Supplementary Document of StraightPCF: Straight Point Cloud Filtering

Dasith de Silva Edirimuni[1], Xuequan Lu[2*], Gang Li[1], Lei Wei[1], Antonio Robles-Kelly[1], Hongdong Li[3]
[1]Deakin University, [2]La Trobe University, [3]Australian National University
{dtdesilva, gang.li, lei.wei, antonio.robles-kelly}@deakin.edu.au,
b.lu@latrobe.edu.au, hongdong.li@anu.edu.au

This supplementary document contains the following:

A. Additional methodology details.
  A.1. VelocityModule training.
  A.2. Inference time filtering objective for full network.
B. Further evaluation on synthetic and scanned data
  B.1. Performance of additional methods on PUNet and PCNet data with Gaussian noise.
  B.2. Additional visual results on real-world scanned data.
C. Further evaluation on PUNet and PCNet data under different noise patterns.
D. Comparison of test times for different methods.
E. Further ablation studies.
  E.1. Higher VelocityModule couplings, K.
  E.2. Impact of Euler step number, N.
F. Discussion of limitations.

A. Additional Methodology Details

A.1. VelocityModule training

In Sec. 4.1 of the main paper, we presented the VelocityModule that enables filtering via straight flows. Here, we provide more detail into its training objective. We train the VelocityModule by minimizing the loss of Eq. (7) in the main paper. During training, we first draw samples $\boldsymbol{X}_0 = \boldsymbol{Y} + \sigma_H \xi \ \wedge \ \xi \sim \mathcal{N}(0, I)$ and $\boldsymbol{X}_1 = \boldsymbol{Y}$ from π_0 and π_1, respectively. Given a pair $(\boldsymbol{X}_0, \boldsymbol{X}_1)$, we use the linear interpolation of Eq. (4), given in the main paper, to sample an intermediate state at time $t \sim \mathcal{U}(0, 1)$. For the filtering task, we do not have explicit knowledge of the time step at which we start the filtering process. Consequently, we do not input the time step t to the VelocityModule, we only provide $\boldsymbol{X}_t$. We then obtain the optimal parameters $\theta^\star$ of the VelocityModule by minimizing the expected value over time of the L_2 norm term in Eq. (7), given in the main paper, such that,

$$\theta^\star = \underset{\theta}{\operatorname{argmin}} \left\{ \mathbb{E}_{t \sim \mathcal{U}(0,1)} \left[\| \boldsymbol{v}_\theta(\boldsymbol{X}_t) - \boldsymbol{\delta}(\boldsymbol{X}_1, \boldsymbol{X}_0) \|_2^2 \right] \right\}. \tag{1}$$

*Corresponding author: X. Lu, supported by fund 3.2501.11.47.

A.2. Inference time filtering objective for full network

In this section, we provide supplementary details on the filtering objective of the full network, with trained VelocityModule coupling and DistanceModule. Eq. (15) of the main paper provides the sequential position update for our StraightPCF network. Now, the full position update across time steps $\mathcal{S} = \{\hat{M}, \hat{M} + K, \ldots, N(K-1)\}$ is,

$$\tilde{\boldsymbol{X}}_1 = \tilde{\boldsymbol{X}}_{\hat{M}/T} + \frac{d_\phi(\tilde{\boldsymbol{X}}_{\hat{M}/T})}{T} \sum_{\hat{t} \in \mathcal{S}} \sum_{k=0}^{K-1} \boldsymbol{v}_\theta^k(\tilde{\boldsymbol{X}}_{(\hat{t}+k)/T}). \tag{2}$$

We apply our StraightPCF network across $T = N \cdot K$ total iterations. At very high noise levels, this entire filtering process has to be repeated, similar to ScoreDenoise [8]. The reason for this is the upper limit of our training noise scales is only $\sigma = 2\%$. Therefore, at inference, for $\sigma = 2\%$ noise, we repeat the full position update 2 times while for $\sigma = 3\%$, we repeat the process 3 times.

B. Further Evaluation on Synthetic and Scanned Data

In this section, we provide additional results on both synthetic and scanned data that could not be added to the main paper, due to constraints of space. Moreover, we provide more detail into our experimental set-up. To obtain Chamfer Distance (CD) and Point2Mesh (P2M) results, we have used the implementation of PyTorch3D [13] with the same settings as ScoreDenoise [8], PDFlow [9] and DeepPSR [2]. For all filtering results, to ensure fair comparison, learning based methods were only trained on the synthetic PUNet training set with Gaussian noise.

B.1. Performance of additional methods on PUNet and PCNet data with Gaussian noise

Table 1 presents the performance of deep learning-based methods and conventional methods which were not included in the main paper, due to limitations of space. These results are for PUNet and PCNet data with Gaussian noise

