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

Abstract

Point cloud filtering is a fundamental 3D vision task, which aims to remove noise while recovering the underlying clean surfaces. State-of-the-art methods remove noise by moving noisy points along stochastic trajectories to the clean surfaces. These methods often require regularization within the training objective and/or during post-processing, to ensure fidelity. In this paper, we introduce StraightPCF, a new deep learning based method for point cloud filtering. It works by moving noisy points along straight paths, thus reducing discretization errors while ensuring faster convergence to the clean surfaces. We model noisy patches as intermediate states between high noise patch variants and their clean counterparts, and design the VelocityModule to infer a constant flow velocity from the former to the latter. This constant flow leads to straight filtering trajectories. In addition, we introduce a DistanceModule that scales the straight trajectory using an estimated distance scalar to attain convergence near the clean surface. Our network is lightweight and only has ∼ 530K parameters, being 17% of IterativePFN (a most recent point cloud filtering network). Extensive experiments on both synthetic and real-world data show our method achieves state-of-the-art results. Our method also demonstrates nice distributions of filtered points without the need for regularization. The implementation code can be found at: https://github.com/ddsediri/StraightPCF.

1. Introduction

In recent years, point clouds have become increasingly popular as the representation-of-choice for storing and manipulating 3D data, with numerous applications in both computer vision [12, 20, 25, 38] and geometry modelling [16, 25, 40]. Point clouds are unordered sets of 3D coordinates which typically represent object surfaces and are captured using 3D sensors such as depth and Lidar devices. However,
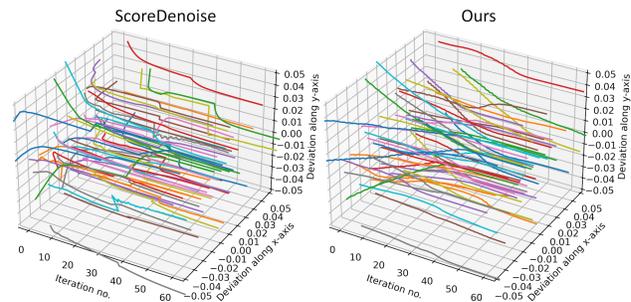
Figure 1. Filtered trajectories for the Isocahedron shape at 50K resolution and noise scale $\sigma = 3\%$. Our StraightPCF filters points along much straighter paths, compared to ScoreDenoise [28].

noisy artifacts may appear in point clouds as a result of sensor limitations and environmental factors. Removing this noise, known as filtering or denoising, is a fundamental 3D vision task. Filtering methods are broadly categorized into two groups: 1) conventional methods involving traditional optimization and 2) deep learning based methods. Conventional methods can further be subdivided into normal based methods which require surface normal information to reliably filter point clouds [2, 10, 13, 24, 30] and point based methods which directly filter point clouds [15, 21, 32]. While normal based methods are limited by the accuracy of normals, point based methods suffer from a loss of geometric details and still exhibit sensitivity to noise. Recently, many deep learning approaches have been proposed to overcome these shortcomings.

Deep learning methods can be divided into 1) **resampling**, 2) **displacement** and 3) **probability** based methods. Resampling-based methods [26] show the least fidelity when recovering the underlying noise-free surfaces as their downsampling procedure results in the loss of crucial geometric information. By contrast, displacement and probability based methods show greater promise as they model the filtering objective as a reverse Markov process, which can be iteratively applied on the input to progressively remove noise. Early displacement based methods, such as

PointCleanNet [7, 35, 45], employ large networks (>1M parameters) that consume a large patch of points to filter a single, central point. This is time-consuming and resource intensive. More recently, IterativePFN [8] models the iterative filtering process internally using multiple *Iteration-Modules* and filters all patch points simultaneously. However, incorporating multiple IterationModules results in a large network (>3.2M parameters) which requires a large amount of GPU memory to process. Probabilistic score based methods such as ScoreDenoise [6, 28] offer a more lightweight network but require a high number of iterations to recover the clean surface. At higher noise levels, they converge to surfaces that retain noticeable amounts of noise. DeepPSR [6] utilizes the score module of [28] but performs an additional Graph Laplacian Regularization [14] step on intermediate point clouds to obtain a better point distribution. More importantly, the iterative filtering objectives of current displacement and probabilistic methods move noisy points to the clean surface via stochastic trajectories and are error prone as incorrectly inferred displacements may result in significant changes to the distribution of filtered points.

