Supplementary Material of PD-Flow: A Point Cloud Denoising Framework with Normalizing Flows

A Estimating ELBO of Augmented Flows

In this section, we present a brief description of the variational augmentation used in Section 3.3. Please refer to VFlow [2] for the detailed theoretical proof.

The augmented data distribution $q(\bar{\mathcal{P}} \mid \tilde{\mathcal{P}}; \phi)$ is modeled with a conditional flow

$$\bar{\mathcal{P}} = f_{\phi}^{-1}(\epsilon; \tilde{\mathcal{P}}), \tag{18}$$

where $\epsilon \sim p_{\vartheta}(\epsilon)$ is a known prior distribution defined by user and is similar to $p_{\vartheta}(\tilde{z})$, f_{ϕ}^{-1} denotes the inverse propagation pass of flow f_{ϕ} (similar to Eq. 3 in the main paper), and ϕ denotes the network parameters of augmentation module $\mathcal{A}$. Here, $\tilde{\mathcal{P}}$ is used as the conditional input for f_{ϕ}^{-1} that helps generate the augmented dimensions $\bar{\mathcal{P}}$. The probability density of $\bar{\mathcal{P}}$ can be computed as

$$\log q(\bar{\mathcal{P}} \mid \tilde{\mathcal{P}}; \phi) = \log p_{\vartheta}(\epsilon) - \log \left| \frac{\partial \bar{\mathcal{P}}}{\partial \epsilon} \right|, \tag{19}$$

where $\log q(\bar{\mathcal{P}} \mid \tilde{\mathcal{P}}; \phi)$ is the second term in Eq. 7 and Eq. 16 in the main paper. Please refer to Section B.1 for the detailed implementation of f_{ϕ}^{-1}.

Learning the joint distribution of $\tilde{\mathcal{P}}$ and $\bar{\mathcal{P}}$ can be regarded as modeling $p(\tilde{\mathcal{H}}; \theta)$, where $\tilde{\mathcal{H}} = \{h_i = [\tilde{p}_i, \bar{p}_i]\}$. Thus, we can formulate the probability density of $\tilde{\mathcal{H}}$ by

$$\log p(\tilde{\mathcal{P}}, \bar{\mathcal{P}}; \theta) = \log p(\tilde{\mathcal{H}}; \theta) = \log p_{\vartheta}\left(f_{\theta}(\tilde{\mathcal{H}})\right) + \log \left| \det \frac{\partial f_{\theta}}{\partial \tilde{\mathcal{H}}}(\tilde{\mathcal{H}}) \right|, \tag{20}$$

where θ denotes the network parameters of flow module $\mathcal{F}$. $\log p(\tilde{\mathcal{P}}, \bar{\mathcal{P}}; \theta)$ is the first term in Eq. 7 and Eq. 16, and is similar to Eq. 5 in the main paper.

B Network Configurations

In this section, we present more details of network modules in PD-Flow.

B.1 Augmentation Module

The architecture of augmentation module $\mathcal{A}$ is composed of three components, as shown in Fig. 9.

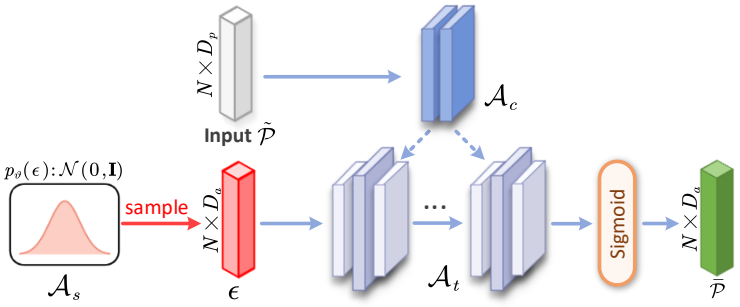


Fig. 9. Architecture of augmentation module $\mathcal{A}$.

The sample module $\mathcal{A}_s$ samples random variables $\epsilon \in \mathbb{R}^{N \times D_a}$ from prior distribution $p_\vartheta(\epsilon)$ as initial augmented dimensions. The condition net $\mathcal{A}_c$ is almost identical to EdgeUnit (described below), but replaces MaxPool with AvgPool. Intuitively, $\mathcal{A}_c$ extracts neighbour context and outputs point-wise features that helps generate extra dimensions. The transformation module $\mathcal{A}_t$ consists of three affine coupling layers and two inverse permutation layers, conditioning on features from $\mathcal{A}_c$. Finally, a sigmoid function is applied to augmented output dimensions $\bar{\mathcal{P}}$ to avoid gradient explosion.

B.2 Flow Components

We briefly review the flow components used in Section 3.4. The forward/inverse propagation formulas and the corresponding log-determinant $\log \left| \det \frac{\partial f_\theta^l}{\partial h^l} \right|$ are listed in Table 5. Note that all operations are performed on channel dimension.

Affine coupling layer. As the core component of flow module $\mathcal{F}$, the affine coupling layer, introduced in RealNVP [5], is a simple yet flexible paradigm to implement invertible transformation.

This layer first partitions the input h^l into two parts $h^l_{1:d}$ and $h^l_{d:D}$: $h^l_{1:d}$ maintains identity and $h^l_{d:D}$ is transformed based on $h^l_{1:d}$. Then, we concatenate them and obtain h^{l+1} as transformed output.

We show the detailed formulation in Table 5, where d is the partition location of channel dimension. We set $d = (D_p + D_a)/2$ in our experimental settings. Transformation units f^l_s and f^l_b represent arbitrarily complex neural networks from $\mathbb{R}^d \mapsto \mathbb{R}^{D_p+D_a-d}$ and are not required to be invertible, as described in Section B.3.

Actnorm. The actnorm layer [7] applies an affine transformation to h_i, with trainable parameters μ and σ. Similar to batch normalization, actnorm helps improve the training stability and performance.

In Table 5, the channel-wise scale term $\mu \in \mathbb{R}^{D_p+D_a}$ and bias term $\sigma \in \mathbb{R}^{D_p+D_a}$ are initialized by the first mini-batch of data to make each channel of h_i obtain zero mean and unit variance.

Table 5. Summarization of flow components.