	Resolution	10K (Sparse)						50K (Dense)					
	Noise	1%		2%		3%		1%		2%		3%	
	Method	CD	P2M	CD	P2M	CD	P2M	CD	P2M	CD	P2M	CD	P2M
PUNet dataset [14]	Bilateral [4]	3.646	1.342	5.007	2.018	6.998	3.557	0.877	0.234	2.376	1.389	6.304	4.730
	Jet [1]	2.712	0.613	4.155	1.347	6.262	2.921	0.851	0.207	2.432	1.403	5.788	4.267
	MRPCA [10]	2.972	0.922	3.728	1.117	5.009	1.963	0.669	0.099	2.008	1.033	5.775	4.081
	GLR [5]	2.959	1.052	3.773	1.306	4.909	2.114	0.696	0.161	1.587	0.830	3.839	2.707
	GPDNet[11]	3.780	1.337	8.007	4.426	13.482	9.114	1.913	1.037	5.021	3.736	9.705	7.998
	DMRD [7]	4.482	1.722	4.982	2.115	5.892	2.846	1.162	0.469	1.566	0.800	2.432	1.528
	Ours	**1.870**	**0.239**	**2.644**	**0.604**	**3.287**	**1.126**	**0.562**	**0.111**	**0.765**	**0.266**	**1.307**	**0.648**
PCNet dataset [12]	Bilateral [4]	4.320	1.351	6.171	1.646	8.295	2.392	1.172	0.198	2.478	0.634	6.077	2.189
	Jet [1]	3.032	0.830	5.298	1.372	7.650	2.227	1.091	0.180	2.582	0.700	5.787	2.144
	MRPCA [10]	3.323	0.931	4.874	1.178	6.502	1.676	0.966	0.140	2.153	0.478	5.570	1.976
	GLR [5]	3.399	0.956	5.274	1.146	7.249	1.674	0.964	**0.134**	2.015	0.417	4.488	1.306
	GPDNet [11]	5.470	1.973	10.006	3.650	15.521	6.353	5.310	1.716	7.709	2.859	11.941	5.130
	DMRD [7]	6.602	2.152	7.145	2.237	8.087	2.487	1.566	0.350	2.009	0.485	2.993	0.859
	Ours	**2.747**	**0.536**	**4.046**	**0.788**	**4.921**	**1.093**	**0.877**	0.144	**1.173**	**0.259**	**1.816**	**0.445**

Table 1. Quantitative filtering results of conventional methods and older learning based methods on the synthetic PUNet and PCNet datasets with Gaussian noise. CD and P2M values are multiplied by 10^4.

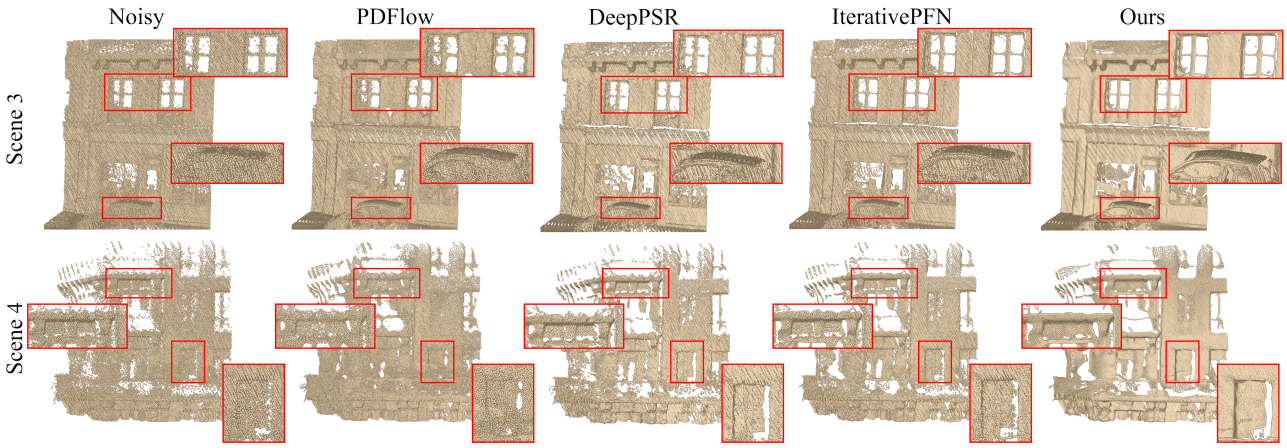

Figure 1. Additional visual results on the real-world Paris-Rue-Madame dataset.

at scales of 1%, 2% and 3% of the bounding sphere's radius. In general, we see that these methods perform suboptimally, with relatively higher CD and P2M errors. Furthermore, conventional methods require hyper-parameter tuning to obtain the best possible results. This is a tedious process which deep learning based methods help remedy.

B.2. Additional visual results on real-world scanned data

Fig. 1 illustrates filtering results on Scene 3 and Scene 4 of the Paris-Rue-Madame dataset. We observe from the close-ups in Scene 3 and 4 that PDFlow and DeepPSR leave behind high amounts of noise near the building windows. By comparison, IterativePFN fairs better but is not able to completely filter these noisy artifacts. StraightPCF is able to recover cleaner, smoother surfaces and removes a higher proportion of noise. This is evident in the close-ups of the windows and the close-up of the car roof. Further evaluation on real-world outdoor scenes is provided in Fig. 2 which illustrates filtering results on 4 scenes of the Kitti-360 dataset [6]. This dataset contains point clouds at a high sparsity setting and high noise level. While other methods leave behind noisy remnants, StraightPCF provides smoothly filtered surfaces. Moreover, the points are better distributed, as compared with DeepPSR which leaves behind clustering artifacts. Next, Fig. 3 demonstrates visual filtering results

