Structured Low-Rank Matrix Factorization for Point-Cloud Denoising

Kripasindhu Sarkar\textsuperscript{1,2}, Florian Bernard\textsuperscript{3,4}, Kiran Varanasi\textsuperscript{1}, Christian Theobalt\textsuperscript{3,4}, and Didier Stricker\textsuperscript{1,2}

\textsuperscript{1}DFKI Kaiserslautern, \textsuperscript{2}TU Kaiserslautern, \textsuperscript{3}MPI Informatics, \textsuperscript{4}Saarland Informatics Campus

Abstract

In this work we address the problem of point-cloud denoising, where we assume that a given point-cloud comprises (noisy) points that were sampled from an underlying surface that is to be denoised. We phrase the point-cloud denoising problem in terms of a dictionary learning framework. To this end, for a given point-cloud we (robustly) extract planar patches covering the entire point-cloud, where each patch contains a (noisy) description of the local structure of the underlying surface. Based on the general assumption that many of the local patches (in the noise-free point-cloud) contain redundant information (e.g. due to smoothness of the surface, or due to repetitive structures), we find a low-dimensional affine subspace that (approximately) explains the extracted (noisy) patches. Computationally, this is achieved by solving a structured low-rank matrix factorization problem, where we impose smoothness on the patch dictionary and sparsity on the coefficients. We experimentally demonstrate that our method outperforms existing denoising approaches in various noise scenarios.

1. Introduction

Point-clouds are a ubiquitous representation of 3D data that are obtained as the output of a 3D scanner (e.g. range scanners, Kinect etc.), or are the end-result of reconstruction algorithms (e.g. structure-from-motion, or KinectFusion [30]). A well-known issue with sensor-based data acquisition is that the obtained measurements may be noisy, so that they do not faithfully reproduce the measured real-world object. In order to deal with this, various denoising methods are designed for different kinds of data. A common assumption of many methods is that one deals with Gaussian noise—an assumption which does not hold in most real-world applications. Moreover, albeit the fact that point-clouds describe geometric data, they lack a regular structure or topology, which makes their denoising more challenging than for structured data that is defined over a regular grid, such as images.

Learning-based denoising methods have been demonstrated to be extremely successful in numerous settings, particularly for image data [17]. Many of them are based on convolutional neural networks (CNNs) (e.g., [22, 43]). They require that the data that is to be processed is amenable to convolution operations, i.e. the data has a well-defined topology which ideally is regular. Since images exhibit these desirable properties, applying convolutions to images is straightforward. However, defining such operations on general geometrical objects is more challenging and is an active area of research (e.g for meshes [8, 29] or for point-clouds [32, 33, 5]). Due to the lack of a well-defined topology, applying such techniques to point-clouds is difficult, so that often the spatial neighborhood information is not used.

In contrast, we propose a simple method that exploits the spatial neighborhood in point-clouds and does not require any training data. We perform point-cloud denoising based on a matrix factorization formulation, where we impose particular attention to finding a suitable fixed-length and regular representation of the point-cloud, such that the above-described problems regarding irregular structure and the lack of a topology are circumvented. To this end, similarly to [16, 36], we represent a given point-cloud by a collection of (local) displacement maps, each of which encode the local structure of the point-cloud within a small patch. Our novel contribution is a strategy for extracting such patches that is robust to noise. In contrast to the prior works, we do not require the patches to be too small [16], or require a semi-automatic mesh quadrangulation step [36]. We learn a dictionary that is able to denoise the extracted displacement maps, such that the denoised point-cloud is obtained from the reconstruction of the denoised local patches. An important and desirable property of our approach is that it works under a wide range of different noise characteristics, as the dictionary learning procedure implicitly deals with the noise model present in the data.

Main Contributions: The technical contributions of our method are as follows:

- Given an input 3D point-cloud, we present a method to robustly extract a collection of patches covering the entire point-cloud, where each patch describes the lo-
We propose a formulation of structured matrix factorization for these 3D patches, by extending it to handle missing data, where the patches may contain unoccupied bins.

- We handle a wide range of noise characteristics including real world 3D scanner noise, unlike previous methods which are limited to normally distributed noise. We demonstrate superior denoising results as compared to other methods.