We propose StraightPCF, which moves noisy points along straight filtering trajectories towards the clean surfaces, as illustrated in Fig. 1. StraightPCF, illustrated in Fig. 2, is a lightweight network ($\sim 530K$ parameters, 17% of IterativePFN [8]) which filters all patch points simultaneously. Our technical contributions are as follows.

- We introduce the patch-wise **VelocityModule** that infers constant, straight flows to filter point cloud patches. The VelocityModule elegantly recovers the underlying clean surfaces with nice point distributions.

- To improve straightness of flows, we propose a novel straightening mechanism consisting of coupled VelocityModules. This **coupled VelocityModule stack** infers straighter filtering trajectories, leading to better results.

- Constant flows may lead to filtered points overshooting the clean surface. Therefore, we design the **Distance-Module** to infer a distance scalar that provides a magnitude to scale the flow velocity. Our architecture is the first to decompose filtering into a dual objective of inferring a vector field of flow velocities and a distance scalar.

2. Related Work

Conventional filtering. Early conventional methods were inspired by the Moving Least Squares (MLS) method of Levin [17] and require normal information for filtering. Notably, Alexa *et al.* [2] employed MLS optimization in recovering denoised surfaces from noisy point sets. The Implicit MLS (IMLS) method of Adamson and Alexa [1] further extended this approach to point-sampled cell complexes which allow for a well defined local geometry. The Algebraic Point Set Surface (APSS) method of Guennebaud and Gross [13] applied MLS optimization for the purpose of

fitting algebraic spheres to recover surfaces while being robust to point set density and underlying curvature. Furthermore, Digne proposed filtering the height map associated to a point set by considering handcrafted features that encode height variations around each point [9]. Digne and de Franchis designed a weighted projection scheme, that moves points to their filtered positions [10], based on the mesh bilateral filtering method of Fleishman, Drori and Cohen-Or [11]. Other normal based methods include the Moving Robust Principal Component Analysis (MRPCA) [30] of Mattei and Castrodad, the Graph Laplacian Regularization (GLR) [14] technique of Hu *et al.* and the Low Rank Matrix Approximation [24] of Lu *et al.* The main drawback of such methods is susceptibility to noise, during both the normal estimation and filtering steps.

By contrast, point based methods employ only point information for filtering. Cazals and Pouget [4] proposed a N-dimensional polynomial surface fitting method that can be used to filter points. Meanwhile, the Locally Optimal Projection (LOP) [21] method of Lipman *et al.* downsampled and regularized noisy point clouds. It was extended by Huang *et al.* and Preiner *et al.* who developed Weighted-LOP (WLOP) [15] and Continuous-LOP (CLOP) [32], respectively. However, these methods do not effectively recover geometric details due to their downsampling step.

Deep learning based filtering. While conventional methods rely on handcrafted features, convolutional neural network architectures have provided great improvements to feature generation. PointProNets [37] by Roveri *et al.* and Deep Feature Preserving (DFP) [23] by Lu *et al.* projected points onto 2D height maps before processing them using CNNs. PointNet [33], introduced by Qi *et al.*, set the precedent for direct point set convolution and was later improved by PointNet++ [34]. Meanwhile, DGCNN [42] of Wang *et al.* reformulated point set convolution as a graph convolution task. PointCleanNet (PCN) [35] was one of the first methods to adopt the PointNet architecture and inferred the filtered displacement of a single central point by considering its neighborhood patch. Pointfilter [45] of Zhang *et al.* furthered this line of research using a bilateral filtering-inspired loss.