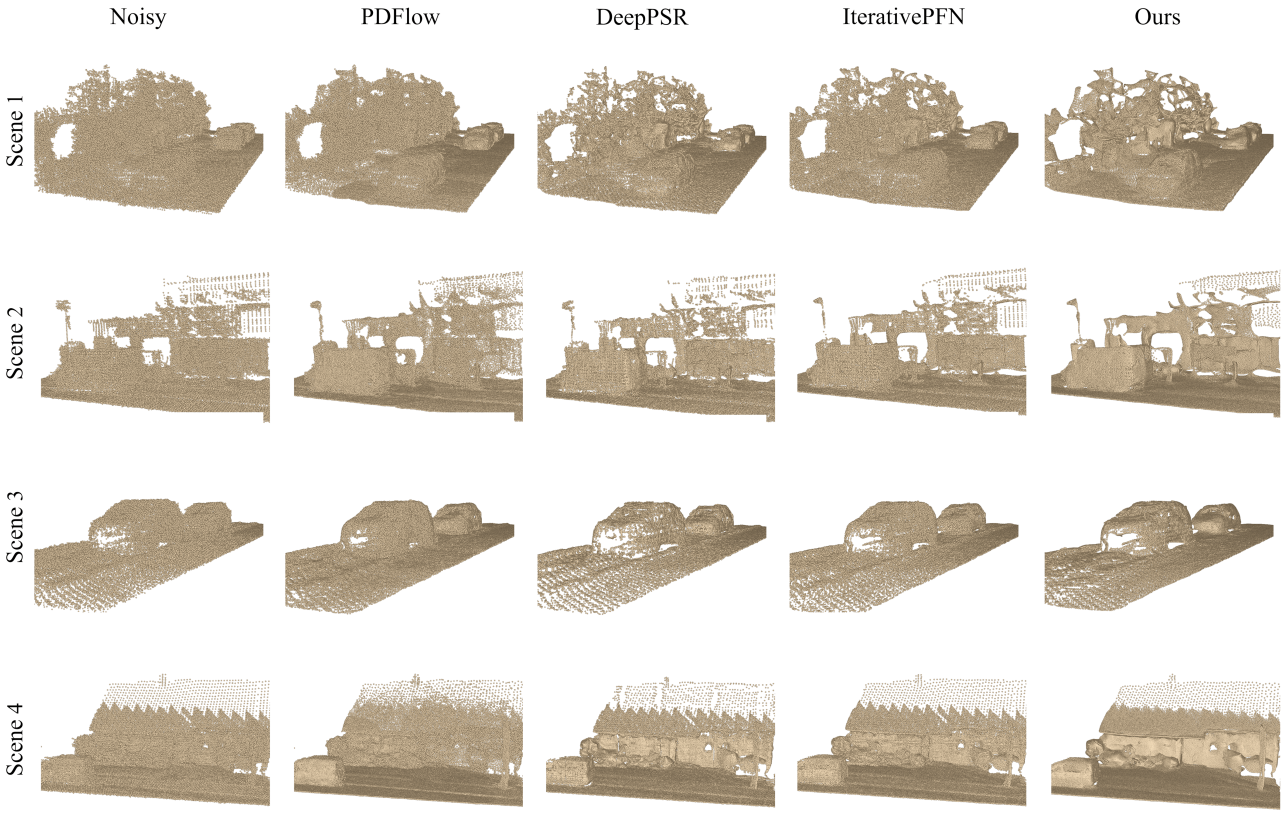

Figure 2. Visual results on 4 scans of the real-world Kitti-360 data.

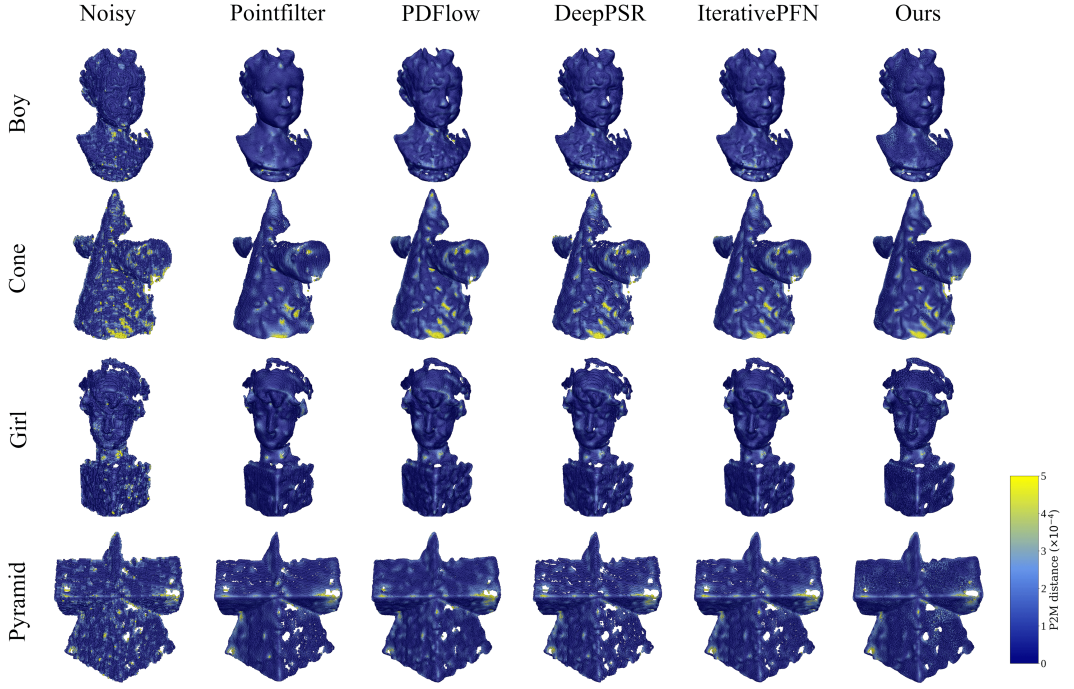

Figure 3. Visual filtering results on 4 scans of the real-world Kinect data. We outperform other methods and recover better distributed points with fewer, and smaller, holes.