### 2. Related Work

**Mesh Denoising:** The problem of denoising is directly connected to building a signal processing framework over a domain on which the signal is defined [38, 39]. Zhong et al. [44] propose a method for 3D shape completion by imposing sparsity on the Laplacian eigenbasis of the shape. The more recent approaches for denoising use machine learning models such as CNNs that are trained on a pre-registered dataset of meshes [9]. However, to be effective, they require training data of meshes that are pre-registered to a common topology, which is a challenging problem in its own right.

**Global models for point-cloud denoising:** The alternative is to consider the 3D point-clouds directly, with no structure and grid pattern given for the domain on which this signal is defined. Unlike with the mesh input, these methods cannot make use of a surface normal for denoising. Nevertheless, it is possible to learn denoising models from unstructured point-clouds [32, 33, 1]. These models are sensitive to a global transformation of the point-cloud, so the 3D alignment problem needs to be addressed by a dedicated spatial transformer network [32], which can be further enhanced by learning local features [33]. Despite their versatility, these methods ignore the fact that most real world 3D point-clouds are sampled from objects which can be represented as manifold surfaces at the local level. Alternatively, a dynamic graph CNN model can be learned that adapts the topology over time [40], but this requires a dynamic graph update of a global model. In contrast to these point-based methods which try to estimate a global model for the entire point-cloud, our method estimates only local models (patches) around points, which are simpler to learn. Moreover, local patch models are well-suited for denoising, as noise affects the shape in a local manner.

**Local methods for point-cloud denoising:** A good strategy of denoising point-clouds is to fit a smooth target surface using the local neighborhood and project points on this smooth surface for obtaining the denoised point-cloud [21, 14, 24, 28, 3, 4]. Out of them, the most popular strategy is the use of a weighted least squares measure biased towards the local point of computation to fit a continuous function—also known as Moving Least Squares methods [3, 4, 21, 24]. On a similar line, [13, 15] use Voronoi-based computations for the surface reconstruction, while [10] uses octree decomposition followed by a modified ‘meshless’ Laplacian smoothing for the purpose of denoising. All these methods perform very well on denoising point-clouds, but often remove important details of the underlying surface. This is mostly because of the local nature of the problem being solved, which does not consider the information coming from different regions of the surface. This motivates us to use a non-local method on local feature descriptors (patches), which is further discussed in the subsequent paragraphs.

**Patch-based methods:** Patch-based methods have been developed first for 2D image processing, where it is assumed that the 2D image has a locally sparse representation in the transformed domain. These algorithms can be categorised into dictionary based [2, 27, 26] and BM3D (Block-matching and 3D filtering) based [11, 12, 25] methods. In BM3D, similar 2D patches are further grouped into 3D groups, which enhances sparsity. Rosman et al. [34] apply this idea towards the denoising of 3D shapes by matching the patches on a collaborative patch representation. Our method does not require this a priori matching step.

Digne et al. [16] proposed a method for point-cloud compression by learning a dictionary of circular patches. They assume that the local patches are sufficiently small (where the shape is parameterizable to a unit disc). In contrast to this work, (i) we address the problem of denoising instead of compression which makes the patch extraction procedure more challenging, (ii) our local reference frame for patches is robust to noise and outliers (in comparison to simple PCA), (iii) our dictionary learning framework handles missing data, and (iv) as a result of aforementioned differences, our patch size is much larger in order to have a meaningful patch description in the presence of noise.

Sarkar et al. [36] are able to address larger patches (in comparison to [16]), by requiring a 3D mesh input along with the point-cloud, and computing local patches based on quadrangulation of its low resolution mesh. Their method learns a sparse dictionary for 3D shape completion and denoising. In a later work [37], they learn a denoising autoencoder neural network model by relying on the same mesh quadrangulation for computing and orienting the patches. Both of these approaches are limited by the assumption of mesh quadrangulation, which may not be possible to compute on general point-clouds.

