

JULIÁN LAMAS-RODRÍGUEZ, MORITZ EHLKE,
RENÉ HOFFMANN, STEFAN ZACHOW

**GPU-accelerated denoising
of large tomographic data sets
with low SNR**

**Application for non-invasive analysis
of paleontological data**

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

e-mail: bibliothek@zib.de
URL: <http://www.zib.de>

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

GPU-accelerated denoising
of large tomographic data sets
with low SNR

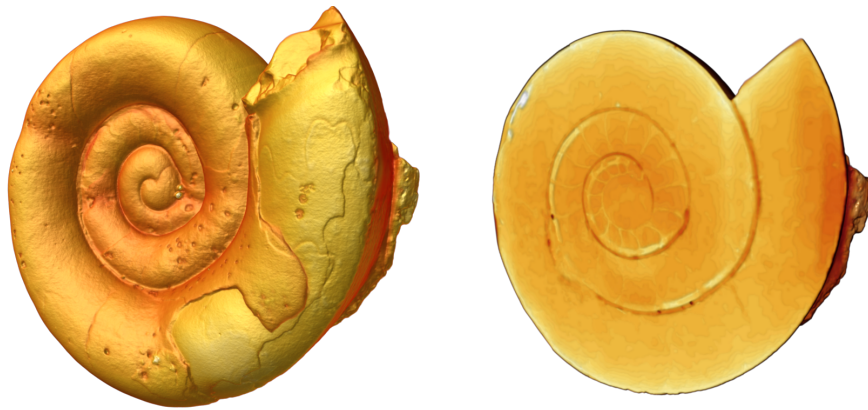
Application for non-invasive analysis
of paleontological data

Julián Lamas-Rodríguez¹, Moritz Ehlke¹, René Hoffmann², and
Stefan Zachow¹

¹Zuse Institute Berlin (ZIB), Department of Visual Data Analysis

²Ruhr-Universität Bochum, Institute of Geology, Mineralogy, and
Geophysics

December 4, 2015



Contents

1	Motivation	3
2	Foundation of this work	6
3	Materials and methods	8
3.1	Evaluation of different denoising methods	8
3.1.1	Denoising an artificial dataset	8
3.1.2	Denoising a paleontological dataset	14
4	GPU implementation of an improved AM-NLM	18
4.1	Filter implementation details	18
4.2	Memory optimization	20
4.3	Performance optimization	22
5	AM-NLM-based denoising in ZIBAmira	23
6	Conclusions and future work	31
A	Notation used in this document	32
B	Mathematical formulation of denoising filters	32
B.1	Gaussian smoothing	32
B.2	Median filter	33
B.3	Anisotropic diffusion	33
B.4	Bilateral filtering	33
B.5	Non-local means	34
C	Signal-processing approaches to bilateral filtering	34
C.1	Linearization of the bilateral filter	34
C.2	Bilateral grid	36
C.3	Gaussian kd-tree	37
C.4	Adaptive manifolds	39
D	Evaluation results	41
D.1	Gaussian smoothing	41
D.2	2D NLM	44
D.3	3D NLM	50
E	Gaussian kd-tree NLM	54
E.1	AM-NLM	58
E.2	k-means clustering	61

Abstract

Enhancements in tomographic imaging techniques facilitate non-destructive methods for visualizing fossil structures. However, to penetrate dense materials such as sediments or pyrites, image acquisition is typically performed with high beam energy and very sensitive image intensifiers, leading to artifacts and noise in the acquired data. The analysis of delicate fossil structures requires the images to be captured in maximum resolution, resulting in large data sets of several giga bytes (GB) in size. Since the structural information of interest is often almost in the same spatial range as artifacts and noise, image processing and segmentation algorithms have to cope with a very low signal-to-noise ratio (SNR).

Within this report we present a study on the performance of a collection of denoising algorithms applied to a very noisy fossil dataset. The study shows that a non-local means (NLM) filter, in case it is properly configured, is able to remove a considerable amount of noise while preserving most of the structural information of interest. Based on the results of this study, we developed a software tool within ZIBAmira that denoises large tomographic datasets using an adaptive, GPU-accelerated NLM filter. With the help of our implementation a user can interactively configure the filter’s parameters and thus its effectiveness with respect to the data of interest, while the filtering response is instantly visualized for a pre-selected region of interest (ROI). Our implementation efficiently denoises even large fossil datasets in a reasonable amount of time.

1 Motivation

The non-invasive analysis of fossil structures, either fully or partially preserved within sedimentary rocks, requires imaging methods that are primarily used in non-destructive material testing. Current state-of-the-art tomographic imaging techniques such as high resolution computed tomography (CT) or micro CT (μ CT), enable a view inside of solid objects with astonishing spatial resolution [9, 20]. The geometric reconstruction of relevant structures from such tomographic measurements typically involves the task of image segmentation. This includes the classification and delineation of structures within the image data. Segmentation works best if structures of interest have a distinct and unique intensity range, and are homogeneous and considerably larger than the image resolution itself (Fig. 1).

However, delineation of structures is often limited by imaging artifacts and sensor noise. Artifacts might, for instance, result from total beam blockage due to extremely dense (i.e., radio-opaque) embedding rock material or sediment filling of fossils. To cope with beam-blockage, the image intensifiers of the tomography scanner are typically operated in a highly sensitive mode [4, 17]. While this allows to capture delicate structures with thickness close to the size of sensor elements (pixels), it also induces additional noise in the images. The tomographic data, thus, often exhibits a very low signal-to-noise ratio (SNR), hampering the segmentation and visualization of small fossil structures for further analysis (cf. Fig. 2).

The aim of this work is to denoise tomographic images for non-invasive analysis of palaeontological data without degrading relevant image information. We are particularly interested in the shape and dimension of complexly folded septal surfaces and delicate shell ornamentation of fossil cephalopods. Respective μ CT datasets are large in size (35 up to 100 GB) and exhibit a low SNR. Since existing filters did not meet our requirements, appropriate tools and algorithms have been developed (i) to cope with the size of the data sets, (ii) to enable a user to interactively determine suitable filtering parameters, and (iii) to efficiently process the 3D image data in a reasonable amount of time. Development and data analysis have been performed within the software platform ZIBAmira in its current version [21].

Structure of this document

The remainder of this manuscript is structured as follows: First, the general advantages and shortcomings of available filters are briefly discussed based on related literature. We then evaluate existing filters w.r.t. to our application in Chapter 3. In Chapter 4, a GPU-based implementation of a Non-Local-Means (NLM) filter is presented, that denoises the given datasets while preserving structures of interest. In order to ensure an adequate filtering response, the filter is configured prior to each denoising pass using a set of filter parameters. We shortly describe an interactive ZIBAmira tool that has been developed for this purpose in Chapter 5 and conclude our findings in Chapter 6. Mathematical details of the investigated filtering algorithms and additional figures are given in the Appendices.

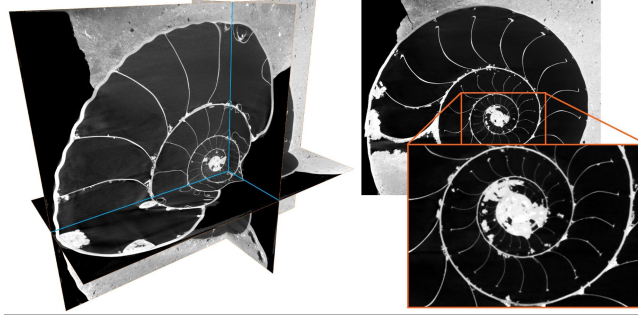


Figure 1: Micro-CT dataset of the Jurassic ammonite *Quenstedtoceras* sp. ($3650 \times 3350 \times 2400$ voxels, 55 GB). Left: Multiplanar visualization showing an ammonite shell preserved in the rare hollow condition and surrounding rock material. Right: Planar image of the same specimen depicting the fragile, internal septa subdividing the shell into discrete chambers which are nicely visible due to the unique preservation.

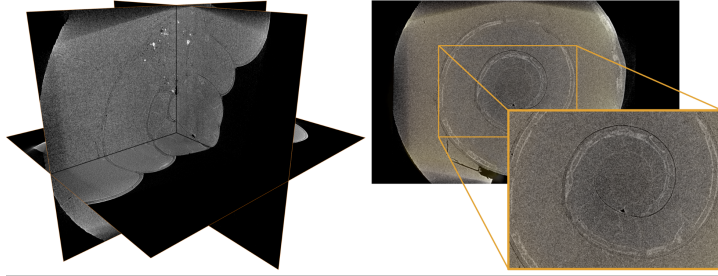


Figure 2: Micro-CT dataset of the Lower Jurassic ammonite *Eleganticeras* sp. ($3000 \times 2700 \times 2400$ voxels, 36 GB). Left: Multiplanar visualization showing the ammonite shell filled completely with former sediment that was turned into sedimentary rock over the last 110 million years. Right: Planar image of the same specimen depicting the low SNR due to sedimentary infill of the ammonite shell with calcareous rocky material that share similar absorption properties like the fragile septa subdividing the shell into separate chambers.

2 Foundation of this work

In this section we provide an overview of commonly applied image filters and shortly review their concepts, strengths and weaknesses. For a mathematical description of each of these filters, we refer the reader to Appendix B.

Gaussian smoothing This is one of the most basic forms of noise reduction.

A Gaussian filter is fast and easy to implement; it replaces each pixel by a weighted average of all pixels in its local neighborhood. The Gaussian smoothing is effective in removing high-frequency components of an image. Unfortunately this means that the filter removes not only noise but also relevant information which is often present in areas with a high image gradients, i.e., edges of objects.

Median filter The median filter replaces the color/intensity of each pixel by the median color/intensity of all pixels in a local neighborhood [13]. Such a filter is also fast and easy to implement. A median filter has the advantage of not introducing values that are not present in the original image, however, it is not efficient in preserving edges in very noisy images.

Anisotropic diffusion An iterative algorithm that generates a succession of more and more blurred images based on a diffusion process [15]. The rate of diffusion determines the degree of smoothing or blurring across edges. Anisotropic diffusion is preserving edges (provided an appropriate rate of diffusion is used), but as other local filters, it performs poorly on homogeneously textured regions [2].

Bilateral filter A generalization of the Gaussian smoothing that operates both in the spatial and in the range domain of an image [19]. Each pixel is also replaced by a weighted average of surrounding pixels, but in this case the weights not only depend on the Euclidean distance of pixels, but also on the radiometric differences (color, intensity, etc). The bilateral filter preserves sharp edges, but at the same time introduces staircase effects and false edges [2].

Non-local means By introducing the concept of neighborhoods, an NLM filter results in a generalization of the bilateral filter in a higher dimensional space. Instead of the radiometric differences, the weight values depend on the similarity between the areas of the image that surround the pixels, also known as neighborhoods (Fig. 3). NLM exploits the redundancy that is present in natural images, and it is able to preserve image features even in the case of high frequencies.

The performance of the NLM filter has been subject of thorough studies [1, 3, 7, 10, 12, 18]. Among other developments, the application of a signal-processing

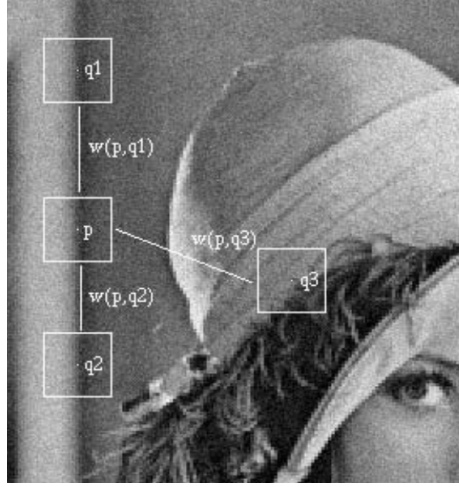


Figure 3: Denoising of pixel p in a 2D image using the NLM filter. Pixels q_1 and q_2 have a larger weight w because they are closer to p and their neighborhoods (represented as squares on the image) are similar. Pixel q_3 , however, has a smaller weight. Reprinted from Ref. [2].

paradigm to the bilateral filter [14] has proven to be key in the improvement of the efficiency of this filtering scheme and resulted in new filtering approaches based on the bilateral grid [5]. Subjects of our investigation were Gaussian *kd*-trees and adaptive-manifolds variants of NLM, which we briefly describe below. For a more in-depth review of the bilateral grid and related filters, we refer the reader to Appendix C.

Gaussian *kd*-tree Approximation of the bilateral grid that aims to reduce the number of computations by using a Monte-Carlo approach. The basic operations to perform the filtering are the same as with the bilateral grid. However, the general approach uses a *kd*-tree that sparsely represents the high-dimensional space containing all possible neighborhoods in the image as values stored in its nodes, thus, restricting the operations to the points of the image in the vicinity of those nodes.

Adaptive manifolds A similar approach to the bilateral grid that samples the high-dimensional space in a collection of manifolds, where the basic bilateral-grid operations are performed. This is the first high-dimensional filter with linear cost both in the number of pixels and in the dimension of the space in which the filter operates.

In this report we explore the parameter spaces of the aforementioned filters within the context of denoising tomographic images of fossil structures in order to become familiar with the effects these parameters do exert on the respective filtering result.

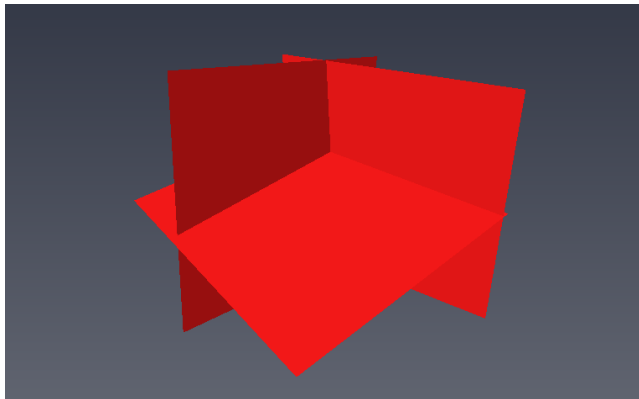


Figure 4: Test case: Three oblique intersecting (thin) planes.

3 Materials and methods

3.1 Evaluation of different denoising methods

In the following section, we present a comparative study of the performance of different denoising filters. Our goal is to analyze (i) how the filters behave under different levels of noise, e.g. how well the filters remove noise, (ii) how much of the structural information is lost during denoising, and (iii) how fast the filtering is performed. The filters are tested using an artificial dataset as well as a detail from a real, noisy tomographic image dataset. We designed the artificial data to emulate the thin septal interfaces found in typical fossil data of the application in question.

3.1.1 Denoising an artificial dataset

For our experiments, an artificial image volume, i.e., a regular voxel grid, of size 64^3 was employed. Three oblique intersecting planes (Fig. 4) were sampled into that three-dimensional scalar field in such a way that voxels being intersected by any of the planes receive a value of 1, and all unaffected voxels a value of 0. That way, the voxel grid contains a binarized representation of the three planes sampled onto a regular voxel grid (cf. Fig. 5, center).

Gaussian white noise was added to the artificial dataset, with different values of variance $\sigma^2 = \{0.01, 0.1, 1.0\}$ (Fig. 5). When the level of noise is small, the disturbance of the data is barely visually noticeable. For a fairly moderate noise level, noise will be noticed in the empty regions of the data volume and the histogram will change in a noticeable way. For a high amount of noise, the planes cannot be visually recognized. The visual effect of the different levels of noise is corroborated by the decreasing values of the peak signal-to-noise ratio (PSNR).

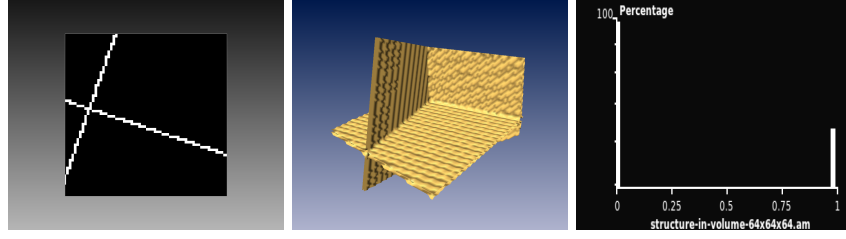
With these artificially degraded datasets, three different denoising algo-

rithms were tested: (i) a Gaussian smoothing, (ii) an implementation of a non-local means (NLM) filter, and (iii) an implementation of the adaptive manifolds, non-local means (AM-NLM) filter. For these three filters, results are demonstrated for efficient filter configurations.

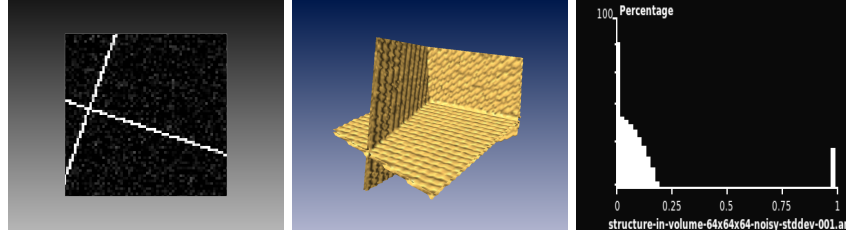
For a low level of noise, all algorithms manage to restore the original data, but the Gaussian smoothing introduces an important variation that results in a blurring of the planes (Fig. 6). As a result, the PSNR is even lower than that of the noisy volume dataset. On the other hand, the family of NLM algorithms are able to restore original structures with nearly pixel-perfect accuracy.

When filtering a fairly noisy dataset, the Gaussian smoothing removes almost all noise but introduces even more blurriness (Fig. 7). The filtered data has a PSNR similar to that of the noisy data. The NLM algorithms reduce the noise without severely degrading the structural information, leading to acceptable values of PSNR.