	Resolution	10K (Sparse)						50K (Dense)					
	Noise	1%		2%		3%		1%		2%		3%	
	Method	CD	P2M	CD	P2M	CD	P2M	CD	P2M	CD	P2M	CD	P2M
PUNet [14]	Score [8]	2.479	0.463	3.701	1.098	4.764	1.984	0.711	0.149	1.319	0.593	2.087	1.162
	PointFilter [15]	2.399	0.433	3.529	0.888	5.240	2.001	0.757	0.185	0.964	0.301	1.918	0.963
	PDFlow [9]	2.103	0.383	3.293	1.060	4.640	2.182	0.653	0.167	1.197	0.604	2.080	1.349
	DeepPSR [2]	2.377	0.328	3.400	0.813	4.179	1.384	0.644	0.075	1.036	0.345	**1.521**	**0.692**
	IterativePFN [3]	1.994	**0.212**	3.030	**0.567**	4.611	1.678	0.602	**0.061**	0.830	**0.205**	2.425	1.377
	Ours	**1.834**	0.238	**2.640**	0.617	**3.479**	**1.292**	**0.556**	0.108	**0.798**	0.296	1.662	0.926
PCNet [12]	Score [8]	3.380	0.831	5.168	1.218	6.816	1.980	1.079	0.178	1.740	0.394	2.692	0.761
	PointFilter [15]	3.016	0.878	4.949	1.329	7.482	2.225	1.069	0.194	1.457	0.310	2.720	0.691
	PDFlow [9]	3.246	0.607	4.722	0.993	6.390	1.738	0.988	0.159	1.734	0.478	2.828	0.811
	DeepPSR [2]	3.146	0.993	4.943	1.342	6.307	1.860	1.011	0.173	1.630	0.432	2.332	0.724
	IterativePFN [3]	**2.639**	0.694	4.492	1.036	6.534	1.910	0.922	**0.142**	1.323	**0.272**	3.046	0.987
	Ours	2.757	**0.543**	**4.080**	**0.830**	**5.209**	**1.329**	**0.883**	**0.148**	**1.229**	0.290	**2.226**	**0.654**

Table 2. Quantitative filtering results of recent methods and our method on the synthetic datasets with non-isotropic Gaussian noise. Our network is lightweight, with just $\sim 530K$ parameters (17% of IterativePFN). CD and P2M values are multiplied by 10^4.

	Resolution	10K (Sparse)						50K (Dense)					
	Noise	1%		2%		3%		1%		2%		3%	
	Method	CD	P2M	CD	P2M	CD	P2M	CD	P2M	CD	P2M	CD	P2M
PUNet [14]	Score [8]	2.903	0.670	4.602	1.812	6.295	3.233	0.825	0.233	1.677	0.885	2.669	1.645
	PointFilter [15]	2.773	0.571	4.250	1.366	7.668	3.905	0.827	0.230	1.244	0.491	2.876	1.800
	PDFlow [9]	2.538	0.574	4.319	1.861	7.817	4.819	0.821	0.296	1.507	0.860	4.532	3.520
	DeepPSR [2]	2.725	0.488	3.915	1.218	4.982	**1.981**	0.740	0.139	1.279	0.526	**1.835**	**0.890**
	IterativePFN [3]	2.393	**0.315**	3.396	**0.806**	6.489	3.131	0.653	**0.089**	0.999	**0.317**	3.670	2.411
	Ours	**2.164**	0.338	**2.982**	0.872	**4.574**	2.163	**0.602**	0.133	**0.966**	0.412	2.536	1.676
PCNet [12]	Score [8]	3.968	0.954	6.193	1.650	8.395	2.618	1.204	0.227	2.058	0.509	3.379	1.013
	PointFilter [15]	3.539	0.985	5.732	1.589	9.821	3.273	1.155	0.227	1.723	0.381	3.536	1.099
	PDFlow [9]	3.760	0.713	5.778	1.404	9.245	2.869	1.166	0.214	2.107	0.544	4.831	1.453
	DeepPSR [2]	3.687	1.133	5.452	1.573	7.142	2.166	1.121	0.226	1.830	0.484	**2.574**	**0.740**
	IterativePFN [3]	3.189	0.801	4.894	1.165	8.173	2.517	0.993	0.172	1.477	**0.288**	3.799	1.138
	Ours	**3.156**	**0.602**	**4.468**	**0.940**	**6.099**	**1.548**	**0.936**	**0.161**	**1.399**	0.332	2.916	0.892

Table 3. Quantitative filtering results of recent methods and our method on the synthetic datasets with Laplace noise. Our network is lightweight, with just $\sim 530K$ parameters (17% of IterativePFN). CD and P2M values are multiplied by 10^4.

on the Kinect data. Methods such as PDFlow, DeepPSR and IterativePFN perform poorly on scans such as Boy and Pyramid, in comparison to StraightPCF. Furthermore, Pointfilter leaves behind a large number of holes on the Pyramid scan whereas StraightPCF recovers a cleaner filtered version of the Pyramid with fewer, and smaller, holes.