Flow Component	Forward Propagation	Inverse Propagation	Log Determinant
Affine Coupling Layer	$h^l_{1:d}, h^l_{d+1:D} = \text{split}\left(h^l\right)$ $h^l_s = f^l_{\theta,s}\left(h^l_{1:d}\right)$ $h^l_b = f^l_{\theta,b}\left(h^l_{1:d}\right)$ $h^{l+1}_{1:d} = h^l_{1:d}$ $h^{l+1}_{d+1:D} = h^l_{d+1:D} \odot \exp\left(h^l_s\right) + h^l_b$ $h^{l+1} = \text{concat}\left(h^{l+1}_{1:d}, h^{l+1}_{d+1:D}\right)$	$h^{l+1}_{1:d}, h^{l+1}_{d+1:D} = \text{split}\left(h^{l+1}\right)$ $h^l_s = f^l_{\theta,s}\left(h^{l+1}_{1:d}\right)$ $h^l_b = f^l_{\theta,b}\left(h^{l+1}_{1:d}\right)$ $h^l_{1:d} = h^{l+1}_{1:d}$ $h^l_{d+1:D} = \left(h^{l+1}_{d+1:D} - h^l_b\right) / \exp\left(h^l_s\right)$ $h^l = \text{concat}\left(h^l_{1:d}, h^l_{d+1:D}\right)$	$\sum_{id} h^l_s$
Actnorm	$h^{l+1} = \left(h^l - \mu\right) / \exp\left(\sigma\right)$	$h^l = h^{l+1} \odot \exp\left(\sigma\right) + \mu$	$N \cdot \sum_d \sigma_d$
Invertible 1×1 Convolution	$h^{l+1} = Wh^l$	$h^l = W^{-1}h^{l+1}$	$N \cdot \log\lvert\det(\mathbf{W})\rvert$

* In Log Determinant column, i and d denote the indices of point and channel, respectively.
* Both $\text{split}(\cdot)$ and $\text{concat}(\cdot)$ functions operate on the channel dimension.
* $\odot$ is the Hadamard product.

Permutation layer. Each affine coupling layer only transforms partial channels of h_i. Thus, it has limited non-linear transform capability. To ensure that all dimensions are sufficiently processed, channel permutation techniques help integrate various dimensions and improve transform diversity.

For instance, reverse and random permutations [4, 5] both shuffle the order of channel dimension. The invertible 1×1 convolution [7] (inv1x1) is a special convolutional layer that supports invertibility. Since the dimension of W is low and $\log\lvert\det(W)\rvert$ is relatively easy to compute, we do not apply LU decomposition [7] to W but simply initialize W randomly. In this paper, we interchangeably use inv1x1 and inverse permutation layer.

B.3 Transformation Unit

We implement the transformation units f^l_s and f^l_b in the affine coupling layer in two types: **EdgeUnit** and **LinearUnit**.

Concretely, let $h^l \in \mathbb{R}^{N \times (D_p + D_a)}$ be the input of l-th affine coupling layer. Then, we split the first d channel of h^l i.e. $h^l_{1:d} \in \mathbb{R}^{N \times (D_p + D_a)/2}$ as input of transformation unit f^l_s / f^l_b.

Table 6. Architecture details of EdgeUnit.

	Layer	Kernel Size	Output Size
K1	Build kNN Graph	-	$N \times K \times 3\left(D_p + D_a\right)/2$
F1	Full-connected + ReLU	$3\left(D_p + D_a\right)/2 \times D_h$	$N \times K \times D_h$
F2	Full-connected + ReLU	$D_h \times D_h$	$N \times K \times D_h$
M1	MaxPooling	-	$N \times D_h$
F3	Full-connected	$D_h \times \left(D_p + D_a\right)/2$	$N \times \left(D_p + D_a\right)/2$

The EdgeUnit applies *EdgeConv* [16] to input features, which extracts high-level features from point-wise kNN graph. The kNN graph extraction process is

formulated as

$$h_i^\star = [h_i, \mathcal{N}(h_i), h_i - \mathcal{N}(h_i)], \tag{21}$$

where $h_i = h_{1:d}^l$ are input features, $\mathcal{N}(\cdot)$ denotes the kNN operator, and $h_i^\star \in \mathbb{R}^{K \times 3(D_p + D_a)/2}$ indicates the point-wise features extracted from kNN graph. $h_i^\star$ is fed into EdegUnit and then passed through the remaining layers, as listed in Table 6. The LinearUnit is simply implemented by a bunch of convolutions, as listed in Table 7.

Table 7. Architecture details of LinearUnit.

	Layer	Kernel Size	Output Size
C1	Conv1D + ReLU	$(D_p + D_a)/2 \times D_h$	$N \times D_h$
C2	Conv1D + ReLU	$D_h \times D_h$	$N \times D_h$
C3	Conv1D + ReLU	$D_h \times D_h$	$N \times D_h$
C4	Conv1D	$D_h \times (D_p + D_a)/2$	$N \times (D_p + D_a)/2$

The flow module $\mathcal{F}$ transforms input $\mathcal{P}$ with $L = 4$ to $L = 12$ flow blocks. As demonstrated in RealNVP [5], low levels of blocks encode high frequencies of data (i.e. concrete details), whereas the high levels encode the low frequencies (i.e. abstract details or basic shape). Based on the above idea, we use more EdgeUnit in low level blocks. The EdgeUnit is used to extract neighbor context, whereas LinearUnit is used to extract high-level point-wise features. For both EdgeUnit and LinearUnit, the hidden channels are set to $D_h = 64$, which is the main overhead of network sizes.

C Implementation details

C.1 Dataset configuration

Training phase. The point clouds for training are extracted from 40 mesh models in [17] and then split into patches of 1024 points. These point clouds are randomly perturbed by Gaussian noise with a standard deviation of 0.5% to 2% of the bounding sphere's radius. A pair of noisy patch and the corresponding noise-free patch are cropped on the fly during training. We adopt common data augmentation techniques for training patches, including point perturbation, scaling, and random rotation to increase data diversity and avoid over-fitting.

Evaluation phase. For PUSet/DMRSet, point clouds for testing are points extracted from 20/60 mesh models, at a resolution range from 10K/20K to 50K points. Noisy test points are synthesized by adding Gaussian noise with standard deviation from 1% to 3% of the bounding sphere's radius. We normalize the noisy input points into a unit sphere before denoising and then split them into a cluster by a patch size of 1K for independent denoising. During patch extraction, we first select seed points from input point set as patch centers by the farthest point sampling (FPS) algorithm, and grow the patch to target size by kNN, as done

in [10]. Thereafter, noisy patches are fed into PD-Flow for filtering. Finally, we merge the output patches and sample output points to the target resolution by the FPS algorithm as our final estimation [13]. For the patch-based method, all evaluation metrics are estimated on the whole point set after merging from denoised patches.

C.2 Network Training

In this section, we present the training details of our method, including training strategy, network size and hyper-parameters.

For FBM filter, we set the last D_m channel of $\bar{m}$ to 0. For LBM filter, we randomly initialize $\tilde{m}$ to $[0, 1]$ for all channels. For LCC filter, we initialize W as an identity matrix.

We train our model with Adam optimizer for 700K iterations. The learning rate is initialized as 2×10^{-3} and updated by ReduceLROnPlateau scheduler with $\mathcal{L}_{\text{EMD}}$ as the monitored metric. We set both prior distribution $p_\vartheta(\tilde{z})$ and $p_\vartheta(\epsilon)$ as standard Gaussian distribution $\mathcal{N}(0, \mathbf{I})$. We use the checkpoint of last training epoch as our final model.