A fundamental problem with applying patch-based methods to point-clouds is that the patch placement and orientation can be ambiguous and cannot always be judged based on the local structure of the point-cloud alone. The previous approaches worked around this problem by assum-
3. Point-Cloud Denoising

Overview: We consider the task of denoising a point-cloud that comprises points that are noisy samples of an (unknown) surface. In order to process a given point-cloud, we represent the entire point-cloud by means of (small) planar patches, where each patch describes the local structure of the point-cloud in terms of a displacement along the patch normal. By associating with each patch a rigid-body pose, i.e., a location and an orientation, one can reconstruct the point-cloud from its patch-based representation. As such, in order to perform denoising, we first extract patches from the noisy point-cloud, we then denoise the patches using dictionary learning, and eventually reconstruct the point-cloud (right).

3.1. Notation

For a given integer $n \in \mathbb{N}$, we define $[n] := \{1, \ldots, n\}$. Let $A$ be an $m \times n$ matrix, and let $i \in [n]$ and $j \in [m]$. By $A_{i,:}$ we denote the $m$-dimensional column vector that is formed by the $i$-th column of $A$, for which we also use the short-hand $A_i$. Moreover, we denote by $A_{:,i}$ the $n$-dimensional row vector that is formed by the $j$-th row of $A$. The operator $\odot$ denotes the Hadamard product. For a point $x \in \mathbb{R}^3$, we use $\mathcal{N}_\epsilon(x) := \{y : \|x - y\| < \epsilon\}$ to denote the $\epsilon$-neighborhood (or $\epsilon$-ball) of $x$, where $\|\cdot\|$ is the $\ell_2$-norm. For a given matrix $Z \in \mathbb{R}^{3 \times n}$ containing $z$ points in 3D, we define $\mathcal{N}_\epsilon^Z(x) := \{Z_i : i \in [z], \|x - Z_i\| < \epsilon\}$ to denote the $\epsilon$-neighborhood of $x$ among the points in $Z$.

3.2.1 Seed Point Selection

We select a subset of points from the point-cloud $X$, written in the matrix $S \in \mathbb{R}^{3 \times s}$, which forms an evenly distributed and sparse representation of $X$. The $s$ points of $S$ are used as seed points for the computation of the patches, which is based on voxel-based downsampling method with grid cell length of $d$. In total, this results in $s \ll n$ seed points, which are then stored in the matrix $S$. The parameter $d$ implicitly controls the density of the patches that are to be computed. It should be chosen (together with the neighborhood radius $r$) in such a way that all points in the point-cloud are contained in the union of the We $r$-neighborhood of the seed points. Formally, this can be expressed as

$$\forall j \in [n] \exists i \in [s] : X_j \in \mathcal{N}_r^X(S_i).$$

While the condition $d < \sqrt{2r}$ ensures that property (1) holds, in practice we choose $d$ to be have a similar value as $r$ in order to achieve a denser covering with the patches. The parameter choices are explained in Section 4.

3.2.2 Robust Patch Extraction

Next, we describe how to robustly extract patches after the seed points have been selected. For the sake of a simple explanation we use square patches (note that disk-shaped patches could also be considered). To this end, for each seed point $S_i \in \mathbb{R}^3$ we extract the patch $P_i$ that covers a square with side length $l = \sqrt{2r}$. The patch $P_i = (t_i, R_i, X_i)$ is represented as a 3-tuple that describes the local structure of
the point-cloud within the \( r \)-neighbourhood \( \mathcal{N}_r(S_i) \) of \( S_i \). Here, \( t_i \) is the location of the center of the patch, \( R_i \) is its orientation, and \( Y_i \in \mathbb{R}^{m \times m} \) is the displacement map. For reasons that will become clear shortly, the seed point location \( S_i \) does not necessarily coincide with patch center \( t_i \). A significant advantage of the displacement map representation compared to the original point-cloud is that now the \( Y_i \) constitute a fixed-length matrix representation of the local surface structure, such that they are amenable to simple linear algebra operations.