Pistilli *et al.* introduced the first graph convolution-based mechanism, named GPDNet [31], while Luo and Hu proposed the DGCNN based DMRDenoise [26]. DMRDenoise filtered points by downsampling noisy inputs and upsampling these less noisy surfaces. Luo and Hu also proposed ScoreDenoise [28] where they formally expressed the filtering objective as the backward Langevin equation that iteratively removes noise using the inferred gradient-log of the probability distribution, i.e., the score, for point positions $\boldsymbol{x}$. This was extended by Chen *et al.* in their DeepPSR [6] which employs an additional graph laplacian regularization post-processing step. Mao *et al.* introduced the normalizing
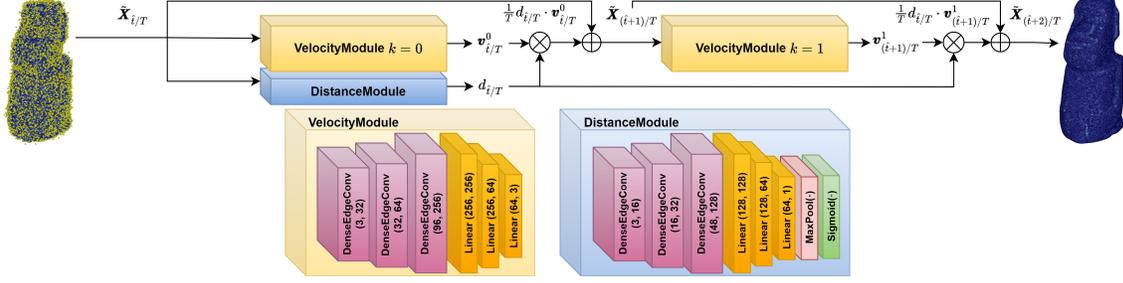
Figure 2. Our StraightPCF network. It involves a coupled VelocityModule stack that infers a constant flow velocity $\boldsymbol{v}_\theta^k$ for patch states $\tilde{\boldsymbol{X}}_{(\hat{t}+k)/T}$. To ensure filtered points converge to the surface, the DistanceModule infers a distance scalar d_ϕ that scales the velocity.

flows based filtering method PDFlow [29] that disentangles noise from the underlying clean representation at higher dimensions. The RePCDNet [5] method of Chen *et al.* sought to model iterative filtering via a recurrent neural network. By contrast, de Silva Edirimuni *et al.* proposed the graph convolution based IterativePFN [8] that models iterative filtering using individual IterationModules. The joint filtering and normal estimation method CFilter [7], developed by de Silva Edirimuni *et al.*, exploited normals to improve filtered point positions. Ma *et al.* introduced learning implicit signed distance functions, by displacing noisy query points back to the surface along normals where the surface corresponds to the zero-level set of the function [3]. This has been exploited for normal estimation [18, 19].

Score and displacement based methods are inspired by diffusive processes [27, 39] and their filtering objectives result in stochastic trajectories. Recent work by Liu, Gong and Liu focuses on Reflow [22], which explored the optimal transport problem of identifying straight paths given two samples from different distributions. The work of Wu *et al.* further extended this Reflow mechanism to point cloud generation, which attempts to generate point cloud shapes given initial samples from the normal distribution [43].

3. Problem Statement and Motivation

Resampling methods demonstrate an inability to recover clean surfaces with high accuracy, unlike probabilistic and displacement based methods. A probabilistic score based method such as ScoreDenoise [28] and a displacement based method such as IterativePFN [8] share similar filtering objectives that may be expressed by,

$$\tilde{\boldsymbol{x}}_t^i = \tilde{\boldsymbol{x}}_{t-1}^i + F_\theta(\tilde{\boldsymbol{x}}_{t-1}^i), t = 1, \ldots, T \quad (1)$$

where $\tilde{\boldsymbol{x}}_t^i$ is the i-th filtered point at time t and F_θ represents the network with parameters θ. The main difference between ScoreDenoise and IterativePFN lies in their training objectives, $\mathcal{L}_S$ and $\mathcal{L}_I$, respectively. For a noisy point $\boldsymbol{x}^i \in \boldsymbol{X}$ where $\boldsymbol{X}$ is the initial noisy patch, we have:

$$\mathcal{L}_S^i = \mathbb{E}_{\boldsymbol{x} \sim kNN(\boldsymbol{x}^i, \boldsymbol{X})} \left[\left\| s(\boldsymbol{x}) - \mathcal{S}^i(\boldsymbol{x}) \right\|_2^2 \right], \quad (2)$$

$$\mathcal{L}_I^i = \sum_{t=0}^{T} \left\| \boldsymbol{d}_t^i - \left(NN(\boldsymbol{x}_{t-1}^i, \boldsymbol{Y}_t) - \boldsymbol{x}_{t-1}^i \right) \right\|_2^2. \quad (3)$$