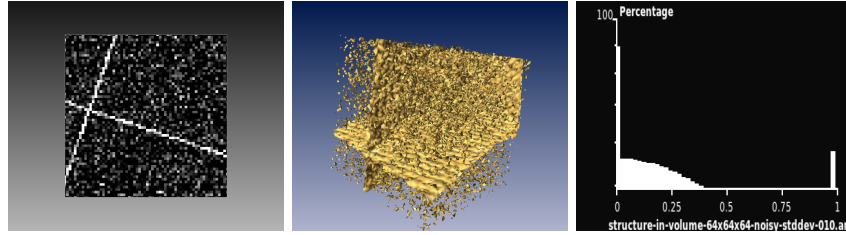
Finally, for the noisiest datasets, none of the algorithms manage to obtain an acceptable result, as most of the noise is still present (Fig. 8). NLM algorithms are at least able to restore the intersections between planes, as shown in the respective visualization of the reconstructed surfaces. Notice also how the AM-NLM interprets most of the noise in empty areas as structural information intended to be preserved.



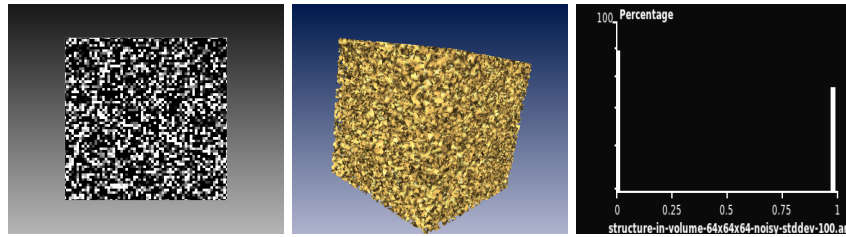
Original volume



$\sigma^2 = 0.01$, $PSNR = 22.999$

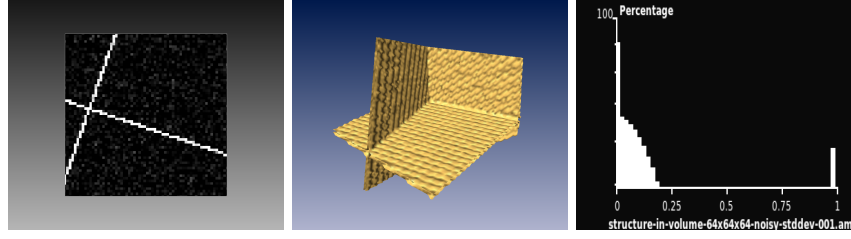


$\sigma^2 = 0.1$, $PSNR = 12.998$

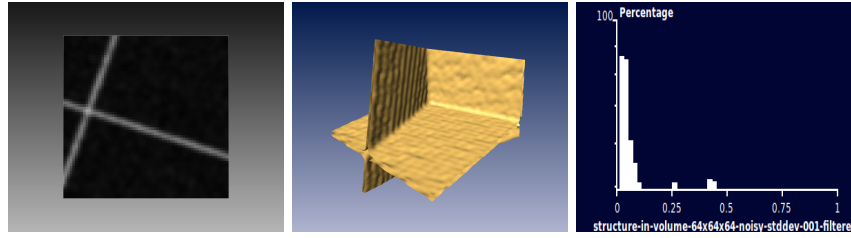


$\sigma^2 = 1.0$, $PSNR = 5.890$

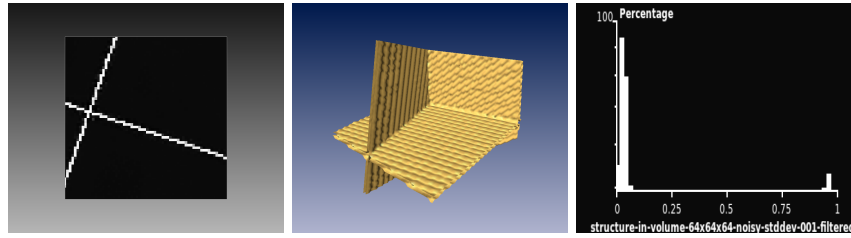
Figure 5: An artificial data volume of 64^3 voxels representing three intersecting planes. In each row, the left image shows a 2D slice of the data, the center image shows a rendering of an iso-surface for a threshold value of 0.4, and the right image contains the respective histogram of the volumetric data. The top row corresponds to the original dataset, prior to adding noise. Each row below shows the same volume with additive Gaussian noise of increasing variance (σ^2). The peak signal-to-noise ratio (PSNR) measured on the noisy volumes provides a quantitative measure of the degradation of the quality of each dataset.



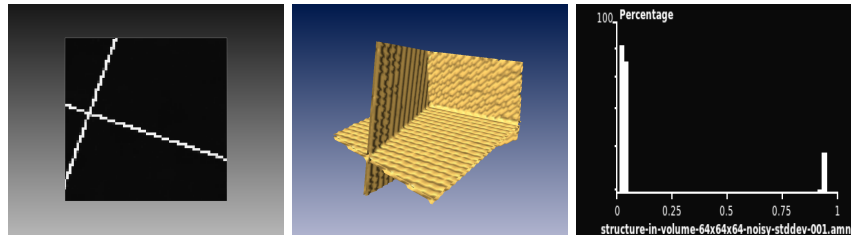
Noisy version ($\sigma^2 = 0.01$)
 $PSNR = 22.999$



Gaussian smoothing ($kernel\ size = 5^3$, $\sigma = 0.4$)
 $PSNR = 16.026$

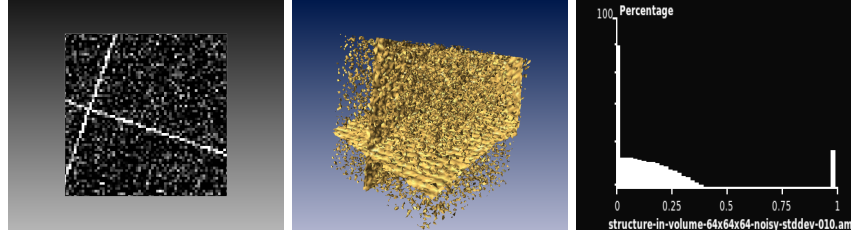


NLM ($search\ window\ size = 21^3$, $neighborhood\ size = 5^3$, $similarity = 1.0$)
 $PSNR = 26.855$

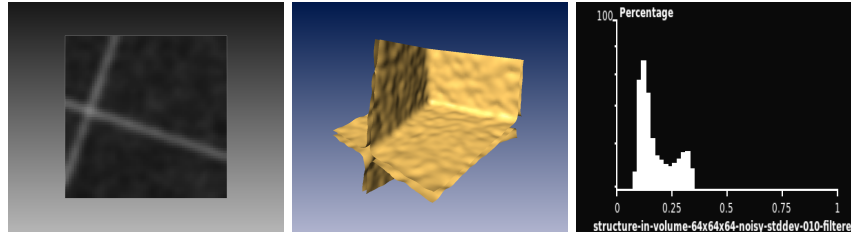


AM-NLM ($\sigma_s = 5.0$, $\sigma_r = 0.3$)
 $PSNR = 31.846$

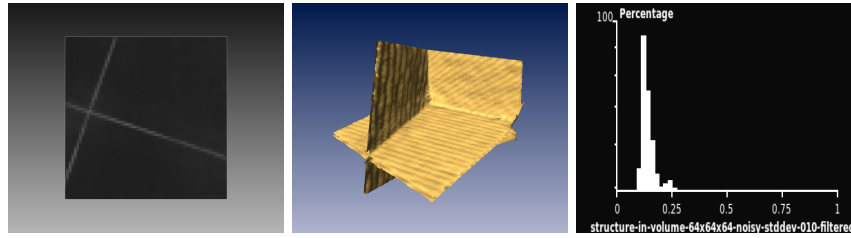
Figure 6: Denoised artificial dataset with Gaussian noise level of $\sigma^2 = 0.01$. The first row shows the noisy version of the dataset that was used as input for the different filters. The rows below show the results of filtering with Gaussian smoothing, NLM, and AM-NLM. The parameters used for each filter are denoted below each image.



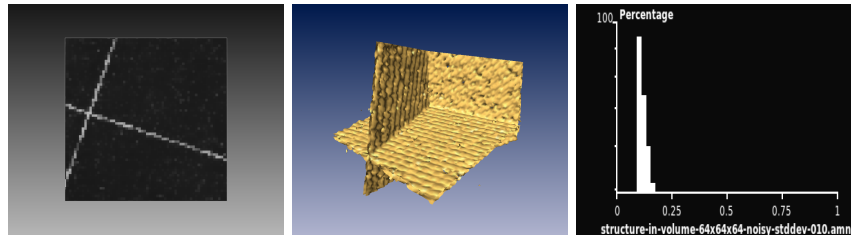
Noisy version ($\sigma^2 = 0.1$)
 $PSNR = 12.998$



Gaussian smoothing ($kernel\ size = 8^3$, $\sigma = 0.4$)
 $PSNR = 13.028$

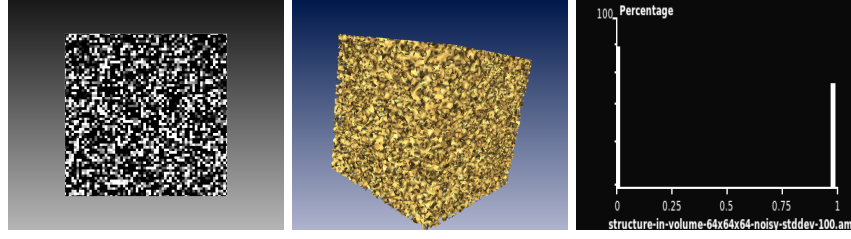


NLM ($search\ window\ size = 21^3$, $neighborhood\ size = 5^3$, $similarity = 3.0$)
 $PSNR = 16.469$

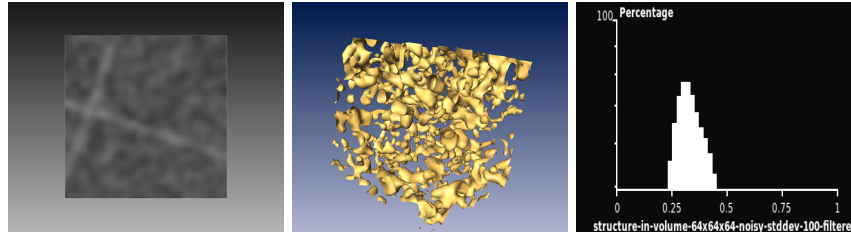


AM-NLM ($\sigma_s = 6.0$, $\sigma_r = 0.5$)
 $PSNR = 19.244$

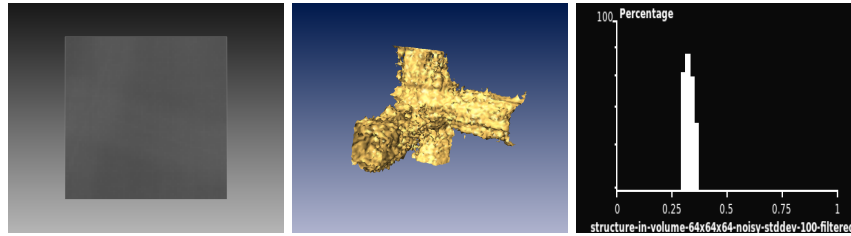
Figure 7: Denoised artificial dataset with Gaussian noise level of $\sigma^2 = 0.1$. The first row shows the noisy version of the dataset that was used as input for the different filters. The rows below show the results of filtering with Gaussian smoothing, NLM, and AM-NLM. The parameters used for each filter are denoted below each image.



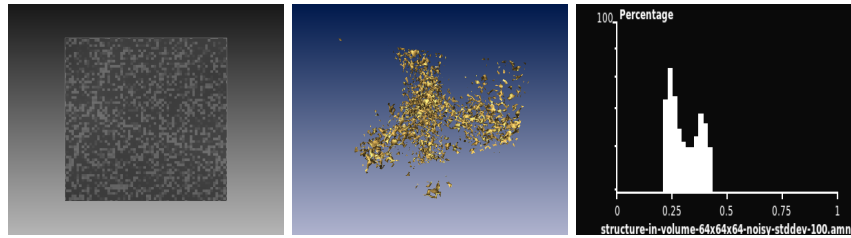
Noisy version ($\sigma^2 = 1.0$)
 $PSNR = 5.890$



Gaussian smoothing ($kernel\ size = 5^3$, $\sigma = 0.4$)
 $PSNR = 7.423$



NLM ($search\ window\ size = 21^3$, $neighborhood\ size = 7^3$, $similarity = 5.0$)
 $PSNR = 7.138$



AM-NLM ($\sigma_s = 8.0$, $\sigma_r = 0.8$)
 $PSNR = 7.570$

Figure 8: Denoised artificial dataset with Gaussian noise level of $\sigma^2 = 1.0$. The first row shows the noisy version of the dataset that was used as input for the different filters. The rows below show the results of filtering with Gaussian smoothing, NLM, and AM-NLM. The parameters used for each filter are denoted below each image.

3.1.2 Denoising a paleontological dataset

In a second series of experiments the results of different filters on a paleontological dataset are assessed. The dataset was obtained from a μ CT-scan of a *Eogaudryceras umbilicostriatus* specimen of about 112 million years old (Cretaceous, Lower Albian). The specimen was filled with calcareous mud containing glauconite mineral. Due to the low PSNR of the μ CT-scan, the ammonite shell (consisting also of calcium carbonate) is barely distinguishable from the surrounding sediments.

The dataset has a total size of $1880 \times 1880 \times 1700$ voxels, consuming approximately 12 GB of memory. A subvolume of size $141 \times 152 \times 180$ voxels was selected (Fig. 9) to explore the effects of different filter configuration parameters (Tab. 1). For a more detailed view of the results of this exploration, we refer the reader to Appendix D.

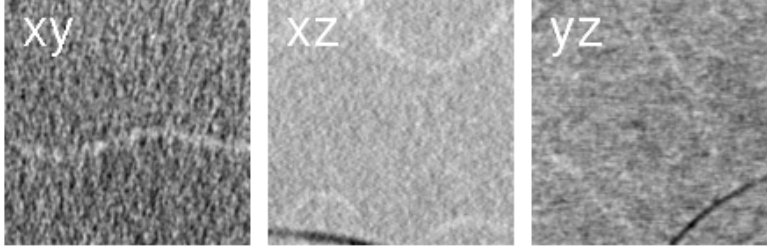


Figure 9: Slices in different orientations through the selected subvolume of the μ CT dataset *Eogaudryceras* sp.

Gaussian smoothing The size of the filter kernel and the standard deviation need to be quite high in order to effectively removing the noise of the dataset. Unfortunately, the filter also removes structural information of the image, which becomes blurrier after its application.

Non-local means ZIBAmira provides an implementation of NLM that can be applied to volumetric data either layer-by-layer (2D) or in 3D. The 2D implementation introduces noticeable artifacts that are only visible when the data volume is explored in directions different to the one the filter was applied to. The results of the 3D implementation did not introduce such artifacts. Applying the filter in several successive iterations considerably reduces noise in the image. Unfortunately, the filter processing can take up to several minutes, even for the small subvolume that was used, depending on the chosen configuration.

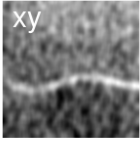
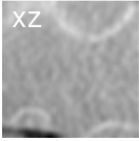

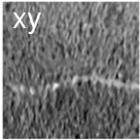
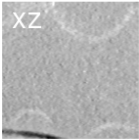
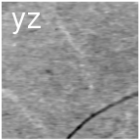
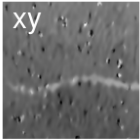
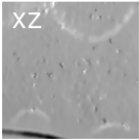
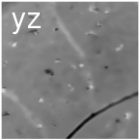
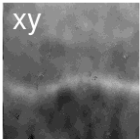

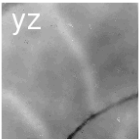
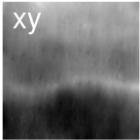

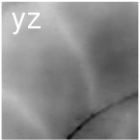
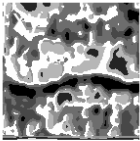
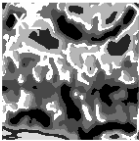
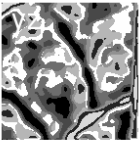
Gaussian *kd*-tree NLM This approximation of the NLM filter yields similar results in a substantially less amount of time. The effect of applying the

same filter in consecutive iterations can be partially replicated by applying it once with appropriately selected filter parameters.

Adaptive Manifolds NLM This filter follows a similar approach as the Gaussian *kd*-tree NLM, but it is noticeable faster. The good results in terms of performance and quality motivated us to design our own implementation of this filter in ZIBAmira (Sect. 4).

***k*-means clustering** One of the results from denoising the dataset with the AM-NLM filter was selected. The contrast was equalized using a CLAHE filter before a *k* – *means* clustering filter was applied. We were able to identify one of the resulting clusters as a partial segmentation of the septa, however, some portions were unrecoverable (Fig. 10).

Table 1: Results of processing a paleontological dataset. The parameters used for each filter are denoted in the middle column. The same set of slices in different orientations facilitates a visual inspection of the results. The 3D variant of NLM yields the best results after applying the same filtering in three consecutive iterations. However, the low performance of this filter prevents an application to the full dataset. Approaches like Gaussian *kd*-tree or adaptive manifolds achieve similar results. A *k*-means clustering can be used to segment the septa in the denoised volumes.