Augmenting noisy input $\tilde{\mathcal{P}}$ with $D_a = 32$ in $\mathcal{A}$ and using a flow module $\mathcal{F}$ with $L = 8$ blocks can achieve good performance for most cases. The tunning hyper-parameters α, β and γ in Eq. 17 in the main paper are empirically set as $1e^{-6}$, 0.1 and 10, respectively.

C.3 Partition denoising

In experiments, we observe an obvious performance degradation of our model in denoising high resolution point clouds along with high noise levels. This is primarily because patch-based denoise methods [12, 9, 10] generally use a fixed patch size as the denoising unit (e.g., 1024 points per patch). Denoising a higher resolution point cloud indicates that the network handles a patch with a smaller surface region. The surface estimation becomes unstable as the respective field that the network perceives become small, particularly in the denoising problem.

To resolve this problem, we first partition the high-resolution point cloud (e.g., 50K points) into several parts (e.g., 10K points), with each part sharing approximately the same shape as the original point cloud. Then, we send each part to the network for patch-based denoising (Section C.1). Finally, we concatenate each part back to the original resolution. In this way, we avoid clustering much noise in a single patch and maintain good denoising performance across different point resolutions. We only use this setting for point clouds with both high resolution and high noise levels.

Another solution to this problem may use ball-query algorithm instead of KNN to generate point patch. This method introduces another hyperparameter (i.e. radius for query), which is required to be fine-tuned for each point resolution. From experiments, we observe that this solution does help to preserve good results under high noise level and high point resolution. In most cases, it achieves competitive performance as the first solution.

D Evaluation metrics

In this section, we introduce the metrics used in quantitative comparison. For all the metrics, the lower the values are, the better the denoising quality is.

Chamfer distance (CD) between the predicted point cloud $\hat{\mathcal{P}}$ and ground truth point cloud $\mathcal{P}$ is defined as

$$\mathcal{L}_{\mathrm{CD}}(\hat{\mathcal{P}}, \mathcal{P}) = \frac{1}{|\hat{\mathcal{P}}|} \sum_{\hat{p} \in \hat{\mathcal{P}}} \min_{p \in \mathcal{P}} \|\hat{p} - p\| + \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \min_{\hat{p} \in \hat{\mathcal{P}}} \|p - \hat{p}\|, \tag{22}$$

where $|\hat{\mathcal{P}}|$ and $|\mathcal{P}|$ denote the number of points in $\hat{\mathcal{P}}$ and $\mathcal{P}$, $\|\cdot\|$ denotes L_2 norm. This first term in Eq. 22 measures the average accuracy to the ground truth surface of each predicted point, whereas the second term in Eq. 22 encourages an even coverage to ground truth distribution.

Point-to-mesh (P2M) distance is defined as

$$\mathcal{L}_{\mathrm{P2M}}(\hat{\mathcal{P}}, \mathcal{M}) = \frac{1}{|\hat{\mathcal{P}}|} \sum_{\hat{p} \in \hat{\mathcal{P}}} \min_{f \in \mathcal{M}} d(\hat{p}, f) + \frac{1}{|\mathcal{M}|} \sum_{f \in \mathcal{M}} \min_{\hat{p} \in \hat{\mathcal{P}}} d(\hat{p}, f), \tag{23}$$

where $\mathcal{M}$ is the corresponding mesh of $\hat{\mathcal{P}}$, $|\hat{\mathcal{P}}|$ is the number of points in $\hat{\mathcal{P}}$, and f is the triangular face in $\mathcal{M}$ (with a total of $|\mathcal{M}|$ faces). The $d(p, f)$ function measures the squared distance from point p to face f. The P2M metric estimates the average accuracy that approximates the underlying surface.

Point-to-surface (P2S) distance is defined as

$$\mathcal{L}_{\mathrm{P2S}}(\hat{\mathcal{P}}, \mathcal{P}) = \frac{1}{|\hat{\mathcal{P}}|} \sum_{\hat{p} \in \hat{\mathcal{P}}} \min_{p \in \mathcal{P}} \min_{q \in \mathcal{S}_p} \|\hat{p} - q\|, \tag{24}$$

where $|\hat{\mathcal{P}}|$ denotes the number of points in $\hat{\mathcal{P}}$ and $\|\cdot\|$ denotes L_2 norm. Note that $\mathcal{S}_p$ is the surface (i.e. flat plane) defined by the coordinate and normal of p, whereas q is the closest points to $\hat{p}$ on $\mathcal{S}_p$. The P2S metric is similar to P2M metric, but requires normal data of testing point clouds instead of mesh data.

Hausdorff distance (HD) is defined as

$$\mathcal{L}_{\mathrm{HD}}(\hat{\mathcal{P}}, \mathcal{P}) = \max(\max_{\hat{p} \in \hat{\mathcal{P}}} \min_{p \in \mathcal{P}} \|\hat{p} - p\|, \max_{p \in \mathcal{P}} \min_{\hat{p} \in \hat{\mathcal{P}}} \|p - \hat{p}\|), \tag{25}$$

where $\|\cdot\|$ denotes L_2 norm. $\mathcal{L}_{\mathrm{HD}}$ measures the maximum distance of point set to the nearest point in another point set, which is sensitive to outliers.

Uniform metric (Uni), which is proposed in PU-GAN [8], is used to evaluate point distribution uniformity.

This metric first uses FPS to pick seed points and then estimates the uniformity on point subset within a ball query centered at each seed point. Depending on the ball query radius r_d, it can evaluate the uniformity of different area sizes.

In general, we prefer the generated point clouds to follow a uniform distribution. In our experiment settings, we evaluate the Uni metric with $r_d = \sqrt{p}$ where $p \in \{0.4\%, 0.6\%, 0.8\%, 1.0\%, 1.2\%\}$. Please refer to [8] for the detailed formulation.

E Additional Quantitative Results

E.1 Evaluation on DMRSet

Table 8 shows the quantitative results evaluated on DMRSet. For a fair comparison, we retrain the deep-learning-based models including PCNet [14], ScoreDenoise [10] and our method on the training set of DMRSet, and use the pretrain model of DMRDenoise [9]. Since the resolution of 20K points is not released by authors [9], we generate 20K points and corresponding normals from the released 50K points.

Table 8. Comparison of denoising algorithms on DMRSet.