C. Further Evaluation on PUNet and PCNet Data Under Different Noise Patterns

Similar to the analysis of ScoreDenoise [8], DeepPSR [2], PDFlow [9] and IterativePFN [3], we look at quantitative and visual results on synthetic data under different noise patterns. More specifically, we look at noise patterns that have the following distributions:

1) Non-isotropic Gaussian distribution where the covariance matrix is given by:

$$\Sigma = s^2 \times \begin{bmatrix} 1 & -1/2 & -1/4 \\ -1/2 & 1 & -1/4 \\ -1/4 & -1/4 & 1 \end{bmatrix} \tag{3}$$

The noise scale s is set to 1%, 2% and 3% of the bounding sphere's radius. In contrast to the synthetic data presented in the main paper, which contain isotropic Gaussian noise,

	Resolution	10K (Sparse)						50K (Dense)					
	Noise	1%		2%		3%		1%		2%		3%	
	Method	CD	P2M	CD	P2M	CD	P2M	CD	P2M	CD	P2M	CD	P2M
PUNet [14]	Score [8]	1.274	0.249	2.467	0.414	3.366	0.977	0.505	0.046	0.691	0.129	0.900	0.288
	PointFilter [15]	1.139	0.287	2.447	0.407	3.031	0.574	0.631	0.148	0.742	0.171	0.798	0.188
	PDFlow [9]	0.874	0.178	2.026	0.332	2.660	0.637	0.456	0.057	0.854	0.326	0.953	0.400
	DeepPSR [2]	1.226	0.190	2.416	0.299	2.969	0.472	0.497	0.027	0.638	0.064	0.928	0.245
	IterativePFN [3]	0.645	**0.088**	2.009	**0.191**	2.684	**0.339**	0.443	**0.014**	0.599	0.054	0.694	**0.110**
	Ours	**0.624**	0.097	**1.834**	0.252	**2.365**	0.427	**0.418**	0.050	**0.557**	0.130	**0.683**	0.228
PCNet [12]	Score [8]	1.789	0.692	3.232	0.795	4.767	1.258	0.714	0.104	1.044	0.164	1.342	0.308
	PointFilter [15]	1.414	0.792	2.926	0.853	4.032	0.999	0.774	0.155	1.060	0.181	1.182	0.210
	PDFlow [9]	2.038	0.506	3.108	0.582	3.954	0.788	0.727	0.096	1.245	0.329	1.374	0.361
	DeepPSR [2]	1.610	0.954	3.065	1.014	4.146	1.129	0.661	0.110	0.991	0.157	1.306	0.270
	IterativePFN [3]	**0.989**	0.598	**2.495**	0.668	3.713	0.805	**0.578**	**0.089**	0.918	**0.134**	1.074	**0.186**
	Ours	1.399	**0.477**	2.917	**0.581**	**3.677**	**0.700**	0.678	0.103	**0.906**	0.175	**1.058**	0.240

Table 4. Quantitative filtering results of recent methods and our method on the synthetic datasets with uniform noise. Our network is lightweight, with just $\sim 530K$ parameters (17% of IterativePFN). CD and P2M values are multiplied by 10^4.

here we look at Gaussian noise that is anisotropically distributed. Table 2 and Fig. 4 provide quantitative and visual results on this noise pattern. Our method outperforms others across most resolution and noise settings on both the PUNet and PCNet datasets.

2) Laplace distribution where the noise scale s is set to 1%, 2% and 3% of the bounding sphere's radius. Table 3 and Fig. 5 provide quantitative and visual results on this noise pattern. We note that the noise intensity for this noise pattern is generally higher than the case of Gaussian noise. However, our method satisfactorily filters synthetic data across both PUNet and PCNet datasets. IterativePFN and DeepPSR obtain competitive P2M results, yet induce clustering which can be seen from the formation of small holes on the Camel and Netsuke shapes of Fig. 5. We also observe that while our StraightPCF method interpolates between Gaussian high noise variant patches and underlying clean patches during training time, the results on the Laplace distributed synthetic data demonstrates the high generalizability of our method.

3) Uniform distribution of noise within a sphere of radius s. The probability to sample noise at a position $\boldsymbol{x}$ within the sphere is given by,

$$p(\boldsymbol{x}; s) = \begin{cases} \frac{3}{4\pi s^3}, & \|\boldsymbol{x}\|_2 \leq s, \\ 0, & \text{Otherwise} \end{cases} \tag{4}$$

where the noise scale s is set to 1%, 2% and 3% of the bounding sphere's radius. This noise distribution is not unimodal unlike the previous distributions and generally has a lower noise intensity than that of Gaussian noise. Table 4 and Fig. 6 provide quantitative and visual results on this noise pattern. Overall, StraightPCF consistently outperforms others on the Chamfer Distance metric, indicating its ability to recover a distribution of points closer to that of the clean point cloud. Furthermore, analysis of the visual results again reinforces the conclusion that while some methods such as IterativePFN may yield lower P2M errors, they cause points to cluster and leave behind small holes.