Filter	Parameters	Result		
Gaussian smoothing	$\sigma = 0.8$ kernel size = 8^3			
		iter-001-sigma-08-kernel-08		
2D NLM	search win. = 21^2 neighbor. = 5^2 similarity = 0.8			
		iter-001-patch-05-win-0021-simil-08		
3D NLM	search win. = 21^3 neighbor. = 5^3 similarity = 1.0 iterations = 3			
		iter-003-patch-05-win-0021-simil-10		
Gaussian <i>kd</i> -tree NLM	neighbor. = 7^3 pca = 7 sp. std. = 5.0 rg. std. = 0.3			
		iter-001-patch-07-pca-005-stddev-03-50		
Adaptive manifolds NLM	neighbor. = 5^3 pca = 5 sp. std. = 9.0 rg. std. = 0.3			
		iter-001-patch-05-pca-005-stddev-03-90		
<i>k</i> -means clustering	clusters = 83 patch = 5 iterations = 20			
		iter-020-clusters-08-patch-05		

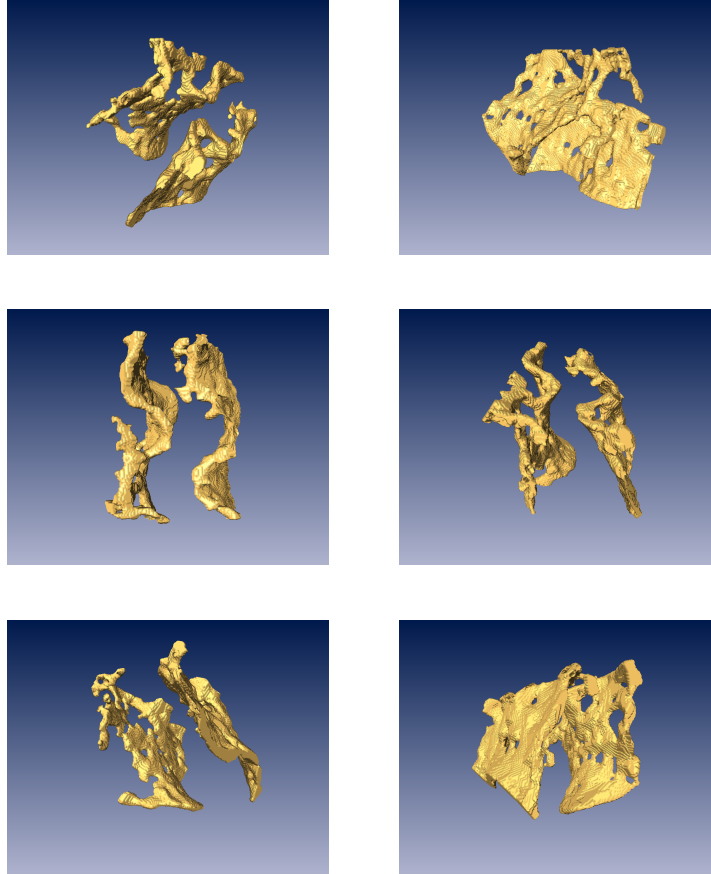


Figure 10: Different views of the segmented septa. The original data was initially denoised with a AM-NLM filter. Afterwards, a CLAHE filter was used to equalize the contrast in the whole volume. Finally, the segmentation was performed via a clustering filter with 20 iterations, 8 clusters, and a patch size of 5^3 . By filling some of the holes manually with ZIBAmira's segmentation editor, the shape of the septum could be partially restored. The portion traversed by the siphuncle, however, seems to be unrecoverable.

4 GPU implementation of an improved AM-NLM

We implemented all previously described algorithms to compute the AM-NLM denoising in CUDA. Our implementation was designed with respect to an application of the filter for large datasets that do not fit into the available memory of the graphical processing unit (GPU). The entire image volume is thus split into partitions of fixed size, enabling the algorithm to perform the denoising process on each of the partitions independently. The size of the partitions is configured depending on the available memory of the GPU. A second objective was to provide immediate visual feedback while a user modifies the parameters of the denoising filter in order to allow for a preview of how the filter settings affect the quality of the denoised image data.

4.1 Filter implementation details

Our implementation operates in two steps. First, a preprocessing step performs a principal-component analysis (PCA) to generate the eigenvectors that allow for an identification of the most important components of all local neighborhoods within the image volume. The second step is the filtering step, where either one or all partitions of the volume are processed to generate the denoised result. The following sections provide a more detailed description of the implementation of both steps.

The objective of the preprocessing step is to reduce the dimensionality of the neighborhood space (i.e., the set of all possible neighborhoods that can be generated from the volume). A PCA is used to identify the most important eigenvectors (i.e., dimensions in the data) in this space. The vectors containing all the pixel values of a neighborhood are projected onto these selected eigenvectors to generate new vectors of reduced dimensionality. For instance, let the size of a neighborhood be $5^3 = 125$ voxels—each possible neighborhood in the volume is a vector of 125 dimensions. Let the number of output dimensions for the PCA be 3—we select the three most important eigenvectors of the neighborhood space. During the filtering, each neighborhood vector will be projected onto these eigenvectors to generate a low-dimensional patch of just 3 dimensions (components). By selecting the most relevant eigenvectors, these components are the most representative ones of the pixel data in the neighborhood. The PCA computation is performed at the beginning, or when the size of the neighborhood or the number of output dimensions of the PCA is changed.

The computation of the eigenvectors starts by selecting a random sample of patches from the whole dataset. To this end, we use a uniformly distributed random sampling process. With a confidence level of α and an error margin of ε , the sample size is computed as in [16]:

$$n = \frac{N z_{1-\frac{\alpha}{2}}^2 r (1-r)}{(N-1) \varepsilon^2 + z_{1-\frac{\alpha}{2}}^2 r (1-r)}, \quad (1)$$

where N is the size of the the number of possible neighborhoods, $z_{1-\frac{\alpha}{2}}$ is the

value that leaves an area of α to the right under the curve of the probability distribution function of a $N(0, 1)$, and r is the response distribution. In our implementation, we used a confidence level of $\alpha = 0.95$, an error margin of $\varepsilon = 0.05$, and a distribution function $r = 0.5$.

The computation of the eigenvectors from the random sample of neighborhoods is an iterative process that has been implemented on the GPU. The process starts by computing a covariance matrix of size $d_R \times d_R$, where d_R is the number of voxels in each neighborhood. The matrix is computed by considering each of the components of a neighborhood as a random variable X_i , where $i = 1, \dots, d_R$. The value in each cell (i, j) of the matrix is obtained as:

$$Cov(X_i, X_j) = E(X_i X_j) - E(X_i)E(X_j). \quad (2)$$

Finally, the preprocessing step ends by computing the eigenvectors using the covariance matrix as a starting point. The eigenvectors are approximated using an iterative approach based on the Gram-Schmidt process [6].

The second step of our implementation, i.e., the filtering process itself, starts by extracting a partition from the dataset, moving the data to the GPU memory, projecting the neighborhoods of the partition into a low-dimensional space, filtering the partition, and copying the denoised partition back to CPU memory. These steps are performed until all partitions are processed. The generation of a set of neighborhoods vectors from the current partition and the subsequent projection are combined into a single operation. Our implementation only stores the set of vectors of reduced dimensionality in memory to keep the memory footprint of this step as little as possible and to increase the computational performance.

The filtering algorithm recursively explores the set of manifolds, which is structured as a tree and built at the same time the data is filtered. Starting from the root node, corresponding to the first manifold, the reduced vectors are projected into the manifold and downsampled. An RF filter [11] is applied to the downsampled data that are upsampled afterwards again to restore the original scale. The blurred results are accumulated into previously blurred manifolds. New manifolds are generated until the tree reaches its maximum height. When the last manifold is processed, the denoised volume is computed.

The aforementioned RF filter approximates the computation of the Gaussian smoothing as a recursive filter. The filter operates in three steps: First processing the rows, second the columns, and finally the slices of the 3D data. Each step requires to traverse the data twice, forwards and backwards. Our GPU implementation processes a single row of data per thread in the X, Y, and Z direction.

The root manifold of the tree is computed as a low-pass filtering of input volume, and has also been implemented by means of the RF filter. As for all remaining manifolds, their construction requires additional steps [10], which are all executed on the GPU.

4.2 Memory optimization

The filtering step of our implementation requires the higher usage of GPU memory of the two steps. For an analysis of a potential optimization, every occurrence of memory allocation and deallocation on the GPU was logged. The implementation uses two kind of different buffers: (i) those allocated through the Thrust API, and (ii) those directly allocated through the CUDA API. The majority of allocations are performed through the CUDA API and, in our design, they are all done at a single point of the implemented code, i.e., a class constructor. In addition, deallocations through the CUDA API are performed within the respective class destructor. To log the memory allocations and deallocations during the execution of the filter, a header with logging code was used [8].

The memory usage in GB during the filtering step varies quite abruptly during the filtering step, and for a specific dataset of 145^3 it reaches peaks close to 1.2 GB (Fig. 11a). The memory footprint was first reduced by reorganizing the code, which allowed to deallocate intermediate buffers immediately after the computation of the manifold has finished. This resulted in a drastic reduction in memory consumption (Fig. 11b).

A close-up of the memory consumption while processing a single manifold reveals some peak allocations that still have the potential for reduction (Fig. 11c). The first peaks correspond to the blurring of the manifold by means of the RF filter. This filter was re-implemented to reduce the number of intermediate buffers and reusing some of the already allocated buffers to compute the filtering in vertical direction and in depth. As a result, the corresponding peak was slightly reduced from approx. 0.45 GB to 0.40 GB (Fig. 11d).

The next two most prominent peaks in memory usage correspond to the computation of the next two manifolds in the tree hierarchy (denoted as η_- and η_+). Initially, both buffers were computed together, but a reordering of the operations showed that it was possible to obtain η_+ from the values of η_- . This allowed us to implement some additional deallocations of buffers, reducing the memory usage for these computations in approx. 0.20 GB and requiring fewer steps to complete the processing of a manifold (Fig. 11e).

The aforementioned optimizations managed to reduce the memory footprint of the whole filtering from 1.2 GB to 0.5 GB (Fig. 11f).

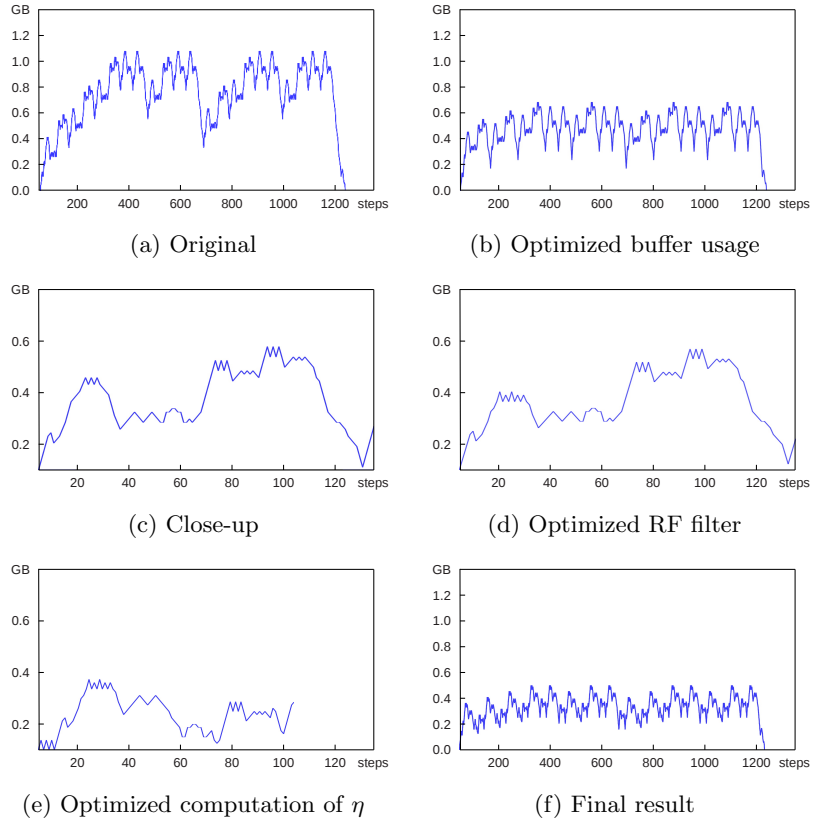


Figure 11: GPU memory usage in GB for filtering a single partition of 145^3 elements (75^3 voxels in the kernel of the partition, and a border with a width of 35 voxels) with neighborhood size 5^3 , PCA output dimensions 3, spatial standard deviation 5.0, and range standard deviation 0.5.

4.3 Performance optimization

Many operations performed during the filtering were implemented using the CUDA Thrust library and the CUBLAS library. An analysis of the performance of the application using the CUDA profiler showed that more than 40% of the time was devoted to a single operation `gemv` from the CUBLAS API. This operation was called during the computation of the eigenvector that is used to compute the two new manifolds η_- from η_+ . It multiplies a transposed matrix of size $k \times N$ by a vector of size N , resulting in a new vector of size k , where N is the total number of patches in a partition and k is the number of elements in each patch (notice also that $N \gg k$).

We concluded that the reduction step required during the computation is not efficient for such a small k (in our tests, we were using $k = 3$, whereas $N = 145^3$), we reimplemented this operation using a custom kernel that executed the multiplication and we used the Thrust API to perform the reduction. The operation, which took more than 150 ms. on an Nvidia GTX Titan graphics adapter, decreased to less than 1 ms.

We achieved another important boost in performance by reimplementing the most expensive operation (RF filter) using multi-channeled CUDA surfaces. As the number of elements in each patch is usually between 2 and 4, storing each patch as a `float2` or `float4` data structure stored in the surface memory of the GPU was perfectly plausible. When the size of the patches is not within this range, we default to the previous implementation of this filter, that works exclusively in global memory.

5 AM-NLM-based denoising in ZIBAmira

We have implemented a module `CUDAAdaptiveManifoldsNLM` in ZIBAmira that performs a GPU-accelerated AM-NLM filtering based on the specification of Gastal and Oliveira [10]. To provide a filtering tool where a user can interactively assess the influence of the filtering parameters for large datasets, a preview approach has been implemented where a user can apply the filter to his or her data for a previously defined subvolume, i.e., a region of interest (ROI). The processing of small regions nearly occurs in real-time, thus a user can change the parameters while their influence on the filtered image data is immediately visualized. Once the user is satisfied with the result of the preview, the filter can be applied with the respective parameter settings to the entire dataset. In the remainder of this section we describe how to use the new filtering tool.

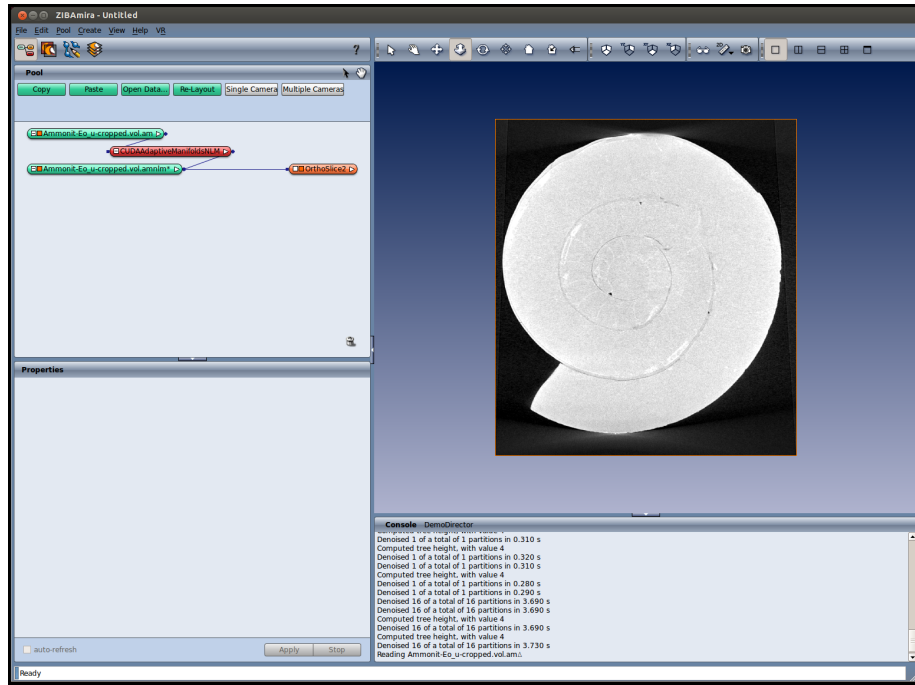


Figure 12: Screenshot of ZIBAmira with data, viewer, and AM-NLM filter module loaded into the object pool.

The user first loads the dataset that is to be processed by the filter into ZIBAmira and then selects our denoising module from the list of available modules. A red module icon entitled `CUDAAdaptiveManifoldsNLM` appears in ZIBAmira's object pool. After attaching this module to the input dataset, a new output dataset is generated. Initially, this dataset is a copy of the input dataset. When the input dataset is filtered, the output dataset is modified with the filtered data. An additional `Orthoslice` module can be connected

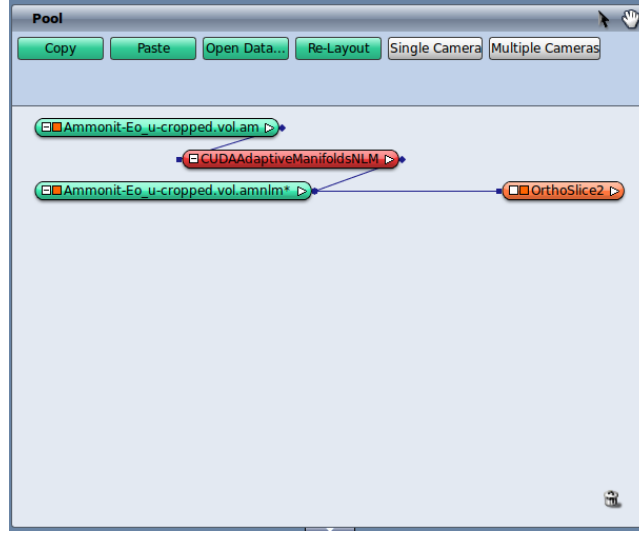


Figure 13: Close-up view of ZIBAmira's object pool.

to the output dataset (Fig. 12). Within the viewer of ZIBAmira the content of the currently selected slice of the image data via the `Orthoslice` module is visualized.