ScoreDenoise is trained to predict a score $\mathcal{S}^i(\boldsymbol{x}) = \text{Score}(\boldsymbol{x} - \boldsymbol{x}^i | \boldsymbol{h}^i)$ for $\boldsymbol{x}$ sampled from the k-nearest neighbors of $\boldsymbol{x}^i$. It is conditioned on the latent feature $\boldsymbol{h}^i$ of $\boldsymbol{x}^i$ and corresponds to the gradient-log of the noise-convolved probability distribution for point position $\boldsymbol{x}$. The ground truth target $s(\boldsymbol{x}) = NN(\boldsymbol{x}, \boldsymbol{Y}) - \boldsymbol{x}$, where $\boldsymbol{Y}$ is the clean patch. Ergo, $s(\boldsymbol{x})$ is the displacement from $\boldsymbol{x}$ to its nearest neighbor in $\boldsymbol{Y}$. Once this training objective has been optimized, score based methods infer aggregate scores, at each iteration t, such that $F_\theta(\boldsymbol{x}_{t-1}^i) = \alpha \mathcal{E}^i(\boldsymbol{x}_{t-1}^i) = (\alpha/K) \sum_{\boldsymbol{x}_{t-1}^j \in kNN(\boldsymbol{x}_{t-1}^i)} \mathcal{S}^j(\boldsymbol{x}_{t-1}^i)$ where α is the step discretization parameter. This score shifts points towards the clean surface along a stochastic filtered trajectory that is sensitive to the choice of α, which must be kept relatively small (~ 0.2). By contrast, IterativePFN internally models the iterative filtering objective and incorporates it within their training objective. They directly infer displacements $F_\theta(\tilde{\boldsymbol{x}}_{t-1}^i) = \boldsymbol{d}_t^i$ as the output of each IterationModule that models an iteration t. From Eq. (3) we identify that the overall filtered trajectories inferred by IterativePFN will also be stochastic in nature as it uses an Adaptive Ground Truth (AGT), $\boldsymbol{Y}_t = \boldsymbol{Y} + \sigma_t \xi \wedge \xi \sim \mathcal{N}(0, I)$, within the training objective. Modelling the iterative filtering process internally results in a large network (>3.2M parameters). The AGT, while recovering the surface, causes clustering along the surface and fails to respect the original point distribution. This is illustrated in Fig. 3.

Inspired by Reflow which focuses on optimal transport between samples of different distributions [22, 43], we pose filtering as an optimal transport problem between point cloud patches. These patches of n points are sampled from 1) the clean surface (i.e., clean patch $\boldsymbol{X}_1 \sim \pi_1$) and 2) a high noise variant of the clean surface (i.e., high noise patch $\boldsymbol{X}_0 \sim \pi_0$). The goal is to determine a transport plan (coupling) such that $\boldsymbol{X}_1 = V(\boldsymbol{X}_0)$ where $V : \mathbb{R}^{n \times 3} \to \mathbb{R}^{n \times 3}$. The approximated flow velocity, $\boldsymbol{v}_\theta : \mathbb{R}^{n \times 3} \to \mathbb{R}^{n \times 3}$ sat-
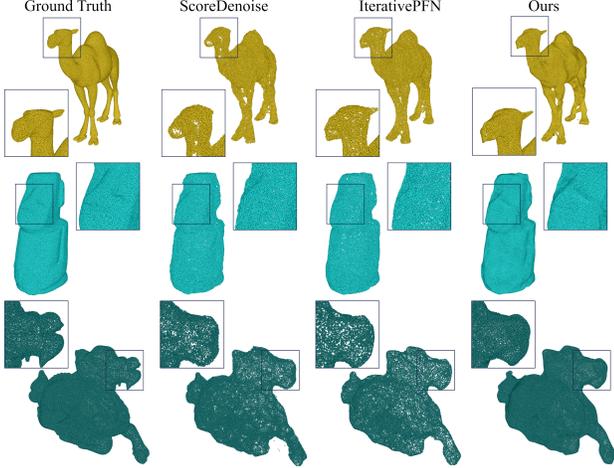
Figure 3. Our StraightPCF is able to recover a better distribution of filtered points, even at very high noise scales ($\sigma = 3\%$ and 50K resolution) unseen during network training.

Figure 4. StraightPCF models initial noisy patches (light blue) as being intermediate states of a linear interpolation between high noise variants (red) and the clean surfaces (dark blue) and encourages straight filtering trajectories.

isfies this transport mapping and $(\tilde{\boldsymbol{X}}_0, \tilde{\boldsymbol{X}}_1)$ form a valid coupling. Consequently, the filtered trajectories are straight (see Fig. 4), unlike ScoreDenoise and IterativePFN. Moreover, our method recovers the underlying point distribution of the surface without clustering artifacts or distortion.