#Points	20K								50K							
Noise	1%		2%		2.5%		3%		1%		2%		2.5%		3%	
Method	CD 10^{-4}	P2S 10^{-4}	CD 10^{-4}	P2S 10^{-4}	CD 10^{-4}	P2S 10^{-4}	CD 10^{-4}	P2S 10^{-4}	CD 10^{-4}	P2S 10^{-4}	CD 10^{-4}	P2S 10^{-4}	CD 10^{-4}	P2S 10^{-4}	CD 10^{-4}	P2S 10^{-4}
Jet[1]	2.21	0.56	2.50	0.73	2.66	0.84	2.85	0.98	2.31	1.01	4.05	2.39	4.89	3.08	5.74	3.78
MRPCA[11]	**2.07**	**0.44**	2.26	**0.51**	2.43	**0.61**	2.67	**0.76**	2.07	0.75	4.06	2.18	5.04	2.97	5.95	3.71
GLR[18]	2.14	0.53	**2.18**	0.59	2.39	0.73	2.61	0.89	**1.78**	**0.65**	2.83	1.47	3.55	2.04	4.41	2.72
PCNet[14]	2.25	0.64	2.69	0.88	3.02	1.11	3.36	1.35	2.15	0.71	3.43	1.54	4.18	2.06	4.92	2.61
DMR[9]	2.13	**0.51**	2.30	**0.64**	2.42	**0.72**	**2.54**	**0.82**	**1.77**	**0.65**	2.70	1.40	3.41	1.97	4.27	2.68
Score[10]	2.21	0.59	2.48	0.76	2.61	0.86	2.74	0.96	1.91	0.73	2.81	1.40	3.43	1.84	4.19	2.38
Ours	**2.02**	**0.50**	**2.17**	**0.61**	**2.30**	**0.71**	**2.49**	0.86	**1.71**	**0.65**	**2.49**	**1.15**	**2.77**	**1.45**	**3.30**	**1.92**

From Table 8, we can observe that our method yields better performance than other methods in most noise settings. The performance improvement becomes obvious especially in denoising 50K points.

E.2 Generalizability on unseen noise pattern

In this section, we investigate the denoising performance of our method on a variety of unseen noise types. We use the same noise settings as that of ScoreDenoise [10]. Please refer to supplementary material of [10] for detailed noise configurations.

For a fair comparison, we evaluate on the same models used in Section 4.2, which are trained with Gaussian noise. We compare our method against Jet [1], MRPCA [11], DMRDenoise [9] and ScoreDenoise [10], as these methods represent the state-of-the-art performance.

Table 9 shows the quantitative denoising results under simulated LiDAR noise. The LiDAR noise reproduces the common noise pattern generated by scanners. We can observe that both ScoreDenoise [10] and our method obtain the most accurate estimation. Our method outperforms ScoreDenoise [10] in different metrics, probably thanks to the uniform distributed pattern.

Table 11 shows the quantitative denoising results under non-isotropic Gaussian noise, uni-directional noise, uniform noise, discrete noise, respectively. Traditional methods generally rely on parameters turning to achieve good denoising performance (such as noise levels, repeating iterations and neighbor sizes, etc).

Table 9. Comparison of denoising algorithms under simulated LiDAR noise.

	Jet	MRPCA	GLR	DMR	Score	Ours
CD	3.84	3.76	3.28	4.28	3.17	**2.82**
P2M	1.37	1.49	1.13	1.67	0.92	**0.76**
HD	3.74	3.69	3.75	4.32	3.56	**3.20**

Therefore, these methods are difficult to preserve consistent performance under various noise settings and point resolutions. Our method obtains the best results in the majority of cases. More importantly, our method can maintain reliable quality and achieve competitive results in all metrics. This indicates the robust generalizability of our method.

E.3 Training on various noise distributions

In this section, we investigate how the noise distribution used for training affects the denoising results of deep-learning based methods. Thus, we retrain these methods on various noise types and evaluate them on PUSet with 10K points. From Table 10, we observe that using the model trained on the same noise type as evaluation can further improve denoising performance.

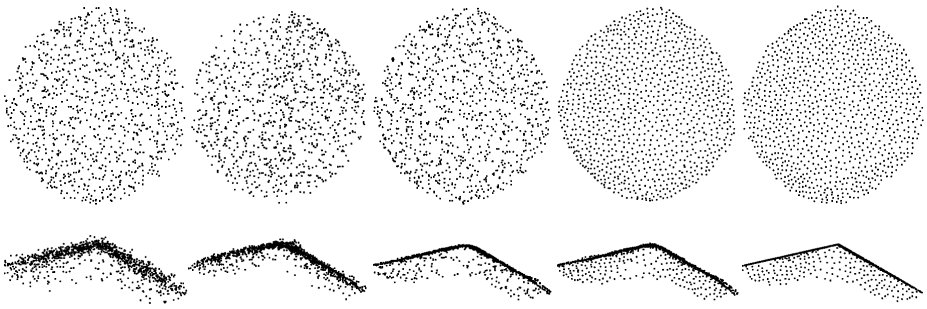
Table 10. Comparison of deep-learning based methods trained on various noise types.

	Noise Type (Training)	isotropic Gaussian			non-isotropic Gaussian			Uniform		
Noise Type (Evaluation)	Method	CD 10^{-4}	P2M 10^{-4}	HD 10^{-3}	CD 10^{-4}	P2M 10^{-4}	HD 10^{-3}	CD 10^{-4}	P2M 10^{-4}	HD 10^{-3}
isotropic Gaussian(2%)	DMR [9]	5.04	2.13	7.02	5.24	2.27	4.58	6.14	2.88	5.31
	Score [10]	3.68	1.08	5.78	3.85	1.34	6.19	4.86	1.91	4.56
	Ours	3.25	1.02	3.71	3.28	1.07	2.70	6.05	3.08	4.62
non-isotropic Gaussian(2%)	DMR [9]	5.26	2.33	7.40	4.00	1.31	3.78	6.24	2.98	5.40
	Score [10]	3.75	1.15	4.35	3.58	1.08	3.26	5.16	2.19	5.65
	Ours	3.36	1.12	3.09	3.19	1.01	2.87	6.25	3.29	6.48
Uniform	DMR [9]	4.52	1.75	6.14	3.96	1.33	6.10	3.75	0.94	3.56
	Score [10]	2.48	0.42	1.27	3.00	0.92	1.89	2.43	0.38	0.92
	Ours	2.05	0.25	0.93	2.51	0.71	1.26	1.94	0.27	0.77

F Additional Qualitative Results

F.1 Visualization on denoised patch

To obtain a more intuitive perception of denoised results for patch-based methods, we visualize the denoised patches of different methods from the same noisy patch (a), as shown in Fig. 10.



(a) Noisy (b) DMR [9] (c) Score [10] (d) Ours (e) Clean

Fig. 10. Visual comparison of a denoised patch from different view angles. The first row shows the top views, and the second row shows the front views.

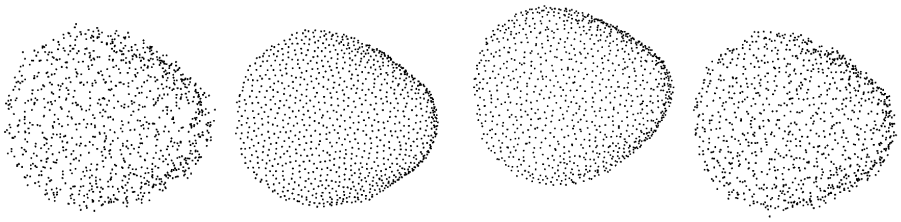
From the first row of Fig. 10, we observe that our method distinguishes from other methods in terms of uniformity, even though we do not explicitly enforce the uniform metric in the training loss. From the second row of Fig. 10, both ScoreDenoise [10] and our method can produce points with less jitters on surface, indicating that both methods can infer smooth surface properties between estimated points.