Method	Time (s)
PCN	186.7
ScoreDenoise	15.8
Pointfilter	100.8
PDFlow	53.8
DeepPSR	8.99
IterativePFN	19.7
Ours	18.2

Table 5. Runtimes of state-of-the-art methods on point clouds with 50K points and 2% Gaussian noise, from the PUNet dataset.

D. Comparison of Test Times for Different Methods

Table 5 provides runtimes for StraightPCF, and other state-of-the-art methods, on synthetic point clouds at 50K resolution and $\sigma = 2\%$. We obtained the runtime for each method by filtering 3 point clouds at 50K resolution and $\sigma = 2\%$, from the PUNet dataset, and taking the mean of the total runtime. In general, DeepPSR is the fastest method but its overall performance across synthetic and scanned data is sub-optimal. To improve performance results, DeepPSR

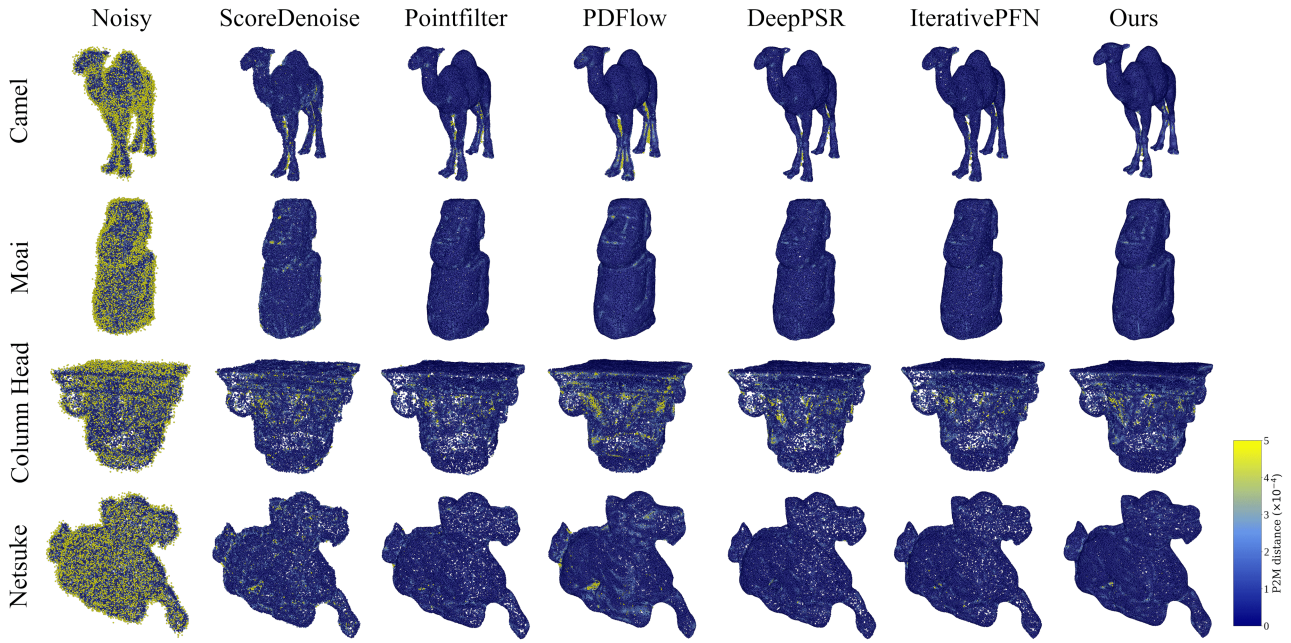

Figure 4. Visual results for 50K resolution shapes with non-isotropic Gaussian noise and noise scale $s = 2\%$ of the bounding sphere radius.