4. Method

Previous methods such as ScoreDenoise [28] and IterativePFN [8] focus on filtering as a reverse Markov process where the forward process would correspond to adding noise. We formulate filtering as an optimal transport plan that moves noisy points back to the clean surface along the shortest (straight) path. We model noisy input patches as intermediate states between high noise variants and their corresponding clean counterparts. We design a graph-convolution based VelocityModule that infers a constant flow velocity for each intermediate state. This encourages noisy points to move along straight filtering trajectories as shown in Fig. 4. We further improve the straightness of these trajectories via VelocityModule coupling. Finally, as the flow is constant, it may result in filtered points over-shooting the clean surface. Therefore, we design a DistanceModule that scales the flow velocity appropriately and ensures convergence near the surface. The overall Straight-PCF architecture is illustrated in Fig. 2, which demonstrates the filtering process that utilizes both the coupled VelocityModule and DistanceModule sub-networks to move points along straightened paths. The *supplementary document* provides additional methodology details.

4.1. Filtering via straight flows

In this section, we introduce the VelocityModule (VM) that moves noisy points along constant, straight flows to-

wards the clean surface. Given a noisy patch $\boldsymbol{X} = \{\boldsymbol{x}^i | \boldsymbol{x}^i \in kNN(\boldsymbol{x}^r, \mathcal{P}_{\boldsymbol{X}}, k)\}$ centered around a reference point $\boldsymbol{x}^r$ in the noisy point cloud $\mathcal{P}_{\boldsymbol{X}}$, the filtering objective aims to move $\boldsymbol{x}^i$ towards the underlying clean patch $\boldsymbol{Y} = \{\boldsymbol{y}^i | \boldsymbol{y}^i \in \mathcal{P}_{\boldsymbol{Y}}\}$. For the filtering task, learning based methods are typically trained on noisy point clouds with Gaussian noise [6, 28, 29, 35, 45]. Iterative filtering depends on the ability of the method to filter patches from a higher noise level to a lower noise level, with the end result converging to the clean surface. Given the highest noise setting σ_H, intermediate noise scales at a time $t \in [0, 1]$ can be expressed as $\sigma = (1 - t)\sigma_H$. Therefore, we model noisy patches $\boldsymbol{X} = \boldsymbol{Y} + \sigma\xi \wedge \xi \sim \mathcal{N}(0, I)$ as intermediate states $\boldsymbol{X}_t$, of the filtering objective that moves points from a high noise variant, $\boldsymbol{X}_0 = \boldsymbol{Y} + \sigma_H\xi \wedge \xi \sim \mathcal{N}(0, I)$ to the clean counterpart $\boldsymbol{X}_1 = \boldsymbol{Y}$. Here, $\sigma_H = 2\%$ of the point cloud's bounding sphere radius and corresponds to the highest noise setting of our training set. We observe that intermediate states $\boldsymbol{X}_t$ can be defined as a linear interpolation of $\boldsymbol{X}_0$ and $\boldsymbol{X}_1$, that is, a straight path, such that,
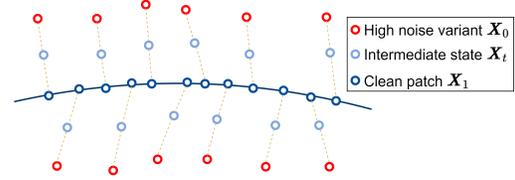
$$\boldsymbol{X}_t = (1 - t)\boldsymbol{X}_0 + t\boldsymbol{X}_1 \qquad (4)$$

Therefore, we intuit that the filtering objective can be reformulated as an optimal transport process that moves points from $\boldsymbol{X}_0$ to $\boldsymbol{X}_1$. The flow of $\boldsymbol{X}_t$ at time t can be expressed via the ordinary differential equation (ODE):

$$\mathrm{d}\boldsymbol{X}_t = \boldsymbol{v}(\boldsymbol{X}_t)\mathrm{d}t, \qquad (5)$$

where $\boldsymbol{v}(\boldsymbol{X}_t)$ is the velocity field at $\boldsymbol{X}_t$. For this ODE process, the linear interpolation of Eq. (4) can be used in the least squares optimization to determine the flow,

$$\min_{v} \int_0^1 \mathbb{E}_{t\sim\mathcal{U}(0,1)} \left[\|\boldsymbol{v}(\boldsymbol{X}_t) - (\boldsymbol{X}_1 - \boldsymbol{X}_0)\|_2^2 \right] \mathrm{d}t, \qquad (6)$$

where times t are sampled uniformly along $[0, 1]$. However, the velocity field at an intermediate state $\boldsymbol{X}_t$ cannot be causally determined as both $\boldsymbol{X}_0$ and $\boldsymbol{X}_1$ are unknown, during filtering. To address this, our idea is to approximate the velocity field for each $\boldsymbol{X}_t$ by a neural network.