F.2 Investigation on disentangled latent space

To verify whether the latent point representation is disentangled, we conduct an experiment to investigate how the latent z of NFs affects generative quality. We inspect the effect of clean and noisy channels by adding noise o_d.

To be specific, let z_p and z_n be the channels of noisy latent $\tilde{z}$, which are supposed to embed point and noise respectively. We set $D_a = 33$ and use FBM filter with $\bar{m} = [\underbrace{1, 1, \cdots, 1}_{\text{18 elements}}, \underbrace{0, 0, \cdots, 0}_{\text{18 elements}}]$, such that z_p and z_n both contain 18 channels. Let $z_c \in \mathbb{R}^{18}$ denotes the channels denoised by FBM filter, where $z_c = 0$ in this case. Fig. 11 visualizes the patches decoded from several latent z combinations by adding noise to different components. The directional noise o_d is defined as $o_d = \delta \mathbf{1}$, where $\mathbf{1} \in \mathbb{R}^{18}$ is an all-ones vector and δ is a scalar adjusted by user ($\delta = 1.0$ in this case).

Since Fig. 11a does not change the noisy latent $\tilde{z}$, our method generates the same patch as noisy input due to the invertibility of NF. By replacing z_n with z_c, we obtain a well-distributed patch in Fig. 11b. Since z_c does not embed any information and NF is a lossless propagation process, without loss of generality, we can hypothesize that NF encodes the noisy patch to a regularized space of z_p where points are uniformly distributed and embeds the point-wise distortion to z_n. This gives us an intuitive explanation of why our method achieves better uniformity.



(a) $\tilde{z} = [z_p, z_n]$ (b) $\hat{z} = [z_p, z_c]$ (c) $z_1 = [z_p + o_d, z_c]$ (d) $z_2 = [z_p, z_c + o_d]$

Fig. 11. Point patches generated from different latent vectors $\hat{z}$.

If we add directional noise o_d to z_p, we observe an unified translation to each point in Fig. 11c. For figure plotting convenience, we only show a relative small amount of translation in Fig. 11c. This phenomenon indicates that z_p probably embeds some information that is absolute to point's coordinate. If we add directional noise o_d to z_c, we observe that each point in Fig. 11d is corrupted by random noise, indicating that the z_c channels probably encode some properties that disturb patch uniformity. The different behaviors in Fig. 11c and Fig. 11d verify that z_p and z_c share different functions in latent point representation; in other words, the latent code space of z is disentangled.

F.3 More denoised results

In this section, we illustrate more denoised results in the following figures. Fig. 12 shows some denoising examples on Paris-CARLA-3D [3] dataset. Fig. 13 shows more denoising examples on *Paris-rue-Madame* [15] dataset. Fig. 14 shows some denoising examples on 3DCSR [6] dataset. Fig. 15 shows more denoising examples under simulated LiDAR noise. Fig. 16 shows more denoising examples under 1% to 3% Gaussian levels. Fig. 17 shows more denoising examples under unseen noise types.

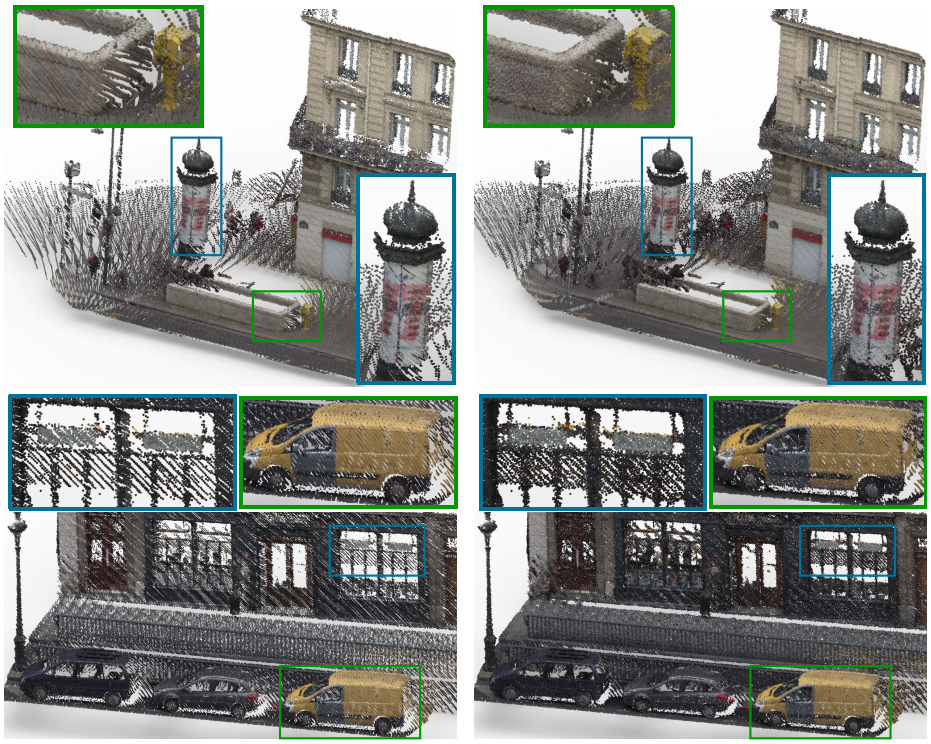
G Limitation and Future work

Most existing methods (including our method) do not contain noise level estimation. Therefore, how many denoising iterations are needed to achieve the best result is uncertain. Generally, for our method, a single iteration is enough for low noise level (e.g., 1% and 2% Gaussian noise), while 2 iterations are needed for high noise level (e.g., 3% Gaussian noise).

In the future, we will enhance the denoising capability of PD-Flow by introducing the outlier filtering operator and make further investigation on the influence of latent dimensions. Furthermore, we can extend the propagation pipeline of PD-Flow to point cloud compression task, which takes high priority in efficient storage and detail reconstruction.

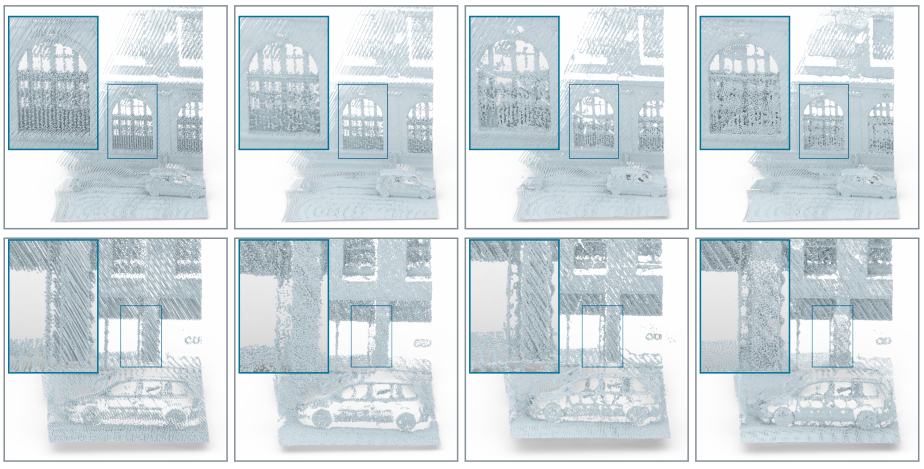
Table 11. Comparison of denoising algorithms under various noise types on PUSet.