By selecting the AM-NLM module icon in the pool (Fig. 13), a control panel is shown, where the user can configure the different options (Fig. 14). Via the `CUDA device` port a GPU device can be selected where the filtering is to be executed. If there are several GPU cards available in the system, the user can select the one that is to be used. The port also shows the available GPU memory of the respective graphics adapter.

The `Spatial std dev` and `Intensity std dev` ports enable the user to configure the values of σ_s and σ_r , respectively. The minimum and maximum values for these two parameters have been chosen to support the most common use cases. For instance, the intensity standard deviation can take values from the range $[0.1, 1.0]$. As the intensity values of the dataset are converted to the interval $[0, 1]$ during the filtering process; using values of σ_r outside of this range is not encouraged. Notwithstanding, these limits can be changed by a user via the port's properties.

Intuitively, the spatial standard deviation σ_s allows the user to control the blurriness of the filter response. The higher this value, the blurrier the result will be. If this value is too low, the effect of the filtering will be poor, i.e., very little noise will be removed. The intensity standard deviation σ_r acts as a fine tuning control of the blurriness. This parameter allows to slightly increase or decrease the blurriness of the result. In some cases, there is a threshold from where on this parameter no longer increases the blurriness of the output, because all neighboring voxels are already contributing to the final intensity of

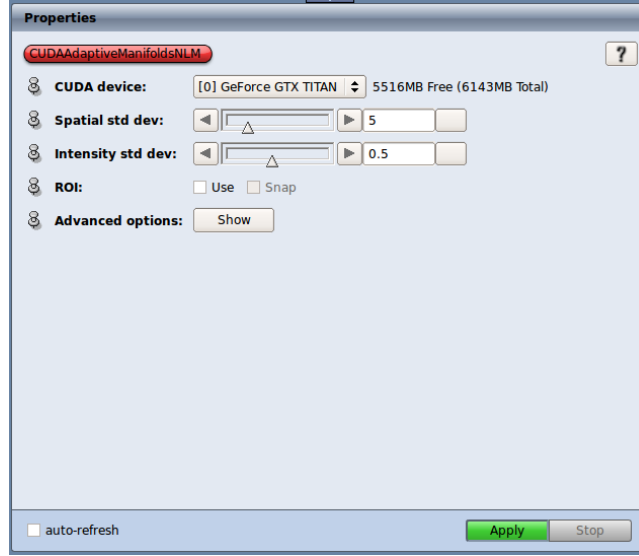


Figure 14: Control panel GUI of the AM-NLM filtering module.

the output voxel, regardless of their intensity values. The effect is then similar to applying a Gaussian smoothing on the input. Moreover, if the value of the spatial standard deviation σ_s is too high, any change of the value of the intensity standard deviation σ_r usually has only a minimal effect.

The ROI toggle allows for an automatic attachment of a **SelectROI** to the module, so the user can apply the filter to a small subregion of the dataset. In case this toggle is activated, a ROI frame appears in the viewer whose boundaries can be interactively dragged to change its position and size (Fig. 15).

Initially the ROI has the same size as a partition. The user can select which partition is to be filtered by dragging the ROI in the viewer. If the **Snap** toggle is active, the user can also click on the **OrthoSlice** and the ROI will be automatically positioned at the respective partition. Once the user has selected a partition to denoise, pressing the **Apply** button will initiate the filtering process which is visualized within the ROI (Fig. 16).

Using the parameters of the connected **OrthoSlice** module, the user can adjust the dynamic range for the visualization of the image data (Fig. 17).

The user can continue changing the values of the parameters by pressing the **Apply** button to denoise the same partition, or reposition the ROI to apply the filter to a different region of the image data. A user can also toggle the **auto-refresh** option. Hence, after every change in the parameters the filter is immediately applied without the need to press **Apply**. After appropriate values for the filter parameters have been found, the whole dataset can be processed using these parameters (Fig. 18–19).

The Advanced options are split in three tabs. The first tab, **Partitions**, allows the user to configure the size of the partitions and the radius of the

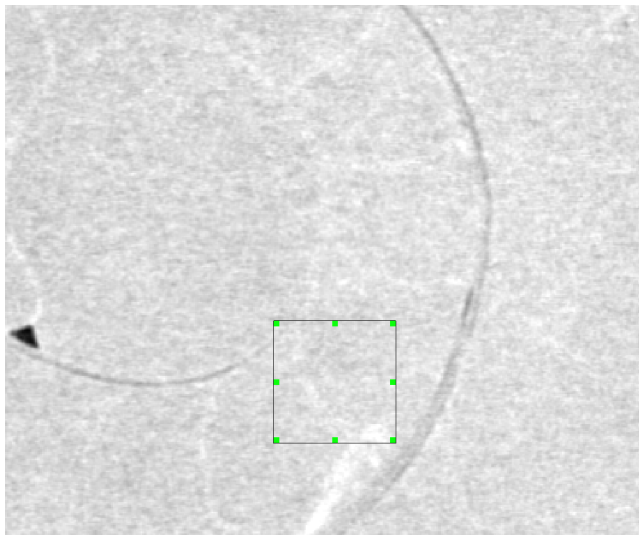


Figure 15: ROI selection of the AM-NLM filtering module.

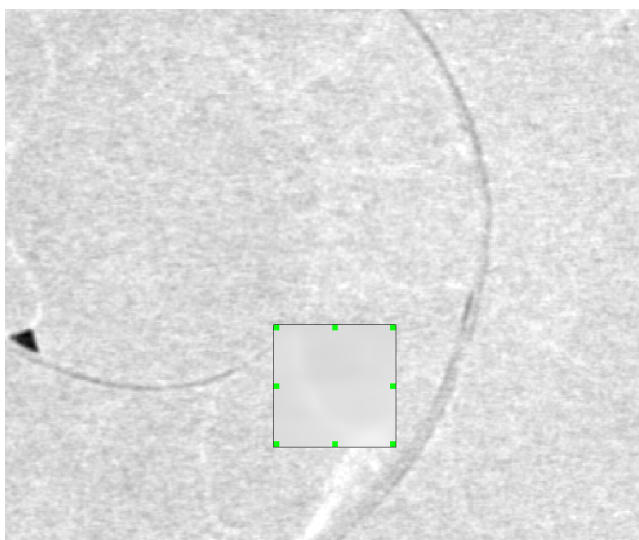


Figure 16: Result of the AM-NLM filtering applied to the given ROI/partition.

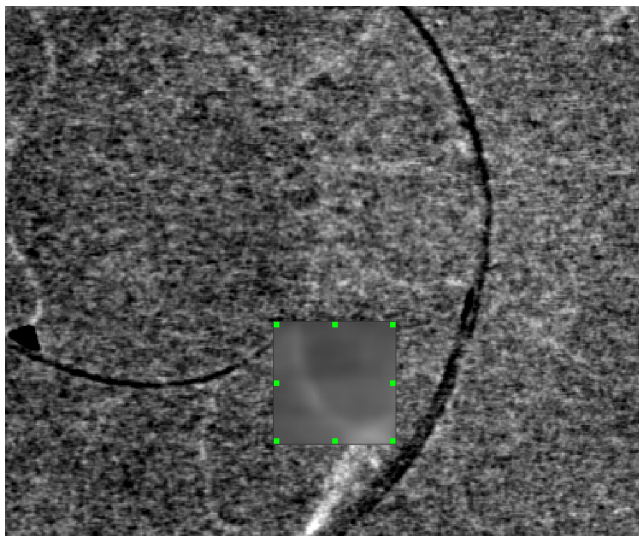


Figure 17: Result of the AM-NLM filtering applied to the given ROI/partition with adjusted dynamic range.

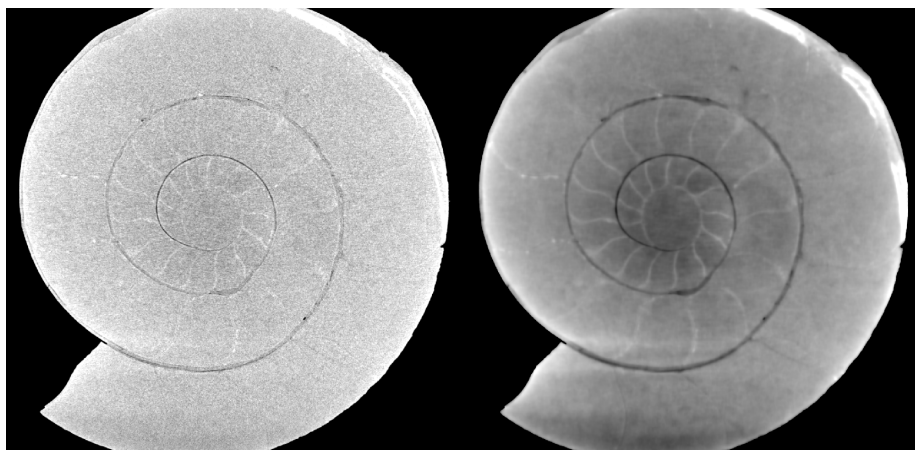


Figure 18: Slice of the initial dataset (left), result of the AM-NLM filtered data (right).

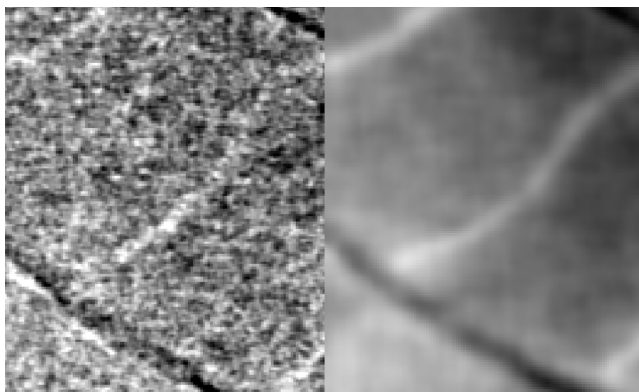


Figure 19: Close-up view.

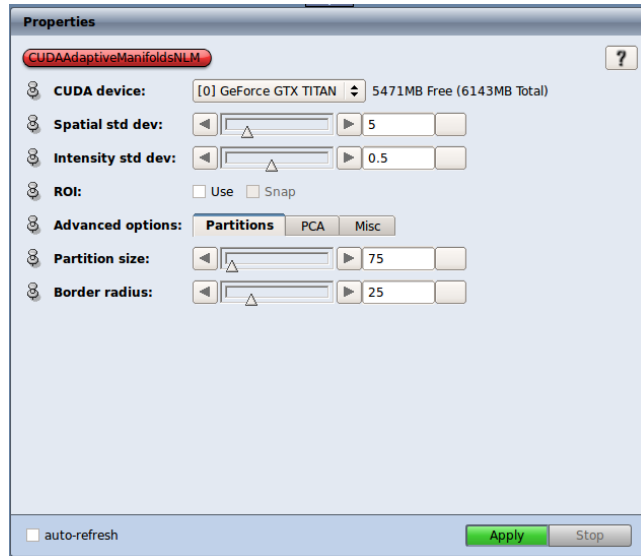


Figure 20: User interface of the AM-NLM module with Advanced options for partitioning activated.

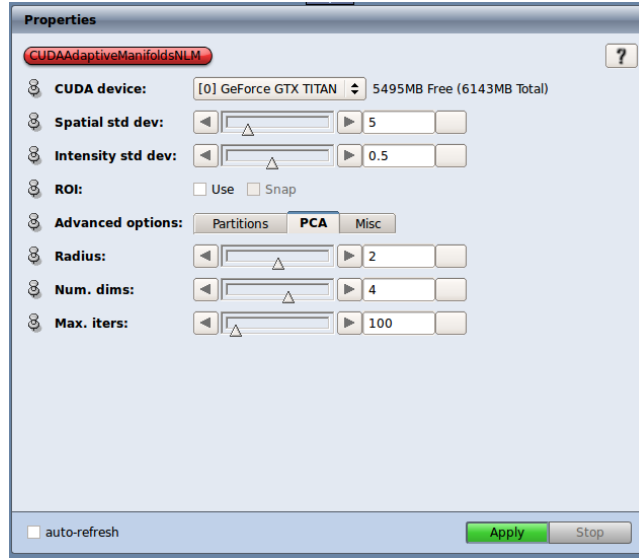


Figure 21: User interface of the AM-NLM module with Advanced options for PCA activated.

border surrounding each partition. Each partition has a border that overlaps to the contents of adjacent partitions. The size of the partition and the width of the border have an important impact on the performance of the algorithm, especially when previewing the result of filtering a single partition. The user can reduce these values in order to get faster results. Reducing the size of the border might result in undesired artifacts at the boundaries of each partition. The default values provide a good compromise between performance and quality (Fig. 20).

Via the PCA tab of Advanced options a user can configure the radius of the local neighborhood. For a 3D dataset, the size of the local neighborhood is $(2r + 1)^3$, where r is the radius. In the Num. dims. port a user can set the number of output dimensions for the PCA (the default value, 4, provides a good result, in terms of quality and performance). With Max. iters the user can set the maximum number of iterations for the PCA. This number might be increased in those cases where the default value, 100, is not sufficient to compute the eigenvectors (Fig. 21).

Via the Misc tab of Advanced options a user can configure the number of random samples that is used to perform the PCA. The default number is computed from the size of the volume with a confidence of 95%. The Eigenvectors port is for a configuration of the number of iterations needed to compute a single eigenvector (Fig. 22). This eigenvector is used during the construction of each manifold. Increasing this value might increase the quality of the results. At the same time, it might also have a negative impact on the performance.

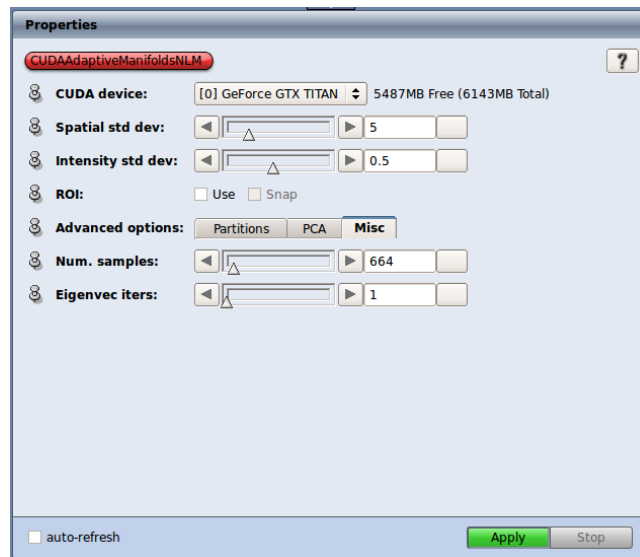


Figure 22: User interface of the AM-NLM module with miscellaneous Advanced options activated.

6 Conclusions and future work

In this report we tackled the problem of denoising very large and noisy datasets, in particular, datasets obtained from tomographic images of fossil structures embedded in sedimentary rocks.

Among the different filtering approaches tested in this work, the classical Gaussian smoothing filter, while being an excellent and fast noise-removal tool, presented the undesirable characteristic of blurring the structural information that we wanted to preserve. On the other hand, the state-of-art non-local means (NLM) filter provided excellent results, but at the expense of large computation times. This made its application unfeasible in the context of very large datasets.

Approaches to the NLM such as the Gaussian kd -tree and the adaptive manifolds (AM-NLM) were able to provide acceptable results by carefully choosing the correct parameter values. We implemented a solution based on the latter within ZIBAmira. This solution splits large volumetric datasets into smaller partitions, and performs the filtering within the GPU exploiting its parallel hardware. Our solution is able to offer a preview of the filtering result in just a few seconds for a small region of large volumetric datasets. A user can configure the filter interactively with the help of a visual feedback.

In future work, we would like to improve our current GPU implementation of the AM-NLM by introducing a more stable computation of the eigenvectors that does not depend on the Gram-Schmidt process. More generally, we believe that there are still opportunities to improve the performance and reduce the memory footprint—for example, by reducing the size of the manifolds. Furthermore, other variants of the NLM filter could be explored, such as random-sampling [3] or shape-adaptive neighborhoods [7].

Last, but not least, denoising is only a preprocessing stage for a subsequent image segmentation. In this report we present some initial results using the k -means clustering. Although these results look promising, some of the structures were unrecoverable with the information extracted from the filter. Further research towards restoring as much as possible from the fossil’s septa will require an additional study of the denoising filters and improved segmentation algorithms.

Acknowledgements

This work was funded by the German Research Foundation (DFG), grant no. ZA 592/2-1; HO 4674/2-1. The authors would like to thank Hendrik Wesendonk (TPW Prüfzentrum, Neuss, Germany) for scanning the two ammonite specimens (Quenstedtoceras and Eogaudryceras), Robert E. Lemanis for testing our implementation and proofreading this manuscript. In addition we also thank David Günther (Saarland University and Max-Planck Institute for Informatics, Saarbrücken) for his valuable comments on his own implementation of a GPU-assisted NLM algorithm in ZIBAmira.

Appendices

A Notation used in this document

Let $S \subset \mathbb{R}^3$ be a spatial domain (similarity in position), and let $R \subset \mathbb{R}$ be a range domain (similarity in intensity). A signal $f : S \rightarrow R$ associates values from the spatial domain S into the range domain R .

Let $\{p_1, \dots, p_N\}$ be a collection of samples of S arranged in a uniform grid. We call each p_i a voxel and denote $f_i = f(p_i)$ the intensity of the voxel p_i . In short, an image volume is a collection of sampled intensity values in 3D space, each one representing an averaged intensity value per voxel with an associated coordinate.

We denote by $\{N_i\}_{i \in V}$ a neighborhood in S , where each $N_i \subset S$ verifies that:

$$\begin{aligned} \forall p_i \in S, p_i \in N_i, \\ \forall p_i, p_j \in S, p_j \in N_i \Leftrightarrow p_i \in N_j. \end{aligned}$$

The restriction of f to a neighborhood N_i is computed as:

$$f_{N_i} = \{f_j = f(p_j)\}_{p_j \in N_i}.$$

Finally, we denote by $\hat{p}_i = (p_i, f_i) \in S \times R$ a point in a 4-dimensional space resulting from the concatenation of the spatial coordinates $p_i \in S$ and the scalar value $f_i \in R$.

