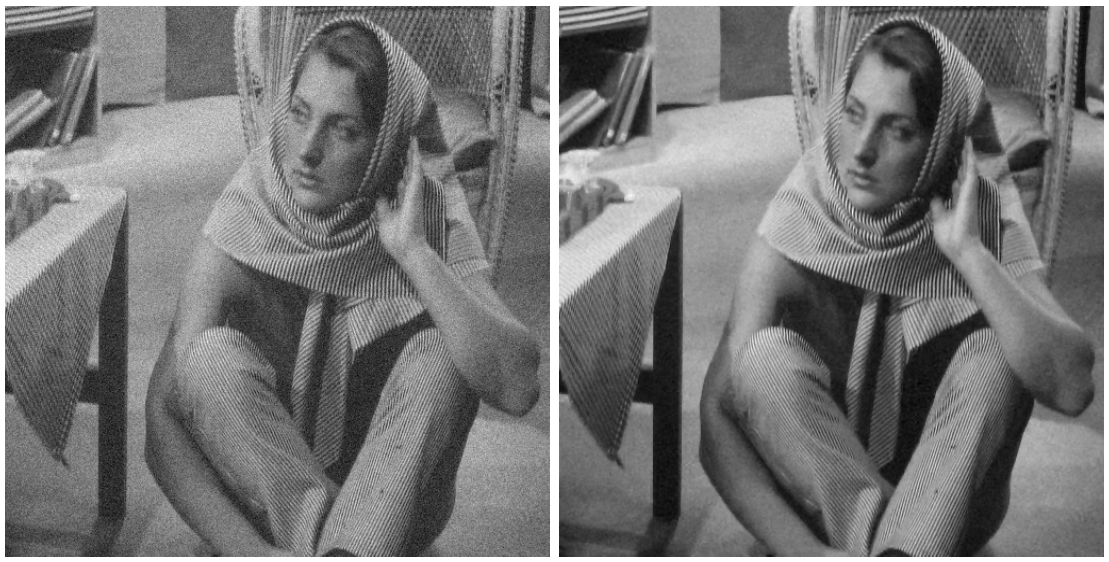
Figure 7: Denoised image barbara.ppm with BM3D using 3D Haar Wavelet + Garotte shrinkage. PSNR noisy 19.668, PSNR denoised 20.332.

We used two schemes for thresholding which in case of applying Haar Wavelets conventionally is called shrinkage. In Equation 18 you can see the hard shrinkage and in Equation 19 the Garrotte shrinkage is shown. The latter gives more visually plausible results, although in our case the PSNR score has also dropped.

$$\sigma^{hard}(x) = \begin{cases} x, |x| > \lambda \\ 0, otherwise \end{cases} \tag{18}$$

$$\sigma^{garotte}(x) = \begin{cases} x - \lambda^2/x, |x| > \lambda \\ 0, otherwise \end{cases} \tag{19}$$

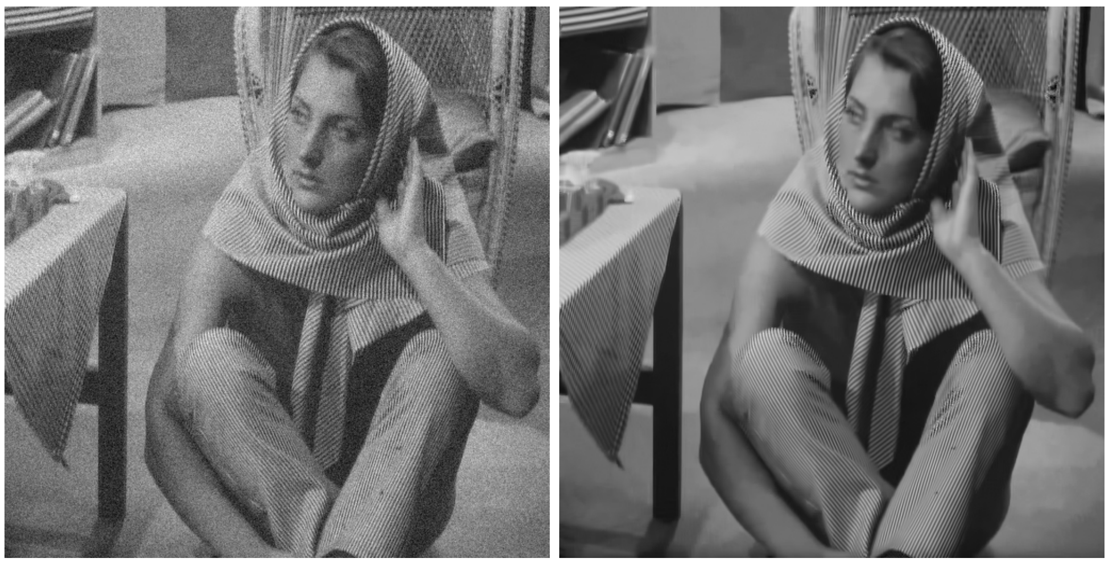
Figure 8: Denoised image barbara.ppm with BM3D using DCT + Walsh-Hadamard transform. PSNR noisy 19.668, PSNR denoised 20.506.