#Points		10K									50K								
Noise Level		1%			2%			3%			1%			2%			3%		
Noise Type	Method	CD 10^{-4}	P2M 10^{-4}	HD 10^{-3}	CD 10^{-4}	P2M 10^{-4}	HD 10^{-3}	CD 10^{-4}	P2M 10^{-4}	HD 10^{-3}	CD 10^{-4}	P2M 10^{-4}	HD 10^{-3}	CD 10^{-4}	P2M 10^{-4}	HD 10^{-3}	CD 10^{-4}	P2M 10^{-4}	HD 10^{-3}
non-isotropic Gaussian	Jet [1]	3.30	1.07	1.81	4.71	1.82	3.21	6.25	3.01	9.49	0.93	0.26	1.05	1.86	0.94	4.49	4.02	2.81	10.6
	MRPCA [11]	3.35	1.30	1.71	4.06	1.48	**2.84**	**4.90**	**2.06**	6.85	0.70	**0.13**	**0.77**	1.51	0.70	3.92	4.61	3.22	10.3
	DMR [9]	4.63	1.82	5.46	5.26	2.33	7.40	6.29	3.20	10.1	1.22	0.50	2.76	1.70	0.91	4.26	2.78	1.85	6.79
	Score [10]	2.50	0.47	1.57	3.75	1.15	4.35	**4.90**	2.14	10.0	0.72	0.15	2.85	1.38	**0.65**	5.49	**2.24**	**1.31**	6.24
	Ours	**2.13**	**0.40**	**1.16**	**3.36**	**1.12**	**3.09**	5.01	2.56	**5.42**	**0.66**	0.17	**1.04**	**1.22**	**0.63**	**2.17**	**2.30**	1.60	**5.12**
Uni-directional	Jet [1]	2.16	0.88	1.50	3.29	1.23	2.31	4.20	1.75	5.21	0.64	0.17	0.65	1.01	0.37	2.08	1.69	0.90	6.38
	MRPCA [11]	2.43	1.25	1.64	3.29	1.43	2.16	3.91	1.71	3.96	0.52	0.11	**0.52**	**0.78**	**0.23**	2.17	1.66	0.91	6.30
	DMR [9]	4.47	1.75	6.06	4.75	1.93	5.87	5.25	2.35	7.04	1.08	0.40	2.05	1.26	0.54	2.79	1.64	0.86	4.99
	Score [10]	1.47	0.28	1.36	2.46	0.56	2.13	3.50	1.25	5.77	**0.49**	**0.06**	1.28	**0.81**	0.26	2.47	**1.19**	**0.55**	9.48
	Ours	**1.19**	**0.23**	**0.75**	**2.21**	**0.51**	**1.74**	**3.07**	**1.06**	**3.77**	0.47	**0.08**	**0.58**	**0.81**	0.32	**1.31**	1.30	0.71	**2.70**
Uniform	Jet [1]	1.97	0.83	1.23	3.30	1.02	1.49	4.04	1.33	2.01	0.65	0.13	0.33	0.89	0.23	**0.56**	1.20	0.42	0.91
	MRPCA [11]	2.29	1.24	1.47	3.40	1.28	1.69	3.82	1.35	2.03	0.61	0.12	0.33	**0.52**	**0.11**	0.61	0.86	**0.22**	**0.80**
	DMR [9]	4.42	1.70	6.17	4.52	1.75	6.14	4.75	1.93	6.61	1.09	0.40	1.65	1.21	0.50	2.12	1.33	0.60	2.87
	Score [10]	1.31	0.25	0.72	2.48	0.42	1.27	3.41	1.01	7.30	0.50	**0.04**	1.19	0.70	0.13	1.67	0.91	0.30	2.90
	Ours	**0.92**	**0.18**	**0.61**	**2.05**	**0.35**	**0.93**	**2.70**	**0.68**	**1.44**	**0.45**	**0.05**	**0.29**	**0.63**	0.15	**0.65**	**0.89**	0.36	**1.22**
Discrete	Jet [1]	1.93	0.82	1.32	2.89	0.98	1.52	3.40	1.26	1.90	0.56	0.14	0.34	0.91	0.15	**0.44**	1.08	0.43	0.99
	MRPCA [11]	2.25	1.24	1.49	3.02	1.28	1.67	3.22	1.32	1.86	1.11	0.46	0.92	**0.54**	**0.11**	0.47	**0.61**	**0.15**	**0.80**
	DMR [9]	4.42	1.70	5.85	4.58	1.78	5.25	4.84	1.98	6.61	1.11	0.42	1.83	1.21	0.50	2.36	1.37	0.63	2.77
	Score [10]	1.27	0.25	0.93	2.19	0.42	4.27	3.08	1.01	3.07	**0.45**	**0.05**	1.20	0.62	0.14	1.84	0.78	0.25	2.02
	Ours	**0.91**	**0.18**	**0.60**	**1.90**	**0.35**	**0.91**	**2.58**	**0.69**	**1.31**	**0.43**	**0.06**	**0.34**	**0.59**	0.14	**0.96**	0.83	0.32	1.45



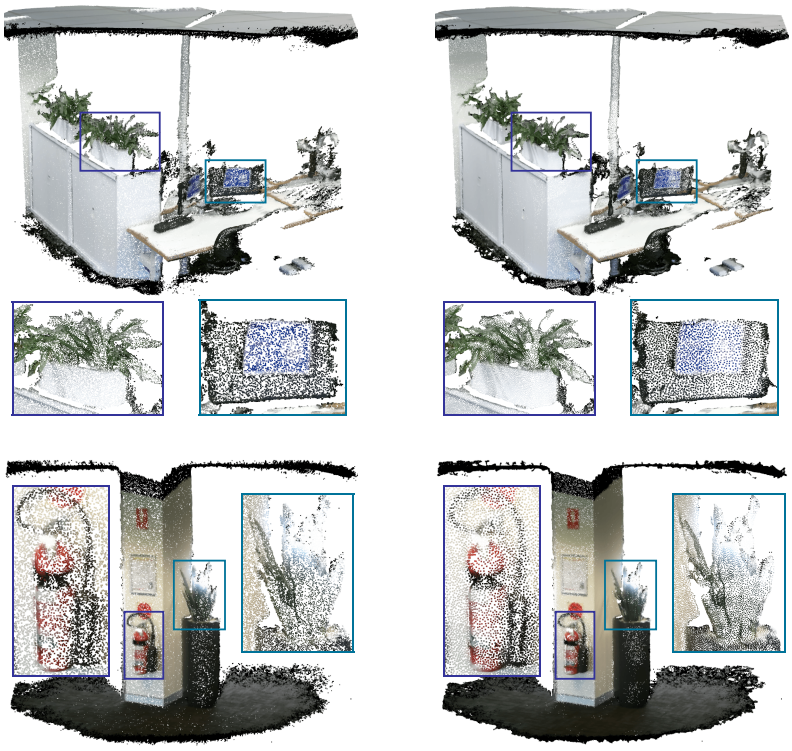
(a) Noisy (b) Ours

Fig. 12. Denoised results on Paris-CARLA-3D [3] dataset.