B Mathematical formulation of denoising filters

The following sections heavily rely on the notation described in Appendix A.

B.1 Gaussian smoothing

The estimated value g_i from the filtered signal g is computed as a convolution of all the voxels in the input signal f with a Gaussian kernel of variance σ^2 :

$$g_i = g(p_i) = \sum_{p_j \in S} \phi_\sigma(p_i - p_j) f_j \quad (3)$$

The Gaussian smoothing function ϕ_σ has the following shape:

$$\phi_\sigma(p_i - p_j) = \exp\left(-\frac{|p_i - p_j|^2}{2\sigma^2}\right). \quad (4)$$

Quite often, a normalization factor is included in order to ensure that the sum of all the contributions from nearby voxels add to one:

$$g_i = \frac{1}{W_i} \sum_{p_j \in S} \phi_\sigma(p_i - p_j) f_j, \quad (5)$$

with W_i as a normalization factor for voxel p_i :

$$W_i = \sum_{p_j \in S} \phi_\sigma(p_i - p_j). \quad (6)$$

B.2 Median filter

The median filter substitutes each pixel f_i by the median of the pixel itself and all pixels that belong to its neighboring window:

$$g_i = \tilde{f}_{N_i}. \quad (7)$$

B.3 Anisotropic diffusion

Anisotropic diffusion is an iterative algorithm that modifies the image by means of the following partial differential equation:

$$\frac{\partial f}{\partial t} = \text{div}(c \nabla f) = \nabla c \cdot \nabla f * c \Delta f, \quad (8)$$

where div denotes the divergence operator, ∇ denotes the gradient, Δ denotes the Laplacian, and c controls the rate of diffusion.

In ZIBAmira's implementation of this filter, the value of function c is computed as:

$$c(|\nabla f|) = 1 - \exp\left(-\frac{3.315}{\left(\frac{|\nabla f|}{K}\right)^4}\right), \quad (9)$$

where K controls the diffusivity.

B.4 Bilateral filtering

The Gaussian smoothing filter uses distances that are measured in the spatial domain. One could define a similar filter as Gaussian smoothing using distances or similarities that are measured in the range (intensity) domain:

$$g_i = \sum_{p_j \in S} \phi_\sigma(f_i - f_j) f_j. \quad (10)$$

Bilateral filters [19] operate in both, the spatial and the range domain of an image. Therefore, Eq. (3) of Gaussian smoothing is combined with a filter that operates in the range domain:

$$g_i = \frac{1}{W_i^b} \sum_{p_j \in S} \phi_{\sigma_s}(p_i - p_j) \phi_{\sigma_r}(f_i - f_j) f_j. \quad (11)$$

The term σ_s denotes the standard deviation of the signal within the spatial domain (spatial standard deviation), and the term σ_r denotes the standard

deviation of the signal within the intensity domain (range standard deviation). The normalization factor is computed as:

$$W_i^b = \sum_{p_j \in S} \phi_{\sigma_s}(p_i - p_j) \phi_{\sigma_r}(f_i - f_j) \quad (12)$$

B.5 Non-local means

Non-local means (NLM) [2] is a generalization of the bilateral filter that works on local neighborhoods instead of individual pixel or voxel values:

$$g_i = \frac{1}{W_i^n} \sum_{p_j \in S} \phi_{\sigma_s}(p_i - p_j) \phi_{\sigma_r}(f_{N_i} - f_{N_j}) f_j, \quad (13)$$

where the normalization factor is computed as:

$$W_i^n = \sum_{p_j \in S} \phi_{\sigma_s}(p_i - p_j) \phi_{\sigma_r}(f_{N_i} - f_{N_j}). \quad (14)$$

C Signal-processing approaches to bilateral filtering

C.1 Linearization of the bilateral filter

The application of the signal processing paradigm converts the bilateral filter into a linear filter [14]. Eqs. (11) and (12) of the bilateral filter can be combined into one equation:

$$\begin{pmatrix} W_i^b g_i \\ W_i^b \end{pmatrix} = \sum_{p_j \in S} \phi_{\sigma_s}(p_i - p_j) \phi_{\sigma_r}(f_i - f_j) \begin{pmatrix} f_i \\ 1 \end{pmatrix}. \quad (15)$$

Eq. (11) of the bilateral filter was multiplied by W_i^b , hence effectively removing the division from the right-hand side. The new equation and the equation to compute the normalization factor were put together using vector notation.

Let us assume that, $\forall p_j \in S, W_j = 1$. We can write:

$$\begin{pmatrix} W_i^b g_i \\ W_i^b \end{pmatrix} = \sum_{p_j \in S} \phi_{\sigma_s}(p_i - p_j) \phi_{\sigma_r}(f_i - f_j) \begin{pmatrix} W_j f_i \\ W_j \end{pmatrix}. \quad (16)$$

Eq. (16) expresses every couple $\begin{pmatrix} W_i^b g_i \\ W_i^b \end{pmatrix}$ as a linear combination of neighboring couples $\begin{pmatrix} W_j f_i \\ W_j \end{pmatrix}$. Each of these tuples, or two-dimensional vectors, are called homogeneous intensities. The computation of g_i is nonlinear, as it requires dividing the first coordinate of each homogeneous intensity by the second one, but this operation can be postponed to the very end of the process.

The product of the functions ϕ_{σ_s} and ϕ_{σ_r} in Eq. (16) defines a high dimensional product space $S \times R$. However, the sum is in a lower dimensional space, S , and hence, it is not a convolution. A higher dimensional space given by the tuples $(x, y, I(x, y))$, where $(x, y) \in S$ and $I(x, y) \in R$, can be considered instead. Thus, the image becomes a manifold of this higher dimensional space, and Eq. (16) can be rewritten as:

$$\begin{pmatrix} W_i^b g_i \\ W_i^b \end{pmatrix} = \sum_{(p_j, f_k) \in S \times R} \phi_{\sigma_s}(p_i - p_j) \phi_{\sigma_r}(f_i - f_j) \delta(f_k - f_j) \begin{pmatrix} W_j f_i \\ W_j \end{pmatrix}, \quad (17)$$

where δ is the Kronecker delta, defined as:

$$\delta(x) = \begin{cases} 1 & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{cases} \quad (18)$$

Notice that $\delta(\zeta - f_j) = 0$ when $\zeta \neq f_j$. This equation is basically the same as the previous linear one, but now the sum is computed on the space given by $S \times R$.

Furthermore, the product $\phi_{\sigma_s} \phi_{\sigma_r}$ defines a separable Gaussian kernel $\phi_{\sigma_s \sigma_r}$:

$$\phi_{\sigma_s \sigma_r} : S \times R \rightarrow \mathbb{R} \quad (19)$$

$$(p_i - p_j, f_i - f_j) \mapsto \phi_{\sigma_s}(p_i - p_j) \phi_{\sigma_r}(f_i - f_j). \quad (20)$$

Let us define the functions v :

$$v : S \times R \rightarrow R, \quad (21)$$

$$(p_j, f_k) \mapsto f_j, \quad (22)$$

and w :

$$w : S \times R \rightarrow R, \quad (23)$$

$$(p_j, f_k) \mapsto \delta(f_k - f_j) W_j, \quad (24)$$

where $W_j = 1$, as stated earlier.

With these functions one can write equation (17) as:

$$\begin{pmatrix} W_i^b g_i \\ W_i^b \end{pmatrix} = \sum_{(p_j, f_k) \in S \times R} \phi_{\sigma_s \sigma_r}(p_i - p_j, f_i - f_j) \begin{pmatrix} w(p_j, f_k) v(p_j, f_k) \\ w(p_j, f_k) \end{pmatrix}, \quad (25)$$

which computes the value at point $\hat{p}_i = (p_i, f_i)$ of the following convolution:

$$\begin{pmatrix} W^b g \\ W^b \end{pmatrix} = \phi_{\sigma_s \sigma_r} * \begin{pmatrix} W f \\ W \end{pmatrix}. \quad (26)$$

C.2 Bilateral grid

The bilateral grid [5] is an efficient implementation of a bilateral filter. The bilateral grid provides a data structure that enables the application of the bilateral filter using the aforementioned signal processing approach. The bilateral grid is actually a 3D array where the first two dimensions correspond to 2D positions in the image plane and form the spatial domain, and the third dimension corresponds to the range domain (in this case, the intensity values of each pixel in the image). The grid is regularly sampled in each dimension, where s_s is the sample rate of the spatial axes, and s_r is the sample rate of the range axis. The bilateral grid stores homogeneous intensities, given by the tuples (wI, w) .

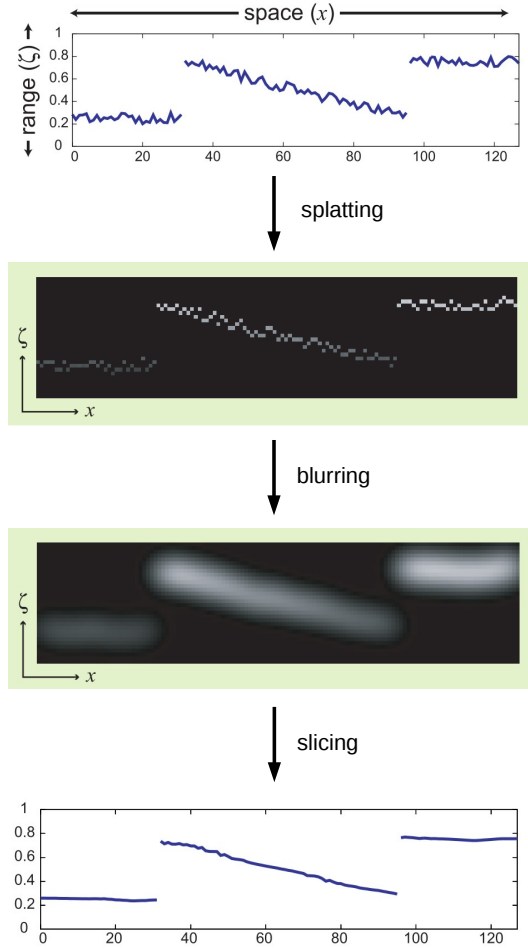


Figure 23: Steps of the bilateral grid approach (Reprinted from [14]).

The bilateral grid approach comprises three steps (Fig. 23):

1. **Splatting:** Given an image f in the range $[0, 1]$, the bilateral grid Γ is first initialized to zero:

$$\forall(i, j, k, l), \Gamma(i, j, k, l) := (0, 0). \quad (27)$$

Then, each cell in the grid is filled with data from its closest pixels:

$$\forall(x, y, z) \in S, \Gamma([x/s_s], [y/s_s], [z/s_s], [f(x, y, z)/s_r]) += (f(x, y, z), 1). \quad (28)$$

This operation is denoted as:

$$\Gamma = c(f). \quad (29)$$

2. **Blurring:** Let ϕ be a function that performs a Gaussian convolution on a 4D space. One can obtain a new grid just by convolution:

$$\tilde{\Gamma} = \phi_{\sigma_r \sigma_s} * \Gamma \quad (30)$$

3. **Slicing:** A 3D map is obtained by accessing the values stored in the grid at positions $(x/s_s, y/s_s, z/s_s, f(x, y, z)/s_r)$ using interpolation. This operation is denoted as:

$$g = s_f(\Gamma). \quad (31)$$

Hence, the bilateral filtering can be expressed in terms of a bilateral grid:

$$g = s_f(\phi_{\sigma_s \sigma_r} * c(f)). \quad (32)$$

C.3 Gaussian kd-tree

The Gaussian *kd*-tree is an approximation to the bilateral grid that aims to reduce the number of computations by using a Monte-Carlo approach [1]. The basic operations of the filtering process are the same as of the bilateral grid (splatting, blurring, and slicing). However, the algorithmic approach uses a *kd*-tree that sparsely represents the high-dimensional space by values stored in its nodes, thus, restricting the operations to the points of the image close to those nodes (Fig. 24).

The high-dimensional space containing the values \hat{p}_i is divided into hyperrectangles or cells. A tree is constructed, where each node represents such a hyperrectangle. The root node is associated to the cell that covers the entire space. The leaf nodes are associated to those elementary cells that are not further divided. The splatting operation samples the input signal f into the centroids of the elementary cells. The blurring operation is performed for each cell, based on the values stored at their centroids. Finally, the slicing operation derives the value for each \hat{p}_i from the centroids of the adjacent cells (Fig. 25).

The filtering operations are implemented as queries to the tree. Each query is a tuple that contains a position in the grid, a number of samples, and a standard

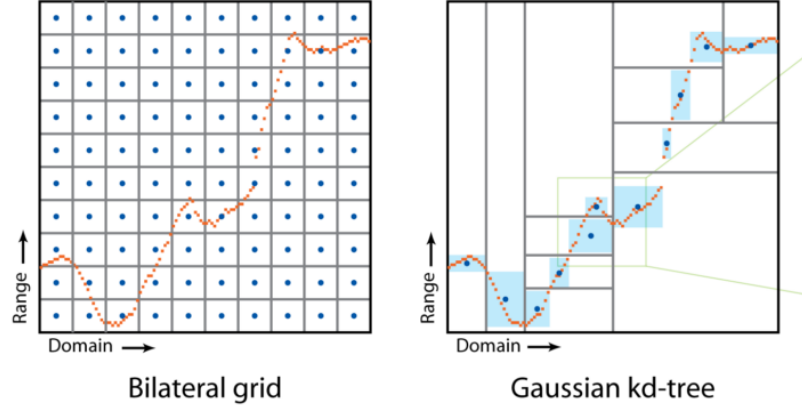


Figure 24: Schematic comparison between the bilateral grid and the Gaussian kd -tree (Reprinted from [14]).

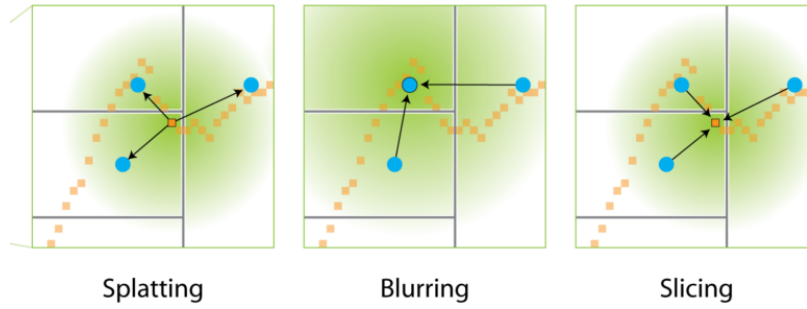


Figure 25: Illustration of the filtering operations in three adjacent nodes of a Gaussian kd -tree (Reprinted from [14]).

deviation. The query is solved using a Monte-Carlo approach. Samples are randomly computed surrounding the query position, and then they are assigned to the leaf nodes whose centroids are closest to the query position. For each of these leaf nodes, the Gaussian distance between the centroid p and the query q is computed as $w = \exp(-|p - q|^2 / 2\sigma^2)$, and the result is then multiplied by the number of samples weighted by the accumulated probability of the leaf node being selected.

The blurring operation computes a Gaussian smoothing on the splatted values stored in the centroids of each leaf node. Instead of querying about the positions in the grid, the queries are now about the centroids of the leaf nodes. A query returns, for each leaf node, a blurred value obtained from the leaf values of close neighboring leaf nodes. These new values are also stored in the centroids, in order to prepare for the next step.

Finally, the slicing operation uses another query that resolves in the same way as for the blurring operation, but in this case the queried positions are the reference ones, also used during the splatting step. The whole process finishes by dividing each obtained value by its corresponding weight.

C.4 Adaptive manifolds

Adaptive manifolds (AM) [10] is another approximation of the bilateral grid. In this approach, a set of non-linear manifolds is constructed considering the standard deviations of the filter and the shape of the signal in the high-dimensional space. The filter response is computed in the positions of the manifolds.

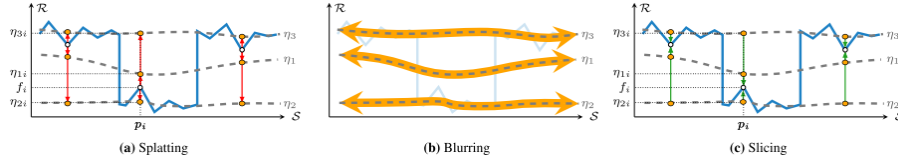


Figure 26: Operations of the AM filtering approach (Reprinted from [10]).