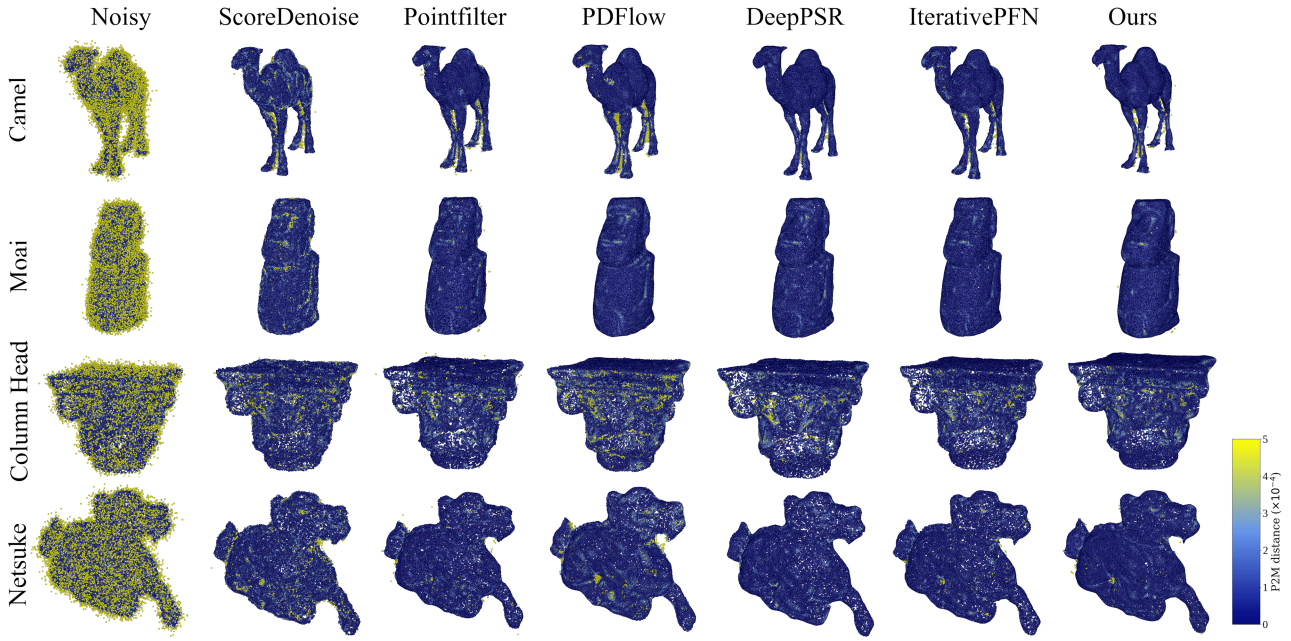

Figure 5. Visual results for 50K resolution shapes with Laplace noise and noise scale $s = 2\%$ of the bounding sphere radius.

would potentially need to filter point clouds for additional iterations, leading to higher runtimes. The same is true for ScoreDenoise. However, we obtain state-of-the-art performance with highly competitive runtimes. As mentioned previously, StraightPCF is much more lightweight than IterativePFN, its closest competitor in terms of performance. It has only $\sim 530K$ parameters compared to IterativePFN's $\sim 3.2M$ parameters.

E. Further Ablation Studies

Tables 6 and 7 provide ablation results on higher VelocityModule (VM) couplings K, with and without the Dis-

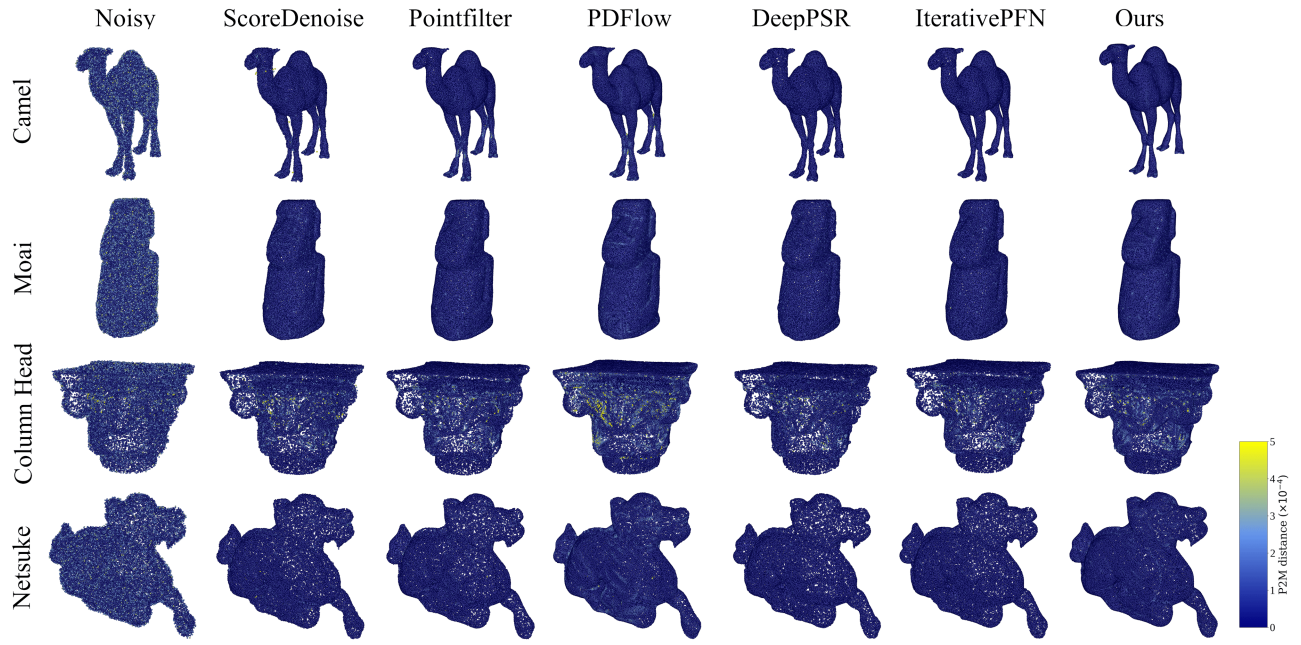

Figure 6. Visual results for 50K resolution shapes with uniformly distributed noise and noise scale $s = 2\%$ of the bounding sphere radius.

Ablation: PUNet	10K points					
	1% noise		2% noise		3% noise	
	CD	P2M	CD	P2M	CD	P2M
***V5*) CVM w/ DM**	1.87	**0.24**	2.64	0.60	3.29	1.13
V6) 3VM w/o DM	1.89	0.27	2.81	0.76	3.43	1.25
V7) 3VM w/ DM	1.87	**0.24**	2.66	0.61	**3.23**	**1.09**
V8) 4VM w/o DM	1.90	0.28	2.87	0.83	3.52	1.34
V9) 4VM w/ DM	**1.86**	**0.24**	**2.62**	**0.59**	3.38	1.20

Table 6. Ablation results for higher VelocityModule (VM) couplings K, with and without the DistanceModule (DM). CD and P2M distances are multiplied by 10^4.