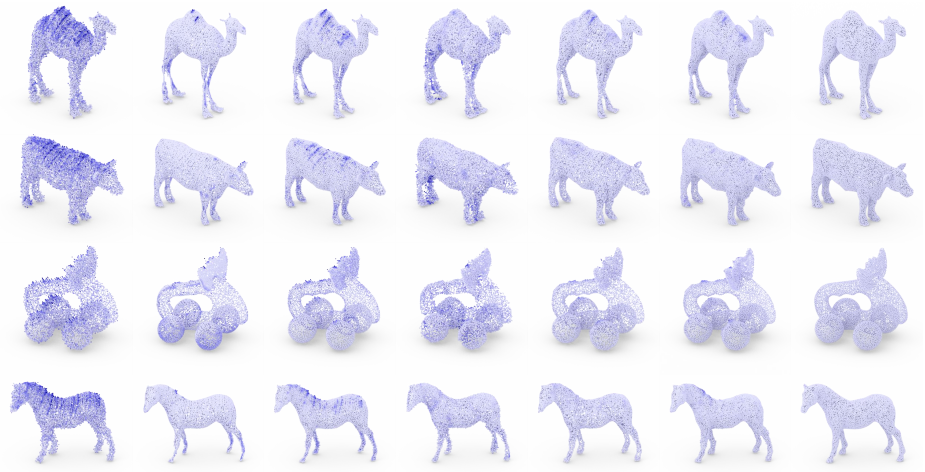
(a) Noisy (b) DMRDenoise [9] (c) ScoreDenoise [10] (d) Ours

Fig. 13. More visual comparison on *Paris-rue-Madame* [15] dataset.



(a) Noisy (b) Ours

Fig. 14. Denoised results on 3DCSR [6] dataset.



(a) Noisy (b)MRPCA[11] (c) PCNet[14] (d) DMR [9] (e) Score [10] (f) Ours (g) Clean

Fig. 15. Visual comparison under simulated LiDAR noise.

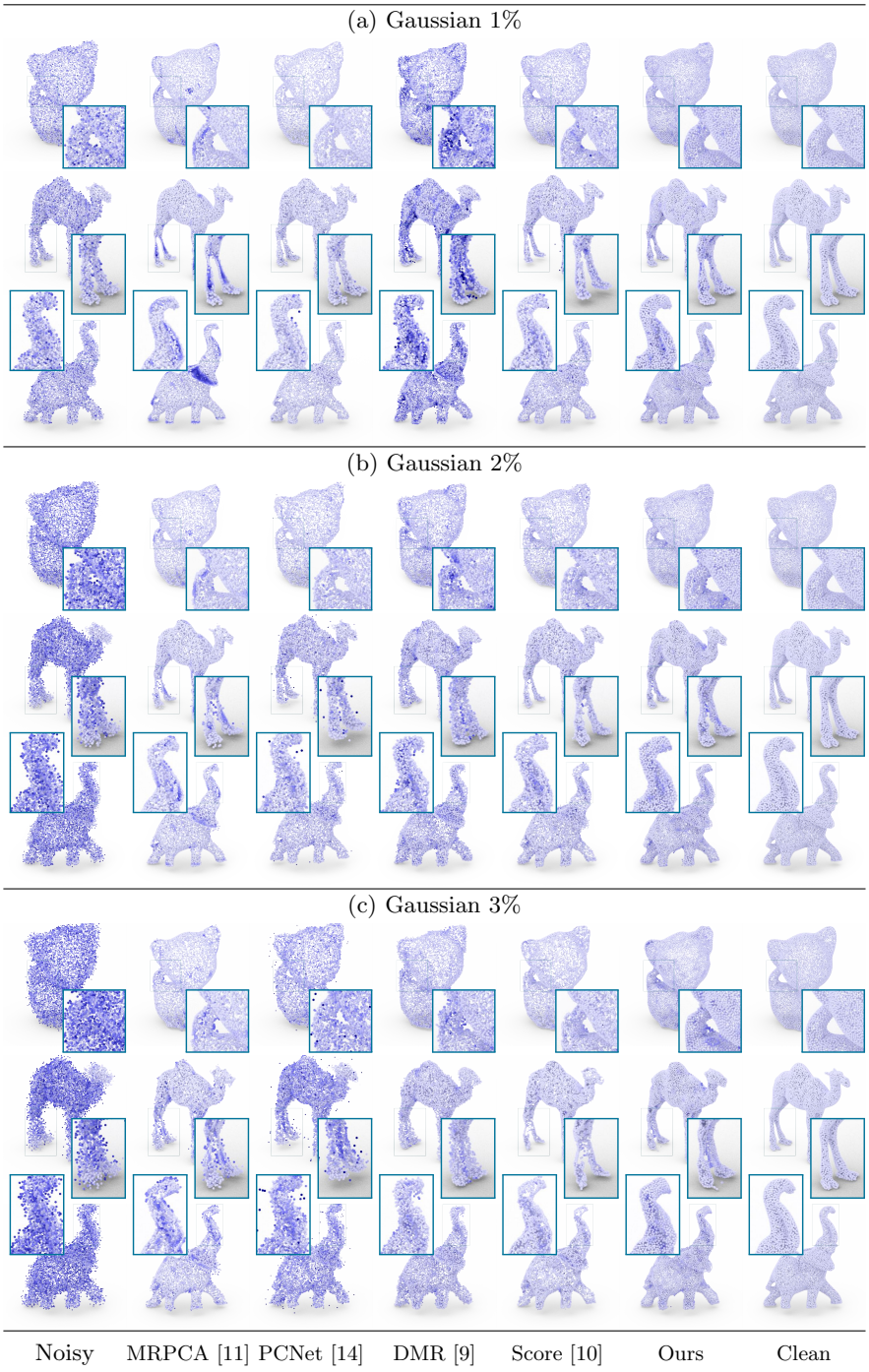


Fig. 16. Visual comparison under various noise levels.