**Robust selection of orientation and location:** In order to obtain a patch that aligns well with the underlying surface—even under significantly noisy point-clouds—we propose to use a robust strategy that makes use of a RANSAC-like sampling strategy [18], which is outlined in Alg. 1. In order to determine whether a point is an inlier for a given patch, we check whether this point is within a given distance to the square patch. By considering the point-to-patch distance (line 8), in contrast to the point-to-plane distance, one can avoid the problem that a patch is fitted in such a way it cuts through the object that is described by the underlying surface (cf. Fig. 5). After the best patch, i.e. the one that leads to the largest number of inliers, is determined, the patch orientation is computed based on principal component analysis (PCA), and the patch center is set to the mean of the inlier points. If the number of inliers are less than a threshold, the seed point and the neighborhood are discarded, making our method naturally robust to noisy outliers.

**Displacement map computation:** Given the computed position \( t_i \) and the orientation \( R_i \) of the patch, we compute the fixed-length displacement map from all neighboring points of the seed point. We represent the \( r \)-neighborhood \( \mathcal{N}_r(S_i) \) of the seed point \( S_i \) in the reference frame defined by \( (t_i, R_i) \), and project all points onto the square patch, and then sample the patch on an \( m \times m \) grid of size \( l = \sqrt{2}r \). The displacement value for each bin is then computed as the median of all “z-coordinates” of the points that fall in that particular bin. For all \( p, q \in \mathbb{N} \), the “z-coordinates” denote the length of displacements from the center point \( a_{pq}^i \in \mathbb{R}^3 \) of the bin at position \( (p, q) \) along the patch normal (cf. Fig. 2). In addition, for each point \( X_j \) of the point-cloud, we keep track of the information to which patch and to which bin of each displacement map it falls during the patch computation. This correspondence information is used for the reconstruction of the shapes from the patches. In order to keep track of this correspondence information, for each \( j \in [n] \) we define the list of \( k_j \in \mathbb{N} \) index triplets \( I_j \in \mathbb{R}^{3 \times k_j} \), where each column \( (I_j)_k \in [s]\times[m] \times [m] \) for \( k \in [k_j] \) contains the patch index \( i \in [s] \) in the first row, and the corresponding bin index \( [p, q]^T \in [m] \times [m] \) in the second and third rows. Note that a point \( X_j \) can belong to multiple patches so that \( k_j \geq 1 \). In overall, given the point-cloud \( X \) as an input, along with the parameters \( d, r \) and \( m \), we compute the patch set \( \{ P_i : i \in [s] \} \) and the set of correspondence information \( \{ I_j : j \in [n] \} \). Note that some of the patch bins may be unoccupied, i.e. there is no point \( X_j \) that falls into such a bin, which particularly occurs in areas of edges and corners.

### 3.2.3 Point-Cloud Reconstruction

Next, we briefly describe how we can reconstruct the point-cloud given the patch set \( \{ P_i : i \in [s] \} \) and the set of correspondence information \( \{ I_j : j \in [n] \} \). To this end, for all \( m^2 \) bins we compute the global bin position \( a_{pq}^i \in \mathbb{R}^3 \) in all \( s \) displacement maps, which we also refer to as anchor points. Then, we obtain each of the \( i \in [k_j] \) reconstructed points as \( a_{pq}^i + (Y_i)_{pq}n_i \), where \( n_i \in \mathbb{R}^3 \) is the normal of the \( i \)-th patch (which is determined by \( R_i \)), and \( [p, q]^T = (I_j)_k \) for all \( k \in [k_j] \).

### 3.3. Denoising by Dictionary Learning

In this section we describe the dictionary learning procedure that is used for denoising the collection of displacement maps \( \mathcal{Y} = \{ Y_1, \ldots, Y_s \} \). An overview of the approach is shown in Fig. 3. A theoretical motivation for using such formulation is provided in the supplementary material.

#### General matrix factorization:

Let \( y_i := \text{vec}(Y_i) \in \mathbb{R}^{m^2} \) be the vectorization of \( Y_i \), and let \( Y = [y_1, \ldots, y_s] \in \mathbb{R}^{m^2 \times s} \) be the matrix that contains the vectorized displacement maps. With that, finding a low-dimensional subspace that is a good approximation of the noisy \( Y \) can be phrased as the general matrix factorization problem

\[
\min_{\Phi, A} \ell(Y, \Phi A) + \Omega(\Phi, A),
\]
noisy displacement
larizer is given by
formations with spatially localized support, a similar set of
Φ
ness as well as
Φ
A
on the one hand, it shall impose sparsity on the coefficients
Φ
such that the error when approximating
placement map matrix
Y
is a binary matrix that masks out unobserved data in the dis-ace, some bins in the displacement maps of the extracted
pute the least-squares error for those elements in
Y
are given by a weighted sum of the dictionary atoms
{A_i}
In the following we will further specify the loss function and the regularizer.