Training objective for single VM. We use a DGCNN based graph neural network to model a VelocityModule $\boldsymbol{v}_\theta$,

with parameters θ to approximate the flow velocity. To train this VelocityModule, we define the following training objective.

$$\mathcal{L}_A = \mathbb{E}_{t \sim \mathcal{U}(0,1)} \left[\| \boldsymbol{v}_\theta(\boldsymbol{X}_t) - \boldsymbol{\delta}(\boldsymbol{X}_1, \boldsymbol{X}_0) \|_2^2 \right], \quad (7)$$

where $\boldsymbol{\delta}(\boldsymbol{X}_1, \boldsymbol{X}_0) = \boldsymbol{X}_1 - \boldsymbol{X}_0$. Hence, at each intermediate state, the corresponding flow velocity is ideally a constant vector that leads to a straight path from $\boldsymbol{X}_0$ to $\boldsymbol{X}_1$.

Filtering objective for single VM. During inference, given a straight flow velocity $\boldsymbol{v}_\theta$, we can move points from $\boldsymbol{X}_{\hat{t}/N}$ to $\boldsymbol{X}_{(\hat{t}+1)/N}$ using the Euler method ODE solver:

$$\tilde{\boldsymbol{X}}_{(\hat{t}+1)/N} = \tilde{\boldsymbol{X}}_{\hat{t}/N} + \frac{1}{N} \boldsymbol{v}_\theta(\tilde{\boldsymbol{X}}_{\hat{t}/N}), \quad (8)$$

where N denotes the total number of filtering steps, $\hat{t} \in [0, \dots, M, \dots, N-1]$ is an integer time step and $\tilde{\boldsymbol{X}}_0 = \boldsymbol{X}_0$. In practice, filtering starts at an unknown time step $\hat{t} = M$. We do not know the noise scale of the input patch and need to model it as an intermediate state $\tilde{\boldsymbol{X}}_{M/N}$ of a higher noise variant $\boldsymbol{X}_0$. The full position update takes the form,

$$\tilde{\boldsymbol{X}}_1 = \tilde{\boldsymbol{X}}_{M/N} + \frac{1}{N} \sum_{\hat{t}=M}^{N-1} \boldsymbol{v}_\theta(\tilde{\boldsymbol{X}}_{\hat{t}/N}), \quad (9)$$

Eq. (9) poses two challenges:
1. The straightness of paths is crucial to reducing the number of total steps and improving efficiency. If paths are not sufficiently straight (i.e., they are curved), we need a higher number of steps to filter points effectively.
2. The starting time $t = M/N$ is unknown. Applying the Euler method for too many steps, N, may lead to points not converging at the surface.

4.2. Straighter flows via VelocityModule coupling

Given noisy initial data, the trajectories of points tend to be curved, with limited straightness. One way to improve straightness is to finetune the velocity network on the coupling $(\tilde{\boldsymbol{X}}_0, \tilde{\boldsymbol{X}}_1)$ to satisfy $\tilde{\boldsymbol{X}}_1 = V(\tilde{\boldsymbol{X}}_0)$. For filtering, we aim to recover surfaces while preserving local geometric details [35]. Applying such finetuning requires the precomputation of $\tilde{\boldsymbol{X}}_1$ for all possible surface patches and is infeasible due to the large number of patches in the training set. We propose a simple mechanism to straighten paths by coupling K VelocityModules together.