As in the bilateral grid, the filtering process involves three operations (Fig. 26):

1. **Splatting:** During the splatting operation, a Gaussian weighted-distance projection of the intensity values of the input signal f onto each manifold η_k is performed.

$$\psi_{splat}(\hat{\eta}_{N_i k}) = \phi_{\sigma_R}(\eta_{N_i k} - f_{N_i})f_i, \quad (33)$$

where ϕ_{σ_R} is a Gaussian-weighted function:

$$\phi_{\sigma_R} = \exp\left(-\frac{|\eta_{N_i k} - f_{N_i}|^2}{2\sigma_R^2}\right). \quad (34)$$

2. **Blurring:** The blurring operation performs a Gaussian smoothing of elements of $\psi_{splat}(\hat{\eta}_{N_{ik}})$ for the current manifold $\eta_{N_{ik}}$, resulting in new values $\psi_{blur}(\hat{\eta}_{N_{ik}})$.
3. **Slicing:** The slicing operation computes the filter response by interpolating the blurred values of the manifolds. Let K be the total number of manifolds, then:

$$g_i = \frac{\sum_{k=1}^K w_{ik} \psi_{blur}(\hat{\eta}_{N_{ik}})}{\sum_{k=1}^K w_{ik} \psi_{blur}^0(\hat{\eta}_{N_{ik}})}, \quad (35)$$

with $w_{ik} = \phi_{\sigma_R}(\eta_{N_{ik}} - f_{N_i})$. The values $\psi_{blur}^0(\hat{\eta}_{N_{ik}})$ correspond to a blurred version of $\psi_{splat}^0(\hat{\eta}_{N_{ik}})$, which is calculated as:

$$\psi_{splat}^0(\hat{\eta}_{N_{ik}}) = \phi_{\sigma_R}(\eta_{N_{ik}} - f_{N_i}). \quad (36)$$

The set of manifolds is structured in a full binary tree. The amount of work needed to process the manifolds is given by the height of the so-called manifold tree, which is computed as:

$$H = 2 + \max(2, \lceil H_s L_r \rceil), \quad (37)$$

where H_s is the height computed from the spatial standard deviation of the filter, and L_r is a linear correlation computed from the range standard deviation:

$$H_s = \lfloor \log_2(\sigma_{s_{max}}) \rfloor - 1, \quad (38)$$

$$L_r = 1 - \sigma_{r_{min}}. \quad (39)$$

The standard deviations $\sigma_{s_{max}}$ and $\sigma_{r_{min}}$ are given by:

$$\sigma_{s_{max}} = \sqrt{\max(\Sigma_s)}, \quad (40)$$

$$\sigma_{r_{min}} = \sqrt{\min(\Sigma_r)}, \quad (41)$$

where Σ_s and Σ_r are the diagonal covariance matrices that control the decay of the Gaussian kernel that is used during filtering the spatial and range dimensions, respectively.

In practice, it is common to use conventional isotropic kernels when filtering in space and range, with standard deviations σ_s and σ_r , respectively. Covariance matrices are then obtained as:

$$\Sigma_s = \sigma_s^2 I_{d_s}, \quad (42)$$

$$\Sigma_r = \sigma_r^2 I_{d_r}, \quad (43)$$

where I_d is a tridimensional identity matrix of size $d \times d \times d$. σ_s is measured in pixels, and σ_r is measured in normalized units (with values between 0 and 1).

Fig. 27 shows a plot of the tree height function for values of σ_s in the range $[1, 128]$ and values of σ_r within the range $[0, 1]$. As derived from previous equations, the tree height has a lower bound of 4, which means that at least $2^4 - 1 = 15$ manifolds will be generated. The tree height slowly increases for low values of σ_s and high values of σ_r . For the usual range of values, the tree height remains between 4 and 8.

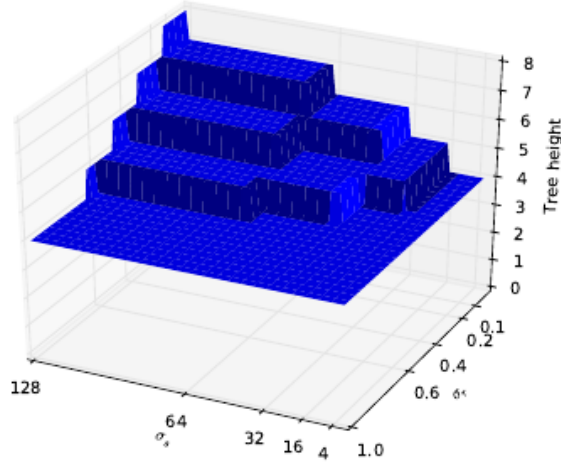


Figure 27: Values of the tree height for different σ_s and σ_r .

D Evaluation results

The following section presents a more comprehensive view of the different results obtained during the evaluation of the parameter space of a collection of denoising filters and their responses to a representative subregion of a paleontological dataset of size $141 \times 151 \times 180$ pixels.

D.1 Gaussian smoothing

We tested the Gaussian smoothing filter with different parameter values (Tab. 2). During the first test a fixed value of the standard deviation was selected, and the filter was applied to the same dataset with different kernel sizes (Fig. 28). The second test opted for the complementary approach, i.e., fixing the kernel size while trying different values for the standard deviation (Fig. 29).

Table 2: Parameter values of the Gaussian smoothing filter used in our tests.

Parameter	Value
Filter	Gaussian smoothing
Direction	3D
Sigma (standard deviation)	$\{0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$
Kernel size	$\{3^3, 4^3, 5^3, 6^3, 7^3, 8^3, 9^3, 10^3\}$

As the images show, the values of the parameters kernel size and standard deviation σ need to be rather large in order to remove the noise in the dataset. However, this blurring effect also removes the structural information in the image.

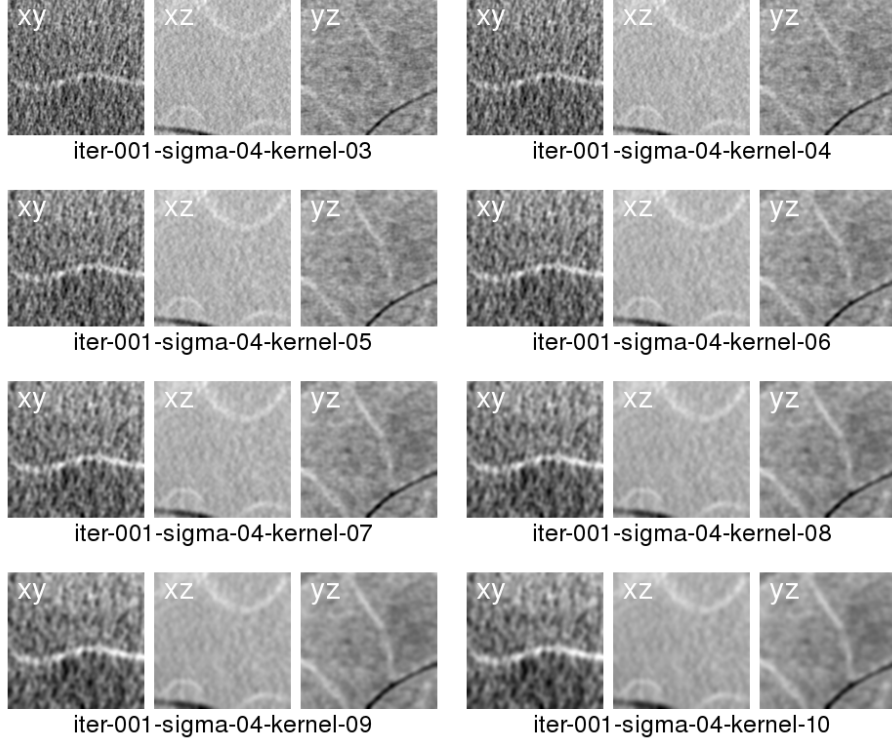


Figure 28: Selected slices in different orientations of the paleontological dataset. Gaussian smoothing was applied with a standard deviation $\sigma = 0.4$ and different kernel sizes in the range $[3^3, 10^3]$. The effect of the kernel size is quite strong—the images with the largest kernel sizes show a distinct blurring effect. The level of noise is considerably reduced when the kernel size is large enough, however, having the undesirable effect of blurring septal structures as well.

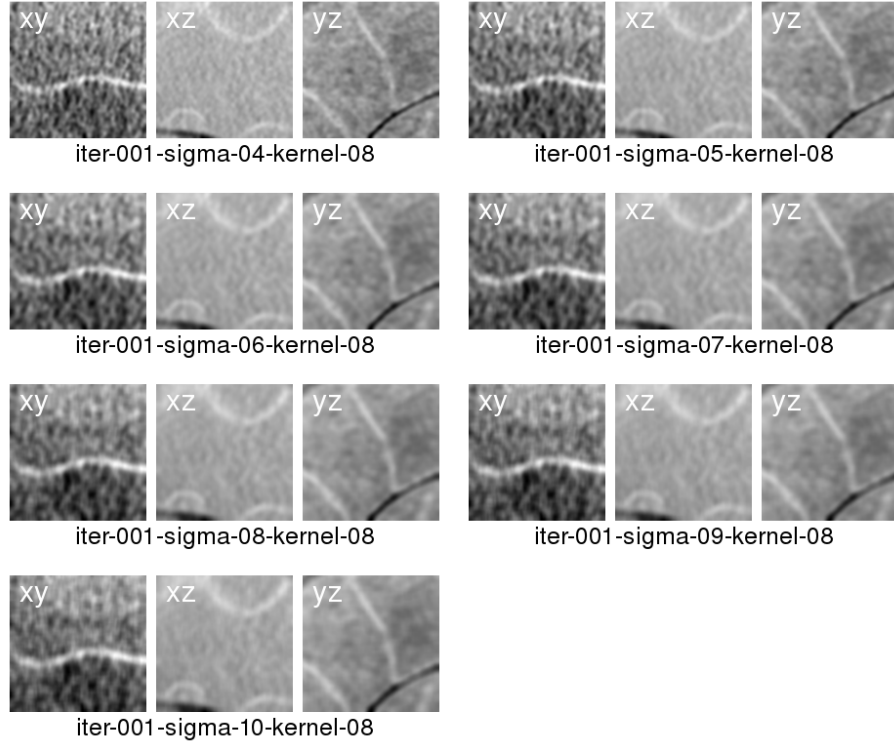


Figure 29: Results of applying Gaussian smoothing with a fixed kernel size of 8^3 and different values of the standard deviation σ within the range $[3, 10]$. Increasing the value of σ contributes to an increase of the blurriness of the result as well as the removal of noise. However, for increasingly large values of σ there is barely any additional impact on the visual result.

D.2 2D NLM

To assess NLM filtering with respect to our paleontological dataset we started using the NLM filter already available in ZIBAmira, which is a GPU-based implementation of NLM as described in [2]. ZIBAmira provides a 2D and a 3D variant of this algorithm. The 2D variant processes volumetric datasets in a two-dimensional fashion on a layer-by-layer basis. In this section the results of the 2D NLM filter with varying parameter settings (see Tab. 3) are presented.

We conducted our analysis in a similar fashion as we did in our investigations before. In each of our experiments, fixed values were chosen for two of the three possible parameters offered by ZIBAmira’s NLM implementation as, for instance, the size of the search window, the size of the neighborhood, and a similarity value. The results of each possible combination of two fixed parameters while changing the value of the third one are presented in Figs. 30–32. Notice how, for this kind of datasets, the similarity parameter has the biggest impact on the result. The figures show only the results of denoising in XY direction—similar results were obtained for the XZ and YZ directions, too.

We also tested the effect of applying the 2D NLM filter with the same parameter values on the same volume in consecutive steps. From the previous experiments, the parameter values that provided the best results were selected, and the filter was consecutively applied up to three times (Fig. 33).

The performance of a single iteration of this filter was also measured (Tab. 4). The quality of the results of a 2D filtering approach, however, were deemed insufficient for our purposes.

Table 3: Parameter values of the 2D NLM filter used in our tests.

Parameter	Value
Filter	Non-local means
Direction	XY
Seach window	$\{11^2, 21^2, 101^2, 201^2\}$
Neirborhood	$\{5^2, 7^2, 11^2\}$
Similarity	$\{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$
Adaptativity	True

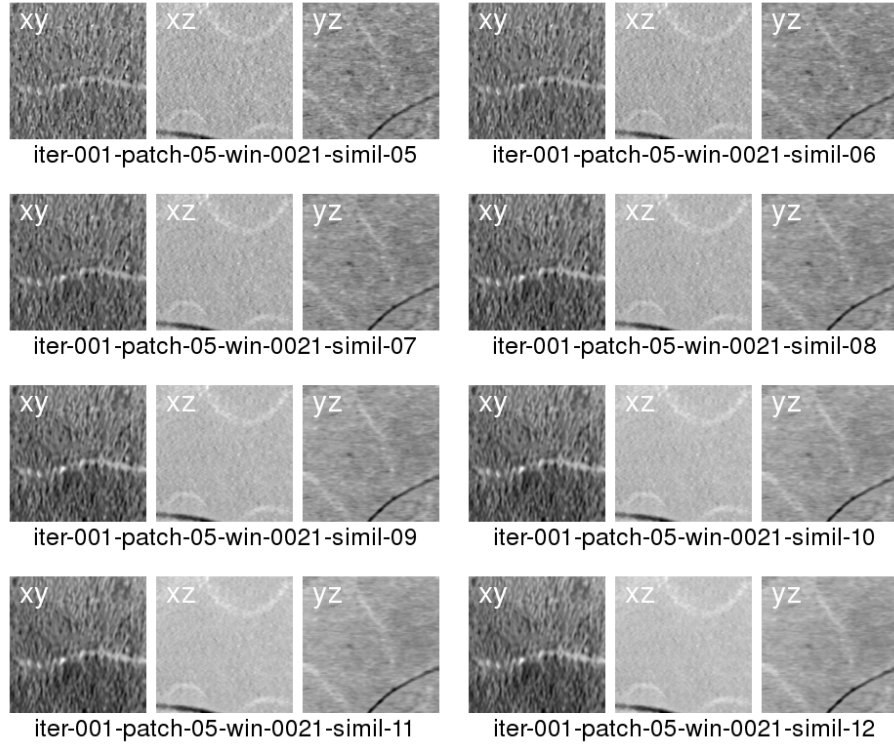


Figure 30: Results of applying ZIBAmira's 2D NLM filter with a neighborhood size of 5^2 , a search window size of 21^2 , and different similarity values in the range $[0.5, 1.2]$. The changes evoked by the similarity value are subtle, but one can see that high similarity values result in blurrier images.

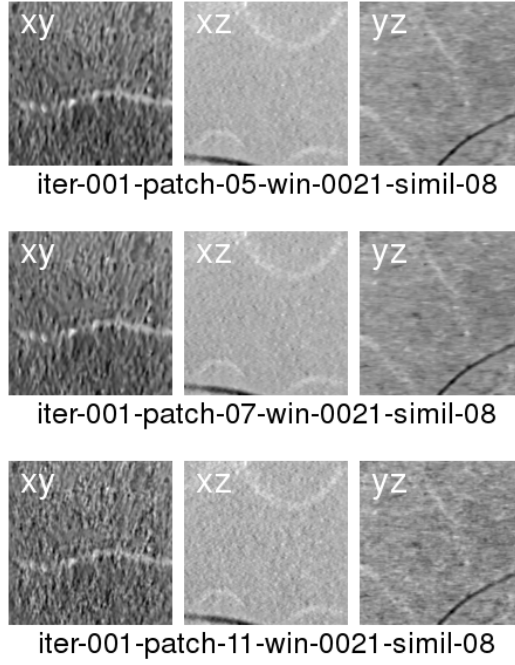


Figure 31: Results of applying ZIBAmira's 2D NLM filter with different values of neighborhood sizes from the set $\{5^2, 7^2, 11^2\}$, a search window size of 21^2 , and a similarity value of 0.8. Larger neighborhoods lead to crisper but noisier results. When the size of the neighborhood increases too much, it becomes difficult to find similarities among them, which in the end degrades the effect of the filter again.

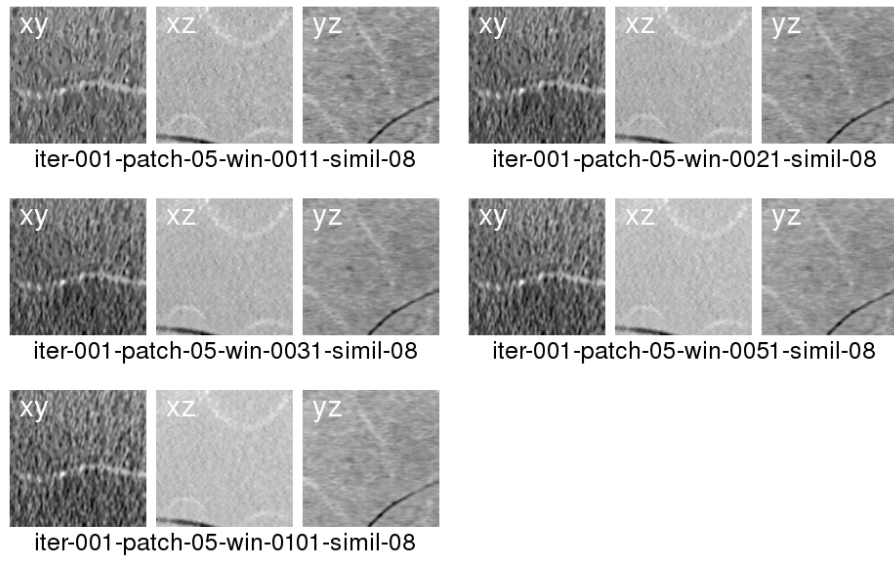


Figure 32: Results of applying ZIBAmira's 2D NLM filter with a neighborhood size of 5^2 , different search window sizes from the set $\{11^2, 21^2, 31^2, 51^2, 101^2\}$, and a similarity value of 0.8. Little to no difference can be seen between the different datasets after visual examination—the result using a window search size of 11^2 seems blurrier than the others.

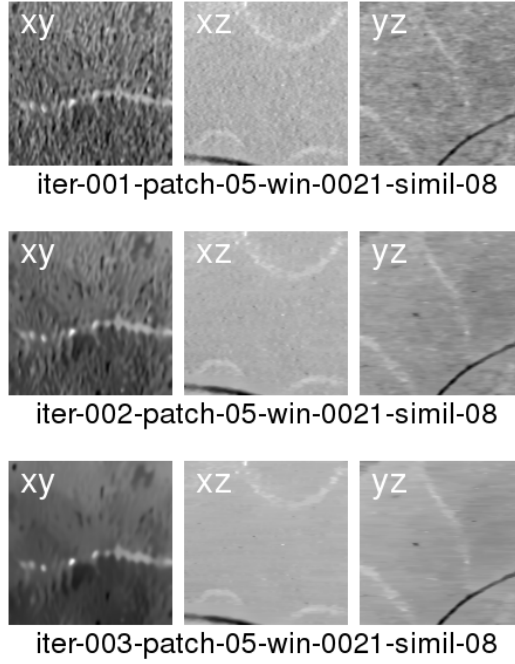


Figure 33: Results of applying ZIBAmira's 2D NLM filter with a neighborhood size of 5^2 , a search window size of 21^2 , and a similarity value of 0.8. From top to bottom, each image set shows the result after applying the filter once, twice, and three times in the same image. In the final iteration, a quite smooth dataset is obtained, with nearly all the noise completely removed. However, some artifacts derived from the direction of filtering (XY plane) are now more noticeable in the planes XZ and YZ. The septum exhibits severe discontinuities that cannot be associated to a natural development, but were introduced by the filter. These discontinuities are even present after the first iteration of the filter.