Matrix Factorization for Denoising: Due to appealing theoretical properties regarding optimality guarantees, as well as a promising performance in various applications (e.g. [6]), for tackling Problem (2) we build upon the structured low-rank matrix factorization framework by Haeffele et al. [20]. For the sake of notational convenience, and w.l.o.g., we assume that
Y
is zero (if this is not the case, we simply subtract the

\Omega(\Phi) = \lambda \sum_i \|\Phi_i\|_2 \|\mathbf{A}^T_i\|_1, \tag{4}

where for
y \in \mathbb{R}^{m \times 2}
and
z \in \mathbb{R}^s
the norms \| \cdot \|_2
and \| \cdot \|_1
are defined as

\|y\|_2 = \lambda_2 \|y\|_2 + \lambda_\mathbf{E} \|E y\|_2,

\|z\|_1 = \lambda_1 \|z\|_1. \tag{5}

The scalars \lambda, \lambda_2, \lambda_\mathbf{E}
and \lambda_1
are non-negative weights that are used to specify the relative importance of the overall regularization term, the \ell_2-norm of the factors, the smoothness of the factors, and the sparsity of the coefficients, respectively. The matrix \mathbf{E}
is the incidence matrix of the 4-neighbourhood graph of the \( m \times m \) grid of the patch, such that \( \|E \cdot \|_2 \) can be seen as a graph-based (semi)-norm that takes the spatial connectivity of the patch bins into account. Its main purpose is to ensure that the learned dictionary atoms \Phi_1, \ldots, \Phi_r
are spatially smooth. We point out that if spatial smoothness is undesirable, one can set \lambda_\mathbf{E}=0.

Once we have found \Phi
and
A,
we obtain the estimated matrix of clean displacement maps as
\mathbf{Y} = \Phi \mathbf{A} \in \mathbb{R}^{m \times s},
which are then used to reconstruct the denoised point-cloud.
Please refer to the supplementary material for the motivation for using this particular form of the regularizer in (4).

Optimization: In order to optimize Problem (2), we use a block-coordinate descent approach [20]. To this end, the variables \Phi
and
A
are updated alternatingly, where for each variable a gradient descent step of the loss function is conducted first, followed by a proximal step for the regularizer. The computation of the proximal operators of the regularizer is described in [6].

4. Experiments

4.1. General Denoising Settings

Dataset: For evaluating our denoising algorithm, we use the models from [36], which include common meshes like Bunny, Fandisk, Milk Bottle, Baseball and several meshes with fine surface detail such as shoe soles (Supernova, Terrrex, Wander, Leather-Shell) and Brain.

Patch computation: To work with similar parameters across all models, we scale the models so that the bounding box corresponds to a unit cube and sample 1000k points from the surface and add different types of noise. We use a neighborhood radius \( r = 0.03 \) (so that the patch length is \( l \approx 0.042 \)) and \( m = 16 \) as patch dimension for all the models. The value for the radius was selected in such a way that the patches are significantly larger compared to the noise level (e.g. Gaussian noise with \( \sigma = 0.0075 \)). Also we have chosen the seed radius \( d \) the same as \( r \) to obtain overlapping patches. Ideally, the patch length should be chosen such that the \( r \)-neighbourhood contains only points within a topological disk on the surface, which restricts it to be less than the...
Figure 4. Qualitative results of our method and a comparison with other denoising methods under Gaussian noise. From left to right we show the following point-clouds: ground truth, noisy, denoised by plane-based method, denoised by [10], denoised by [4], and denoised by our method. In the magnification of the high-curvature ear region of the bunny it can be seen that qualitatively our denoising method achieves the sharpest results that appears closest to the ground truth. In addition, as shown in Table 1, quantitatively our method outperforms the other approaches.