Ablation: PUNet	10K points					
	1% noise		2% noise		3% noise	
	CD	P2M	CD	P2M	CD	P2M
$N = 1$	2.03	0.30	2.84	0.76	3.44	1.26
$N = 3$	1.87	**0.24**	2.64	0.60	**3.29**	**1.13**
$N = 5$	**1.86**	**0.24**	**2.62**	**0.59**	3.33	1.15

Table 7. Ablation on different numbers of Euler steps N. Based on the above results, we see that $N = 3$ is the optimal number of Euler steps as it provides a good balance between performance and efficiency. CD and P2M distances are multiplied by 10^4.

tanceModule, and the impact of tuning the number of Euler steps N, respectively. We see that the optimal architecture is V5 as it only contains $K = 2$ VMs and is smaller than

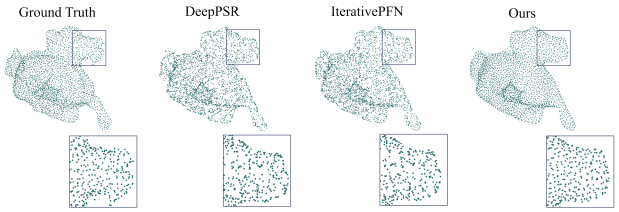

Figure 7. Visual results for the 3K resolution Netsuke shape with Gaussian noise and $\sigma = 3\%$ of the bounding sphere radius. Filtering sparse point clouds at high noise is a challenge for Straight-PCF. However, it recovers a better distribution of points, as compared with other state-of-the-art methods.

the other variants, leading to a faster runtime. Generally, it also provide either the best or second best results on the ablation set. The results of Tables 6 further reinforces the importance of the DistanceModule to ensure convergence at the clean surface, especially as noise increases. Moreover, Table 7 demonstrates the importance of Euler steps N. For $N = 1$, we obtain sub-optimal results. For $N = 3$, we strike a balance between performance and runtime efficiency as runtime increases as N increases. Also, we see only a marginal improvement in performance for $N > 3$.

F. Discussion of Limitations

Finally, Fig. 7 illustrates the visual filtering results for the Netsuke shape with 3K resolution and $\sigma = 3\%$. We see that at this high sparsity setting, it is difficult to recover high levels of geometric detail. However, our method performs bet-

ter than other state-of-the-art methods and recovers a nicer distribution of points.

References

[1] Frederic Cazals and Marc Pouget. Estimating differential quantities using polynomial fitting of osculating jets. *Computer Aided Geometric Design*, 22(2):121–146, 2005. 2

[2] H Chen, B Du, S Luo, and W Hu. Deep point set resampling via gradient fields. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 45:2913–2930, 2023. 1, 4, 5

[3] Dasith de Silva Edirimuni, Xuequan Lu, Zhiwen Shao, Gang Li, Antonio Robles-Kelly, and Ying He. Iterativepfn: True iterative point cloud filtering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13530–13539, 2023. 4, 5

[4] S. Fleishman, Iddo Drori, and D. Cohen-Or. Bilateral mesh denoising. *ACM SIGGRAPH 2003 Papers*, 2003. 2

[5] W Hu, X Gao, G Cheung, and Z Guo. Feature graph learning for 3d point cloud denoising. *IEEE Transactions on Signal Processing*, 68:2841–2856, 2020. 2

[6] Yiyi Liao, Jun Xie, and Andreas Geiger. KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *arXiv preprint arXiv:2109.13410*, 2021. 2

[7] Shitong Luo and Wei Hu. Differentiable manifold reconstruction for point cloud denoising. In *Proceedings of the 28th ACM International Conference on Multimedia*, page 1330–1338. Association for Computing Machinery, 2020. 2

[8] Shitong Luo and Wei Hu. Score-based point cloud denoising. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4583–4592, 2021. 1, 4, 5

[9] Aihua Mao, Zihui Du, Yu-Hui Wen, Jun Xuan, and Yong-Jin Liu. Pd-flow: A point cloud denoising framework with normalizing flows. In *The European Conference on Computer Vision (ECCV)*, 2022. 1, 4, 5

[10] E Mattei and A Castrodad. Point cloud denoising via moving rpca. *Computer Graphics Forum*, 36:123–137, 2017. 2

[11] Francesca Pistilli, Giulia Fracastoro, Diego Valsesia, and Enrico Magli. Learning graph-convolutional representations for point cloud denoising. In *Computer Vision – ECCV 2020*, pages 103–118. Springer International Publishing, 2020. 2

[12] Marie-Julie Rakotosaona, Vittorio La Barbera, Paul Guerrero, N. Mitra, and M. Ovsjanikov. Pointcleannet: Learning to denoise and remove outliers from dense point clouds. *Computer Graphics Forum*, 39, 2020. 2, 4, 5

[13] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *ArXiv*, 2020. 1

[14] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-net: Point cloud upsampling network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 4, 5

[15] Dongbo Zhang, Xuequan Lu, Hong Qin, and Y. He. Pointfilter: Point cloud filtering via encoder-decoder modeling. *IEEE Transactions on Visualization and Computer Graphics*, 27:2015–2027, 2021. 4, 5