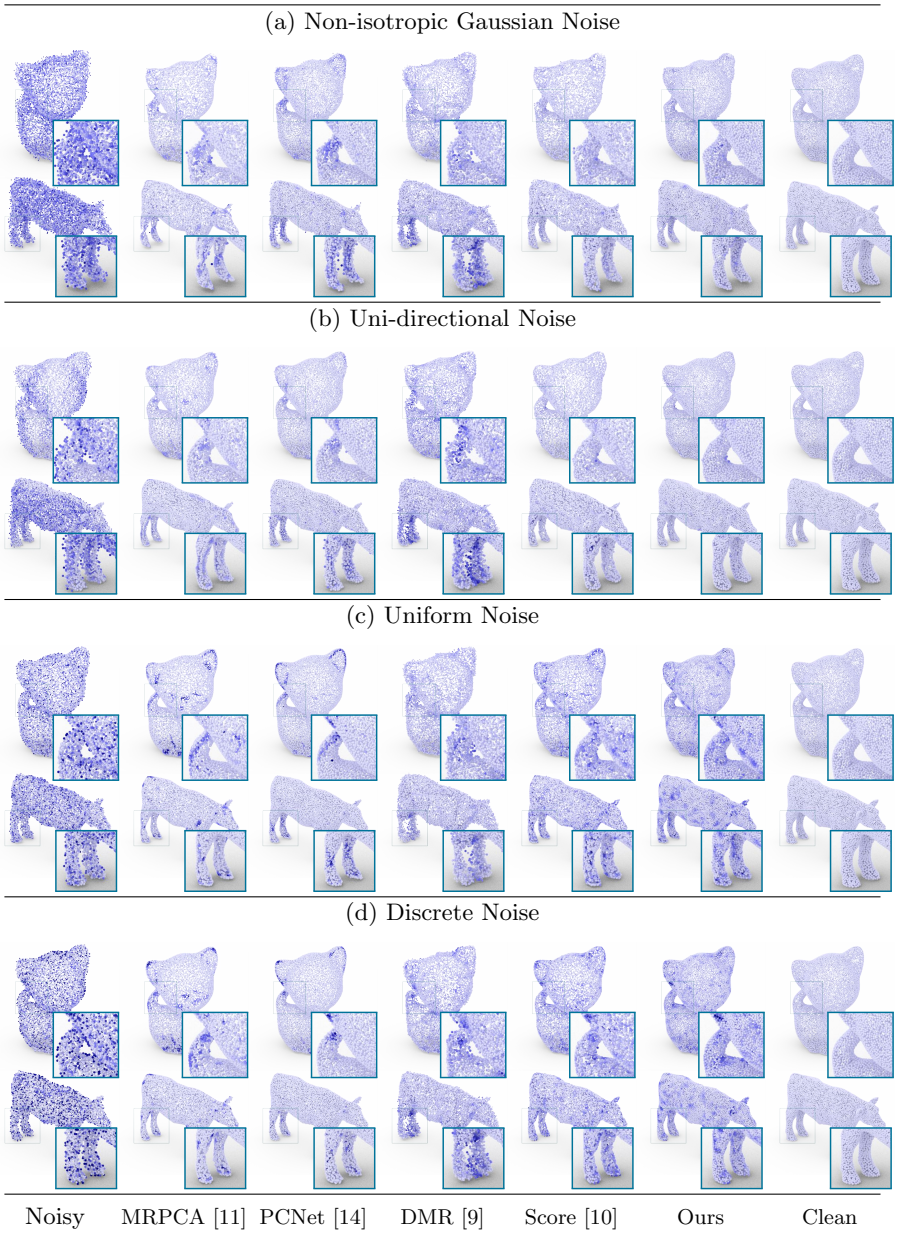


Fig. 17. Visual comparison under various noise types.

References

1. Cazals, F., Pouget, M.: Estimating differential quantities using polynomial fitting of osculating jets. Computer Aided Geometric Design **22**(2), 121–146 (2005)
2. Chen, J., Lu, C., Chenli, B., Zhu, J., Tian, T.: Vflow: More expressive generative flows with variational data augmentation. In: International Conference on Machine Learning. pp. 1660–1669 (2020)
3. Deschaud, J.E., Duque, D., Richa, J.P., Velasco-Forero, S., Marcotegui, B., Goulette, F.: Paris-carla-3d: A real and synthetic outdoor point cloud dataset for challenging tasks in 3d mapping. Remote Sensing **13**(22) (2021). https://doi.org/10.3390/rs13224713, https://www.mdpi.com/2072-4292/13/22/4713
4. Dinh, L., Krueger, D., Bengio, Y.: Nice: Non-linear independent components estimation. arXiv preprint arXiv:1410.8516 (2014)
5. Dinh, L., Sohl-Dickstein, J., Bengio, S.: Density estimation using real NVP. In: 5th International Conference on Learning Representations, ICLR 2017 (2017)
6. Huang, X., Mei, G., Zhang, J., Abbas, R.: A comprehensive survey on point cloud registration. arXiv preprint arXiv:2103.02690 (2021)
7. Kingma, D.P., Dhariwal, P.: Glow: Generative flow with invertible 1x1 convolutions. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018)
8. Li, R., Li, X., Fu, C.W., Cohen-Or, D., Heng, P.A.: Pu-gan: a point cloud upsampling adversarial network. In: IEEE International Conference on Computer Vision (ICCV) (2019)
9. Luo, S., Hu, W.: Differentiable manifold reconstruction for point cloud denoising. In: Proceedings of the 28th ACM International Conference on Multimedia. pp. 1330–1338 (2020)
10. Luo, S., Hu, W.: Score-based point cloud denoising. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 4583–4592 (October 2021)
11. Mattei, E., Castrodad, A.: Point cloud denoising via moving rpca. In: Computer Graphics Forum. vol. 36, pp. 123–137. Wiley Online Library (2017)
12. Pistilli, F., Fracastoro, G., Valsesia, D., Magli, E.: Learning graph-convolutional representationsfor point cloud denoising. In: The European Conference on Computer Vision (ECCV) (2020)
13. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 652–660 (2017)
14. Rakotosaona, M.J., La Barbera, V., Guerrero, P., Mitra, N.J., Ovsjanikov, M.: Pointcleannet: Learning to denoise and remove outliers from dense point clouds. In: Computer Graphics Forum. vol. 39, pp. 185–203. Wiley Online Library (2020)
15. Serna, A., Marcotegui, B., Goulette, F., Deschaud, J.E.: Paris-rue-madame database: a 3d mobile laser scanner dataset for benchmarking urban detection, segmentation and classification methods. In: 4th International Conference on Pattern Recognition, Applications and Methods ICPRAM 2014 (2014)
16. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph cnn for learning on point clouds. ACM transactions on graphics (TOG) **38**(5), 1–12 (2019)

17. Yu, L., Li, X., Fu, C.W., Cohen-Or, D., Heng, P.A.: Pu-net: Point cloud upsampling network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2790–2799 (2018)
18. Zeng, J., Cheung, G., Ng, M., Pang, J., Yang, C.: 3d point cloud denoising using graph laplacian regularization of a low dimensional manifold model. IEEE Transactions on Image Processing **29**, 3474–3489 (2019)