Table 4: Time consumed by a single iteration of ZIBAmira’s 2D NLM filter using an Nvidia GTX TITAN graphics adapter.

Search window size	Neighborhood size	Time
11^2	5^2	42s
11^2	7^2	44s
11^2	11^2	44s
21^2	5^2	44s
21^2	7^2	45s
21^2	11^2	49s
101^2	5^2	1m 6s
101^2	7^2	1m 25s
101^2	11^2	2m 52s
201^2	5^2	1m 37s
201^2	7^2	2m 42s
201^2	11^2	9m 52s

D.3 3D NLM

In this section we present the results of our investigation of the 3D variant of the NLM filter provided by ZIBAmira. Again different parameter settings were assessed with regard to the filtering effect on the given dataset (see Tab. 5). The parameter space explored in this evaluation is smaller than in the previous tests, since denoising of each dataset can take several minutes. We focused our efforts on these values that already provided better results in the 2D filtering approach (Figs. 34 and 35). We also tested several applications of the filter in consecutive times (Fig. 36).

This variant gives good results in terms of quality, but its computational performance is a major issue (Tab. 6).

Table 5: Parameter values of the 3D NLM filter used in our tests.

Parameter	Value
Filter	Non-local means
Direction	3D
Seach window	$\{11^3, 21^3, 31^3\}$
Neirborhood	$\{5^3\}$
Similarity value	$\{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$
Adaptativity	True

Table 6: Time consumed by a single iteration of ZIBAmira’s 3D NLM on a GTX TITAN graphics adapter.

Search window size	Neighborhood size	Time
11^3	5^3	47s
21^3	5^3	3m 38s
31^3	5^3	45m 52s

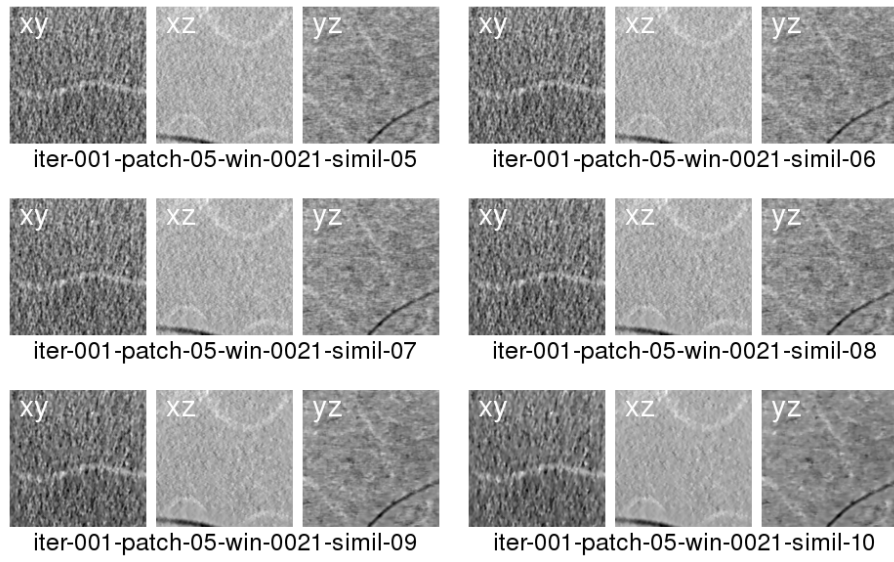


Figure 34: Results of applying ZIBAmira's 3D NLM filter with a neighborhood size of 5^3 , a search window size of 21^3 , and varying similarity values in the range $[0.5, 1.0]$. The same conclusions as in the 2D case can be drawn. The higher the similarity value, the blurrier the result—although in this particular case the blurriness is only visible with the highest similarity value.

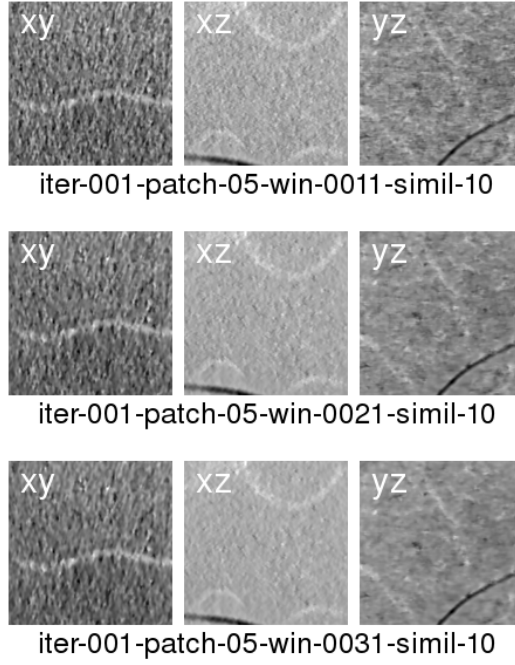


Figure 35: Results of applying ZIBAmira's 3D NLM filter with a neighborhood size of 5^3 , different search window sizes from the set $\{11^3, 21^3, 31^3\}$, and a similarity value of 1.0. As in the 2D case, significant differences due to parameter changes cannot be identified. For the larger window sizes, the results look slightly blurrier and less noisy. However, the increased time to process the dataset does not compensate for the improvement in quality.

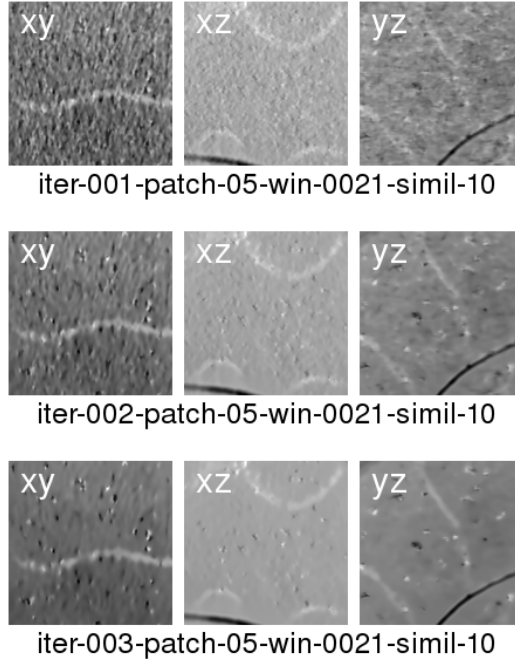


Figure 36: Results of applying ZIBAmira's 3D NLM filter with a neighborhood size of 5^3 , a search window size of 21^3 , and a similarity value of 1.0. From top to bottom, each image set shows the result after applying the filter once, twice, and three times via the same image (i.e., the same slice within the 3D data volume). The artifacts that were present in the 2D case cannot be observed here, and the surface of the inner walls (septa) is nearly continuous. We also see that most of the noise has been removed. The filter interprets some dots or small islands in the dataset as relevant structures, which are clearly noticeable after three iterations. However, these structures do not yield a significant problem, as ZIBAmira provides semi-automatic tools to remove such small islands within a subsequent segmentation process.

E Gaussian kd -tree NLM

We assessed the Gaussian kd -tree NLM using the same approach as with Gaussian smoothing and plain NLM (Tab. 7).

Table 7: Parameter values of the Gaussian kd -tree filter used in our tests.

Parameter	Value
Filter	Gaussian kd -tree NLM
Direction	3D
Neighborhood	$5^3, 7^3, 11^3$
PCA output dimensions	$\{3, 5, 7\}$
Spatial standard deviation	$\{0.5, 0.6, 0.8, 1.0, 5.0, 9.0\}$
Range standard deviation	$\{0.1, 0.3, 0.5, 0.6, 0.8, 1.0\}$

Within our investigations we analyzed the effects of the two standard deviation parameters (spatial and range) on the output of the filter (Figs. 37 and 38).

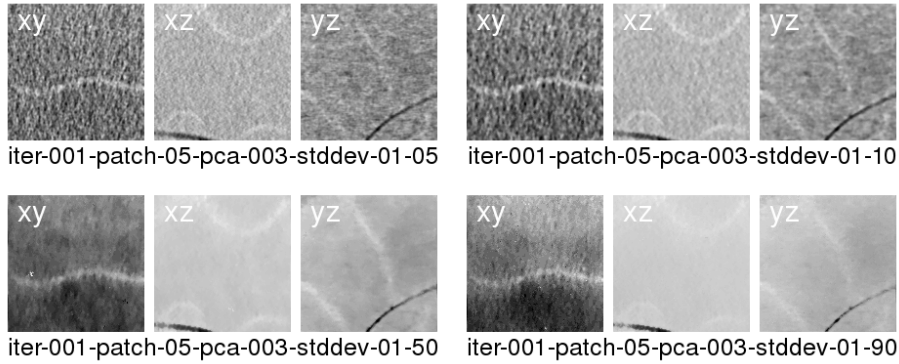


Figure 37: Results of applying a Gaussian kd -tree NLM filter with a neighborhood size of 5^3 , a PCA output dimension of 3, a range standard deviation of 0.1, and different values of the spatial standard deviation from the set $\{0.5, 1.0, 5.0, 9.0\}$. This last parameter has a very strong influence on the results. A higher spatial standard deviation implies a higher level of blurriness and thus a higher effect on denoising. Notice how for a value of σ_s equal or higher than 5.0, most of the noise is nearly gone while the structural information in the images is kept intact.

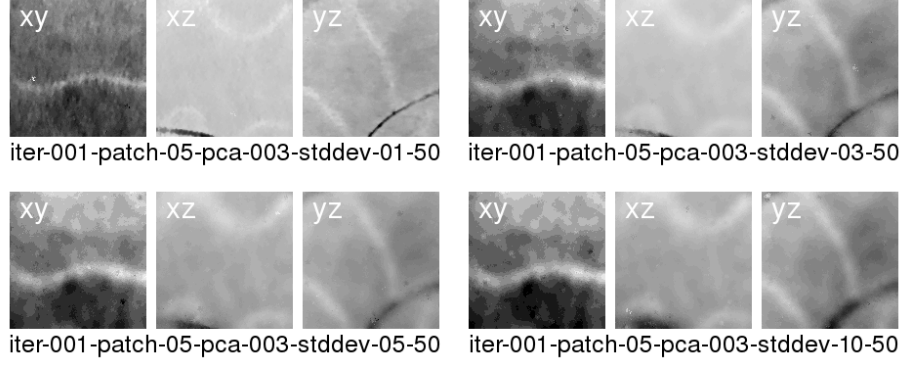


Figure 38: Results of applying a Gaussian kd -tree 3D NLM filter with a neighborhood size of 5^3 , a PCA output dimension of 3, different values of the range standard deviation from the set $\{0.1, 0.3, 0.5, 1.0\}$, and a spatial standard deviation of 5.0. Higher values of the spatial intensity tend to increase the blurriness of the result. When the value is high enough, increasing the value even further has little to no effect. However, for this dataset using high values of σ_r incurs in some loss of structural information, so we propose using lower values.

We also kept the standard deviations fixed while varying the neighborhood size and the PCA output dimensions (Fig. 39). The effects on the performance for different parameter settings can be observed in Table 8.

Table 8: Time consumed by a single iteration of the Gaussian *kd*-tree approach for different combinations of parameter values.

Neighborhood size	PCA output dimensions	Spatial std. deviation	Range std. deviation	Time
5^3	3	1	0.1	32s
5^3	5	1	0.1	33s
5^3	7	1	0.1	34s
5^3	3	5	0.5	42s
5^3	5	5	0.5	46s
5^3	7	5	0.5	48s
7^3	3	1	0.1	1m 18s
7^3	5	1	0.1	1m 17s
7^3	7	1	0.1	1m 19s
7^3	3	5	0.5	1m 31s
7^3	5	5	0.5	1m 35s
7^3	7	5	0.5	1m 36s
11^3	3	1	0.1	13m 33s
11^3	5	1	0.1	13m 38s
11^3	7	1	0.1	13m 41s
11^3	3	5	0.5	13m 55s
11^3	5	5	0.5	14m 00s
11^3	7	5	0.5	13m 58s

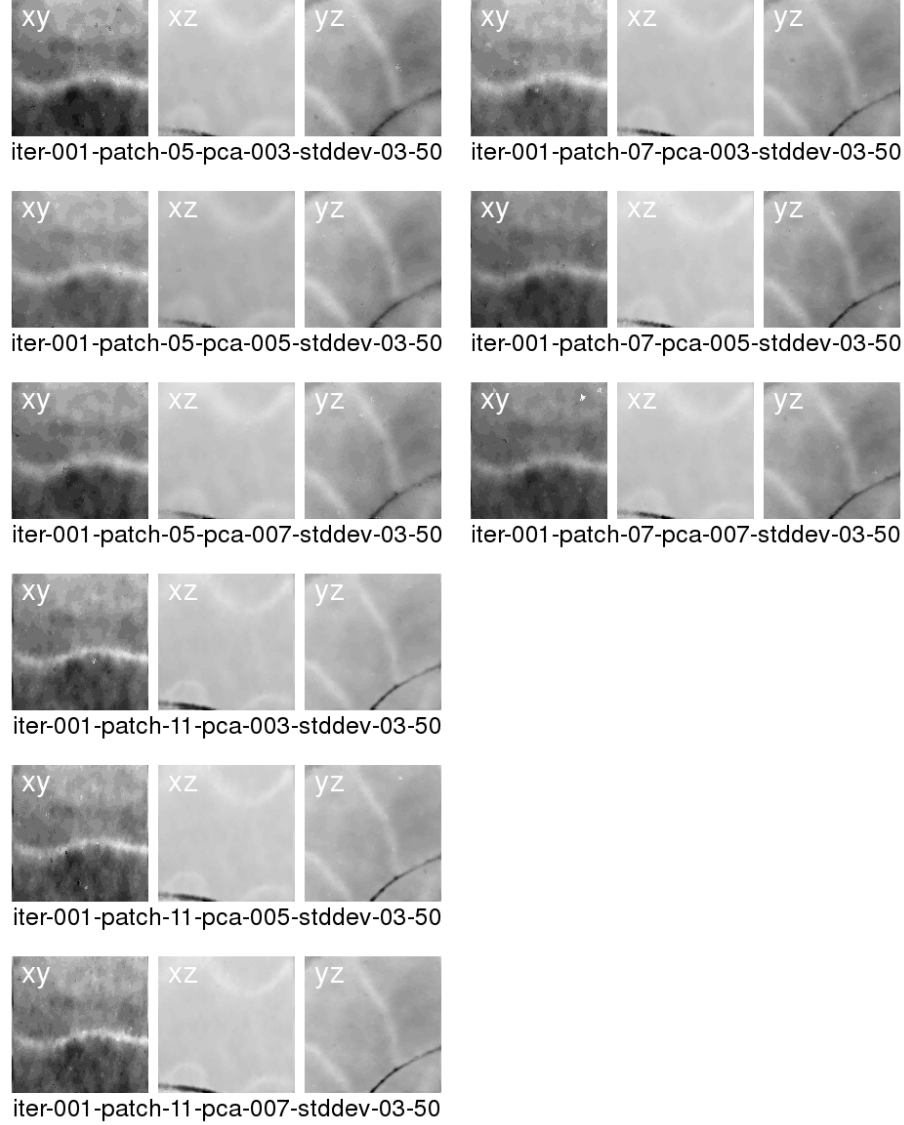


Figure 39: Results of applying a Gaussian *kd*-tree 3D NLM filter with different local neighborhood sizes from the set $\{5^3, 7^3, 11^3\}$, different PCA output dimensions from the set $\{3, 5, 7\}$, a range standard deviation value of 0.3, and a spatial standard deviation of 5.0. For the lowest value of local neighborhood size, varying the number of output dimensions has barely no effect. By increasing the size of the local neighborhood in combination with the number of output dimensions, the resulting images get slightly sharper. This is a similar effect to increasing the size of the local neighborhood in the plain NLM implementation. Notice that this effect stands out even when the number of output dimensions obtained from the PCA is kept constant. This means that having just a few significant components from each local neighborhood (between 3 and 5) is in this application sufficient.

E.1 AM-NLM

The assessment of our GPU implementation of the 3D AM-NLM filter followed similar steps as for the previous filter. The results of altering the spatial and range standard deviations with a fixed value of neighborhood size and output dimensions appear in Figs. 40 and 41, whereas the results for fixed values of the standard deviations appear in Fig. 42.

Table 9: Parameter values of the AM-NLM filter used in our tests.

Parameter	Value
Filter	Adaptive manifolds non-local means
Direction	3D
Local neighborhood	$\{5^3, 7^3, 11^3\}$
PCA output dimensions	$\{3, 5, 7\}$
Spatial standard deviation	$\{0.5, 1.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0\}$
Range standard deviation	$\{0.1, 0.3, 0.4, 0.5, 0.6, 0.7, 1.0\}$

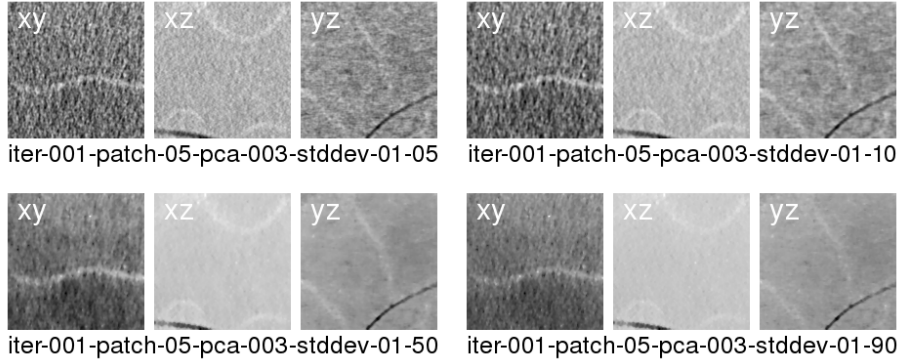


Figure 40: Results of applying an AM-NLM filter with a local neighborhood size of 5^3 , a PCA output dimension of 3, a range standard deviation value of 0.1, and different values of spatial standard deviation from the set $\{0.5, 1.0, 5.0, 9.0\}$. Not surprisingly, this last parameter has a very strong influence on the results, as the algorithm still is a variant of the NLM algorithm in the same way as the Gaussian kd -tree algorithm is. Again, a higher spatial standard deviation implies more blurriness and less noise.