<table>
<thead>
<tr>
<th>noisy</th>
<th>plane</th>
<th>[10]</th>
<th>[4]</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.93e-03</td>
<td>3.21e-03</td>
<td>6.46e-03</td>
<td>2.40e-03</td>
<td>1.60e-03</td>
</tr>
</tbody>
</table>

Table 1. Summary of the results of denoising 9 shapes (Section 4.1 - Eg. Bunny, Fandisk, Supernova, etc.). The values represent the mean of RMS point-to-mesh error (relative to bounding box) of different algorithms for Gaussian noise with $\sigma = 0.0025$, 0.005 and 0.075, for 10 shapes. For exact result of each shape with different noises, please refer to the supplementary document. A qualitative result of one such shape (Bunny) is provided in Figure 4.

...distance between surface points and the shape medial axis. However, because of our robust patch fitting and the handling of missing data during matrix factorization, we can generously extend this limit. We have empirically found the chosen radius of $r = 0.03$ to perform well and thus we used it in all the experiments with the meshes from [36].

**Matrix factorization:** For all point-clouds we normalize the parameters of the matrix factorization such that they are invariant to the problem size. With that, we set the parameters to $\lambda = 6.4 \cdot \frac{m^2}{s^2}, \lambda_2 = 10^{-5} \cdot \frac{1}{m^2}, \lambda_E = 2 \cdot 10^{-5} \cdot \frac{1}{m^2}$ and $\lambda_1 = 45 \cdot \frac{1}{s}$. The number of dictionary atoms $r$ is set to 50 for all the point-clouds. Figure 4 shows the qualitative results of our method under Gaussian noise. Experiments are described later with more details.

### 4.2. Evaluation of Design Choices

**Robust patch computation:** Fig. 5 illustrates, qualitatively, how our patch computation method outperforms both (i) simple PCA based approach (left), and (ii) RANSAC plane fitting approach (middle). Obtaining local reference frames for patches using simple PCA results in considering the entire cloud neighborhood which does not align the patch well to the underlying surface in noisy and sharp neighborhoods, whereas our approach aligns it well. Also, robust plane fitting can accommodate outliers far away from the patch region which is not desired. Our method is resistant to both of the unwanted characteristics.

When the amount of noise is small and the patch size is also sufficiently small, the commonly used PCA-based patch fitting is reasonable. In this case, with a sufficiently large amount of random samples, our method becomes close to such a PCA-based approach, which we also demonstrate experimentally: With an added noise with $\sigma = 0.0025$, for the bunny model the error with the PCA-based patch computation results in an RMS error of 6.61e-04, in comparison to 7.17e-04 with our method. However, for more severe noise or larger patches, the assumption that the points within a neighborhood are (approximately) normally distributed breaks, and hence the PCA-based approach is not suitable anymore. Experimentally, we found that with a larger noise of $\sigma = 0.005$, the error increases to 1.02e-03 with the PCA-based approach, in comparison to 1.15e-03 with our method. Hence, our proposed patch extraction method is particularly well-suited for computing patches in highly noisy areas and complicated shapes.

**Evaluation of matrix factorization method:** We have also evaluated the matrix factorization method without our generalization to handle missing data, cf. (3). In this case, for learning the dictionary we replace each missing value in the input matrix $Y$ by the average value taken over all patches. On the bunny model, this results in an RMS of 8.42e-04 for $\sigma = 0.0025$ (compared to 7.17e-04 for our implementation) and for $\sigma = 0.005$ in an error of 1.15e-03 (similar as our implementation 1.15e-03).

Moreover, in Table 1 (and the corresponding table in supplementary material) the column ‘plane’ corresponds to...
For the model FanDisk the RMS error after denoising is $8.05e^{-04}$ (in comparison to $8.12e^{-04}$ in the experiment without white noise). Hence, our denoising algorithm is resistant to outliers.

4.3. Comparison to Other Methods

To get a better understanding of the denoising performance of our method, we perform extensive experiments with the simple i.i.d. Gaussian noise in different point-clouds, and compare our results with the denoising algorithm of (i) Cheng et al. [10], (ii) moving least-squares [4], and (iii) the plane-based approach described in the previous paragraph (i.e. our method without matrix factorization). For [4] we use the implementation in Point Cloud Library (PCL) [35]. The recent work of [10] uses octree decomposition followed by a modified ‘meshless’ Laplacian smoothing and claim to be better, in terms of details retained, as compared to Voronoi based algorithms such as Cocone and RobustCocone [13, 15]. We could not compare our results with another non-local based method [34] because of the lack of the availability of the source code.