We also studied the performance of our implementation for different parameter values (Tab. 10). The best results are obtained when the number of PCA output dimensions is set to 3 or 4, as in this case, the implementation is using the texture processing pipeline of the GPU. Furthermore, using 4 dimensions leads to slightly better performance than using 3, probably due to a better memory alignment of the data.

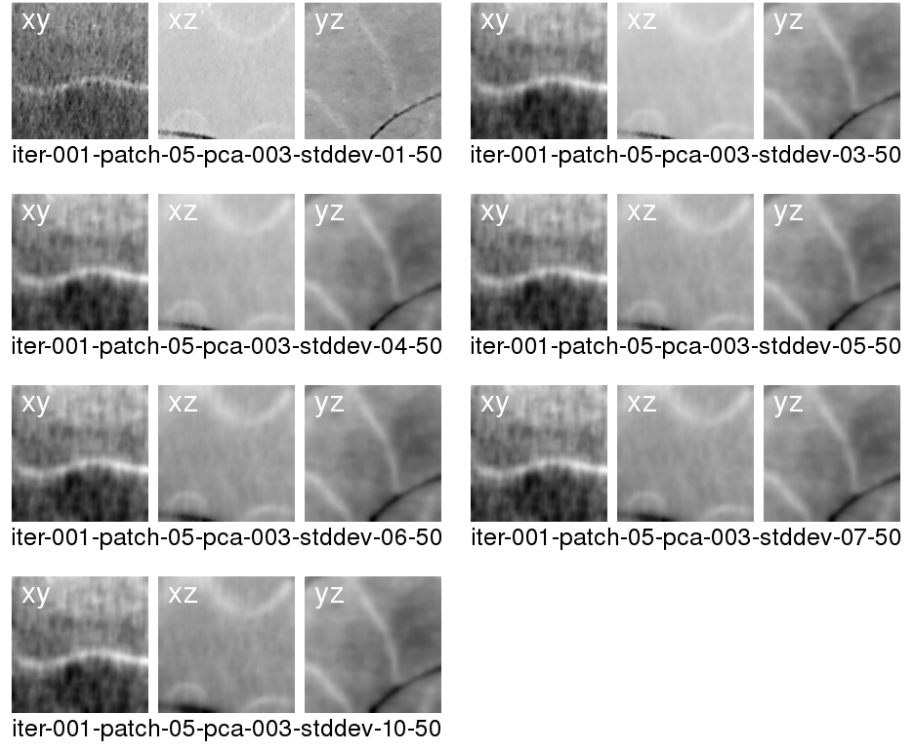


Figure 41: Results of applying an AM-NLM filter with a local neighborhood size of 5^3 , a PCA output dimension of 3, different values of the range standard deviation from the set $\{0.1, 0.3, 0.4, 0.5, 0.6, 0.7, 1.0\}$, and a spatial standard deviation value of 5.0. The range standard deviation has a noticeable effect for small values. Once the value is large enough, the result becomes blurrier and less noisy. However, from this point on, no matter how high this value is chosen, the filtering result does not significantly change.

Table 10: Time consumed by a single iteration of the AM-NLM filter for different combinations of parameter values.

Neighborhood size	PCA output dimensions	Spatial std. deviation	Range std. deviation	Time
5^3	3	1	0.1	9s
5^3	4	1	0.1	10s
5^3	5	1	0.1	13s
5^3	3	5	0.5	9s
5^3	4	5	0.5	8s
5^3	5	5	0.5	13s

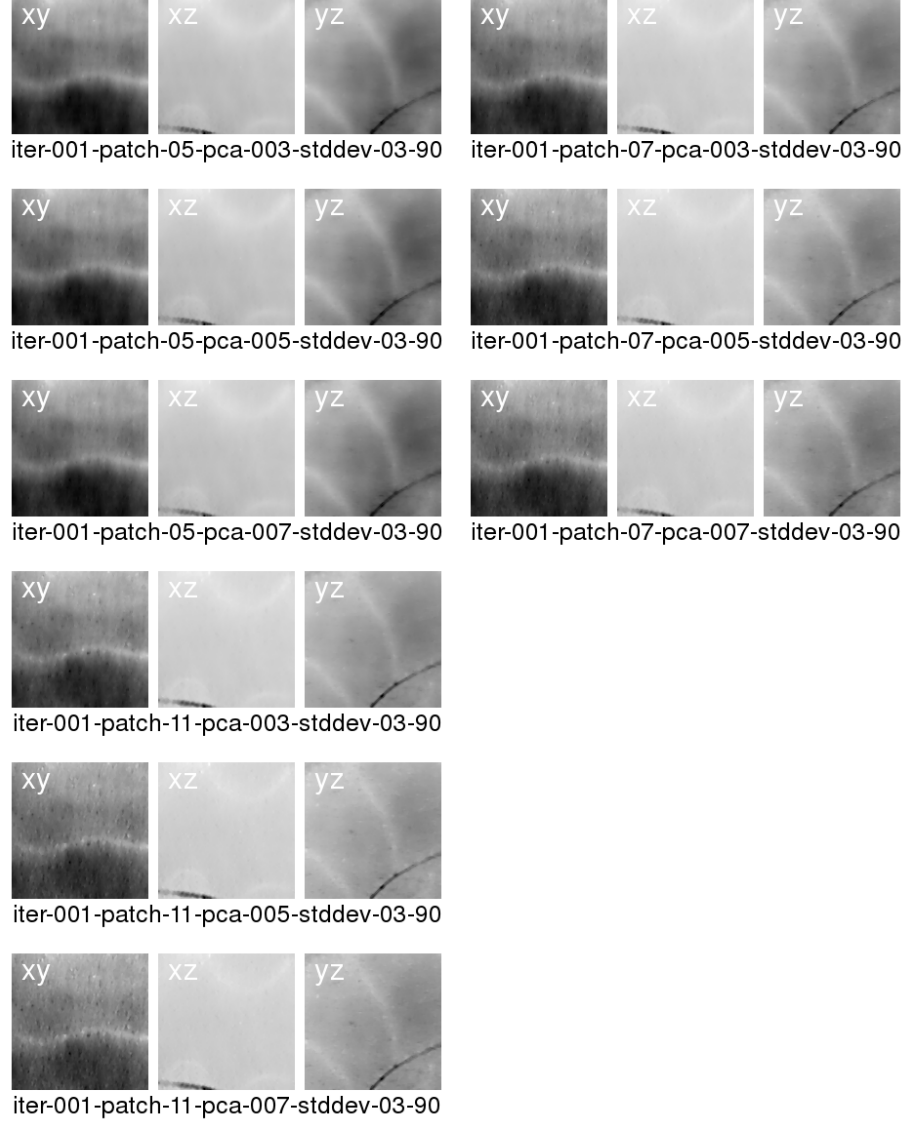


Figure 42: Results of applying an AM-NLM filter with different local neighborhood sizes from the set $\{5^3, 7^3, 11^3\}$, different PCA output dimensions from the set $\{3, 5, 7\}$, a range standard deviation value of 0.3, and a spatial standard deviation value of 9.0. Although the effect is not as obvious as with the Gaussian *kd*-tree approach, it is noticeable that results get sharper with increasing size of the local neighborhood. The size of the output dimensions, however, does not seem to exert a significant effect on the results.

E.2 k-means clustering

As an example on how good septal surfaces can be segmented from a denoised dataset, we performed a series of experiments using the k -means clustering algorithm. One of the results from denoising with AM-NLM was chosen, and an adaptive histogram equalization (CLAHE) was applied to it. The clustering filter was then applied to the equalized result (Tab. 11).

Table 11: Parameters to denoise, equalize, and split in clusters that have been used in our tests.

Parameter	Value
Denoising	AM-NLM
Local neighborhood size	$5 \times 5 \times 5$
PCA output dimensions	3
Range standard deviation	0.5
Spatial standard deviation	5.0
Equalization	CLAHE
Contrast	21
Clustering	k -means
Number of iterations	{5, 10, 15, 20, 25}
Number of clusters	{2, 3, 4, 5, 6, 7, 8, 9, 10}
Patch size	{ 3^3 , 5^3 }

There are several parameters of the clustering filter that can be adjusted, so we proceeded as in previous experiments. First, the number of iterations and the patch size was fixed to a high value, under the assumption that the best result is always obtained the higher the number of iterations and the larger patch size is (Fig. 43). The next step was to find which patch size gives the best results (Fig. 44). Finally, the effect of changing the number of iterations was explored (Fig. 45).

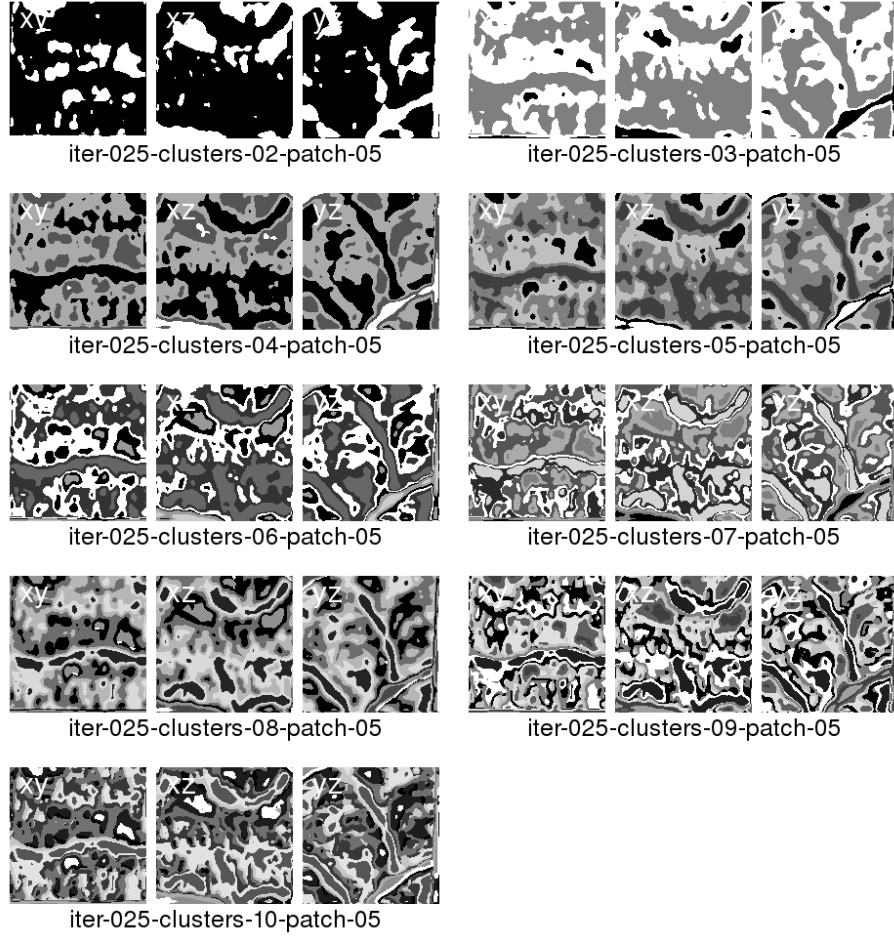


Figure 43: Results of applying a k -means clustering filter on a denoised dataset. The number of iterations is set to 25, the number of clusters is in the range $[2, 10]$, and the patch size is set to 5^3 . It can be seen that we need at least four or five clusters to identify the regions that belong to the septa—the higher the number of clusters, the thinner becomes the septal region. Unfortunately, the cluster segmenting the septa also contains other regions which obviously belong to the chamber. In any case, from the experiments we conclude that eight clusters yield a good compromise.

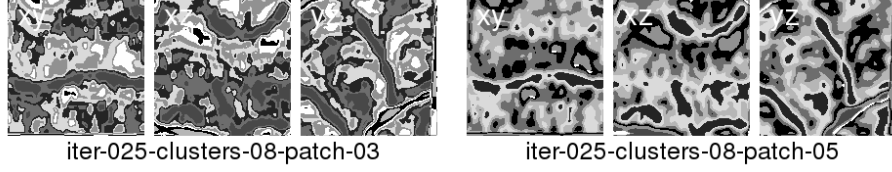


Figure 44: Results of applying a k -means clustering filter on a denoised dataset. The number of iterations is set to 25, the number of clusters is set to 8, and the patch size takes values from the set $\{3^3, 5^3\}$. One can clearly see that a patch size of 5^3 yields a better result.

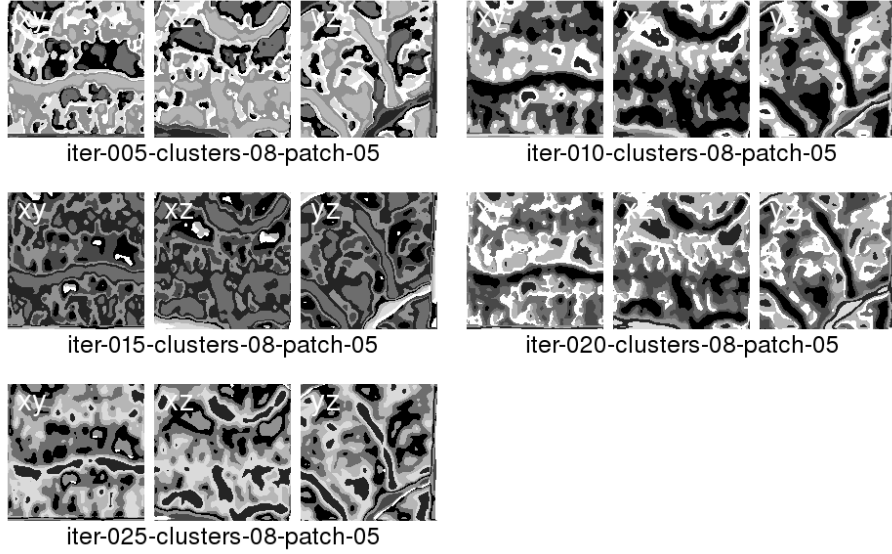


Figure 45: Results of applying a k -means clustering filter on a denoised dataset. The number of iterations takes values from the set $\{5, 10, 15, 20, 25\}$, the number of clusters is set 8, and the patch size is set to 5^3 . One can see that 20 iterations seem to be enough to get a reasonable result where the septa is identified with one of the clusters.

References

- [1] Adams et al. Gaussian kd-trees for fast high-dimensional filtering Proceedings of ECCV 28(3). 2009.
- [2] Buades et al. A review of image denoising algorithms, with a new one. Multiscale modeling & simulation 4(2):490–530. 2005.
- [3] Chan et al. Monte Carlo Non-Local Means: random sampling for large-scale image filtering. IEEE Transactions on Image Processing 23(8). 2014.
- [4] Cunningham et al. A virtual world of paleontology. Trends in Ecology & Evolution 29(6). 2014.
- [5] Chen et al. Real-time edge-aware image processing with bilateral grid. ACM Transactions on Graphics 26(3). 2007.
- [6] Courant and Hilbert. Methods of Mathematical Physics (vol. 1 p.50). Interscience Publishers, New York. 1953.
- [7] Deledalle et al. Non-local methods with shape-adaptive patches (NLM-SAP) Journal of Mathematical Imaging and Vision 43(2). 2012.
- [8] <http://www.drdobbs.com/cpp/logging-in-c/201804215>
- [9] Ketcham and Carlson. Acquisition, optimization and interpretation of x-ray computed tomographic imagery: Applications to the geosciences. Computers and Geosciences 27(4). 2001.
- [10] Gastal and Oliveira. Adaptive manifolds for real-time high-dimensional filtering. ACM Transactions on Graphics 31(4). 2012.
- [11] Gastal and Oliveira. Shared sampling for real-time alpha matting Computer Graphics Forum 29(2). 2010.
- [12] Manjón et al. Adaptive non-local means denoising of MR images with spatially varying noise levels. Journal of magnetic resonance imaging 31(1). 2010.
- [13] Mastin. Adaptive filters for digital image noise smoothing: an evaluation. Computer Vision, Graphics, and Image Processing 31(1). 1985.
- [14] Paris and Durand. A fast approximation of the bilateral filter using a signal processing approach. Technical Report. Computer Science and Artificial Intelligence Laboratory. 2006.
- [15] Perona and Malik. Space-scale and edge detection using anisotropic diffusion. IEEE Transactions on Pattern Analysis and Machine Intelligence 12(7). 1990.
- [16] <http://www.raosoft.com/samplesize.html>

- [17] Sutton. Tomographic techniques for the study of exceptionally preserved fossils. *Proceedings of The Royal Society B*, 275. 2008.
- [18] Tasdizen. Principal components for non-local means image denoising *Proceedings of International Conference on Image Processing*. 2008.
- [19] Tomasi and Manduchi. Bilateral filtering of gray and color images. *6th International Conference on Computer Vision*. 1998.
- [20] Ukeneder et al. Computed tomography and laser scanning of fossil cephalopods (Triassic and Cretaceous). *Kataloge des Oberösterreichischen Landesmuseums, Neue Serie* 157. 2014.
- [21] <http://amira.zib.de>