4.4. Evaluation of Different Noise Models

Many of the existing point-cloud denoising techniques are closely tied to a reconstruction algorithm itself [42, 41], where the main objective is to achieve a better reconstruction. In contrast, in this work we provide a generic method for processing and denoising point-clouds. Hence, in order to get a better understanding of our algorithm with respect to these regards, in the previous sections we have extensively evaluated it based on i.i.d. Gaussian noise and compared it with other methods that solve the same problem. To complement these evaluations, in this section we provide additional experiments with different types of noise in order to see how well our method generalizes to varying scenarios. Here, our aim is to analyze the general performance of our method under different types of noise commonly observed in real-world applications (e.g. data of different scanner types).

**Noise perpendicular to the surface:** In this experiment...
we add i.i.d. Gaussian noise in the direction of the surface normal, and then use our method to denoise the noisy point cloud. Since our method denoises values that are displacement maps along the normal of the computed patch center, this can be considered as the simplest noise model for our method. The quantitative result for this noise model with $\sigma = 0.005$ is provided in Table 2 (left).

**Noise towards camera:** Analyzing the noise that arises in Kinect scans is an important topic of research in the vision community because of the immense popularity of Kinect for capturing depth maps [31, 23]. It has been found that in Kinect data the axial noise (noise in the depth measurement towards the camera direction) varies significantly and to a large extent with the depth, in comparison to the lateral noise (noise perpendicular to the camera direction) [31]. Motivated by this we investigate a noise model where we add i.i.d. Gaussian noise in the direction of a virtual camera. To make the setup more realistic we scale the shape Bunny and Fandisk so that they have a bounding box of size 50cm, and place the virtual camera at 1.5 meters from the center. Following [31], we added Gaussian noise of standard deviation $\sigma = 0.003$ meters in the direction of the camera to all the points. We apply our denoising algorithm along with the two other methods and show the quantitative and qualitative results in Table 2 (right) and Figure 7 respectively.

**Kinect simulator:** Directional noise provides a good approximation of the axial noise from Kinect depth maps, but it does not provide several other typical artifacts of Kinect, e.g. occlusion boundaries, noise due to distance between IR projector and IR camera, projection patterns etc. For this we use the Kinect simulator provided by [7] and produce a point-cloud for the bunny mesh. We then use our algorithm for denoising and show the result in Figure 8. Even though this noise is far from Gaussian, our method handles it well.

### 4.5. Real-World Applications

**Denoising point-clouds from a range scanner:** We use our method to denoise the data obtained from a structured light scanner (provided by [34]), for which we show qualitative results in Figure 9. It can be seen that the result of our method visually looks better compared to the other methods and the input raw point-cloud.

**Denoising for reconstruction:** Denoising methods are often targeted specifically to improve the quality of reconstruction from a noisy point-cloud [10]. Hence, in this experiment we reconstruct a point-cloud after denoising with our method and show the qualitative result in Figure 10. As a proof-of-concept, these results show that our method can also be used as a preprocessing step for increasing the quality of reconstruction.

### 5. Conclusion

In this work we have presented a method for denoising point-clouds based on dictionary learning. To this end, we robustly extract local planar patches (that represent fixed-length displacement maps) from a given point-cloud. In order to denoise the point-cloud, we denoise the displacement maps based on dictionary learning, and then reconstruct the point-cloud from the denoised patches. Our dictionary learning generalizes the recent structured low-rank matrix factorization [20] so that it can also handle displacement maps with missing data, which is crucial for the proposed denoising method. A particular strength of our method is that it learns a new point-cloud-specific dictionary for each denoising task. It therefore works under a wide range of different types of noise, which we have also confirmed experimentally. Overall, we have shown that our point-cloud denoising framework handles various common types of noise better than previous methods. In the future, we also plan to explore further point-cloud processing tasks within this framework, such as point-cloud inpainting.
References