Training objective for coupled VMs. Given a noisy patch $\boldsymbol{X}_t$, we partition the trajectory from $\boldsymbol{X}_t$ to $\boldsymbol{X}_1$ into K segments and obtain the velocity flow $\boldsymbol{v}_\theta^k(\tilde{\boldsymbol{X}}_{t_k})$ at times $t_k = (t(K-k) + k)/K$ where $k \in \{0, 1, \dots, K-1\}$. Intermediate positions at times t_{k+1} are given by $\tilde{\boldsymbol{X}}_{t_{k+1}} = \tilde{\boldsymbol{X}}_{t_k} + (1/K) \, \boldsymbol{v}_\theta^k(\tilde{\boldsymbol{X}}_{t_k})$. We empirically find that two VelocityModules, i.e., $K = 2$, provide the best balance between accuracy and efficiency. The VelocityModules are

pretrained using the training objective of Eq. (7) and the coupled VMs are finetuned with the introduced objective,

$$\mathcal{L}_B = \mathbb{E}_{t \sim \mathcal{U}(0,1)} \left[\sum_{k=0}^{K-1} \left\| \boldsymbol{v}_\theta^k(\tilde{\boldsymbol{X}}_{t_k}) - \boldsymbol{\delta}(\boldsymbol{X}_1, \boldsymbol{X}_0) \right\|_2^2 \right.$$
$$\left. + \lambda_1 \sum_{k=0}^{K-2} \left\| \boldsymbol{\delta}(\tilde{\boldsymbol{X}}_{t_{k+1}}, \boldsymbol{X}_{t_{k+1}}) \right\|_2^2 \right], \quad (10)$$

for $K \geq 2$, $\lambda_1 = 10$ and $\tilde{\boldsymbol{X}}_{t_0} = \boldsymbol{X}_{t_0} = \boldsymbol{X}_t$. The first term of $\mathcal{L}_B$ encourages coupled VelocityModules to infer a constant velocity at times t_k, while the second encourages filtered points $\tilde{\boldsymbol{X}}_{t_k}$ to move closer to interpolated points $\boldsymbol{X}_{t_k}$.

Filtering objective for coupled VMs. At inference, we apply a modified form of the Euler method to filter points, similar to Eq. (8). We apply K coupled VelocityModules N times, resulting in $T = K \cdot N$ total filtering steps. We adjust our earlier notation of integer time steps such that $\hat{t} \in \{0, K, \dots \hat{M}, \dots, K(N-1)\}$ where $\hat{M}$ is divisible by K and $\tilde{\boldsymbol{X}}_0 = \boldsymbol{X}_0$. The sequential position update becomes,

$$\tilde{\boldsymbol{X}}_{(\hat{t}+k+1)/T} = \tilde{\boldsymbol{X}}_{(\hat{t}+k)/T} + \frac{1}{T} \boldsymbol{v}_\theta^k(\tilde{\boldsymbol{X}}_{(\hat{t}+k)/T}). \quad (11)$$

The corresponding full position update across time steps $\mathcal{S} = \{\hat{M}, \hat{M}+K, \dots, N(K-1)\}$ is,

$$\tilde{\boldsymbol{X}}_1 = \tilde{\boldsymbol{X}}_{\hat{M}/T} + \frac{1}{T} \sum_{\hat{t} \in \mathcal{S}} \sum_{k=0}^{K-1} \boldsymbol{v}_\theta^k(\tilde{\boldsymbol{X}}_{(\hat{t}+k)/T}). \quad (12)$$

4.3. Distance estimation to the clean surface

We now address the second challenge related to Eq. (9) and introduce the DistanceModule that scales the overall straight trajectory. This leads to better convergence near the surface. The DistanceModule estimates a distance scalar, corresponding to the standard deviation of initial noisy points from the clean surface. More specifically, the DistanceModule $\mathcal{D}_\phi(\cdot)$ is used to approximate a mapping $d_\phi : \mathbb{R}^{n \times 3} \to \mathbb{R}$, such that,

$$d_\phi(\tilde{\boldsymbol{X}}_{t_0}) = \text{Sigmoid}(\text{Max}(\mathcal{D}_\phi(\tilde{\boldsymbol{X}}_{t_0}))). \quad (13)$$

The distance d_ϕ is then used to scale the output of VelocityModules, as illustrated in Fig. 5. The DistanceModule is embedded within the StraightPCF architecture (see Fig. 2). We keep the weights of the finetuned coupled VelocityModules fixed when training the DistanceModule. The respective training objective for optimizing the parameters ϕ is,

$$\mathcal{L}_C = \mathbb{E}_{t \sim \mathcal{U}(0,1)} \left[\left(d_\phi(\tilde{\boldsymbol{X}}_{t_0}) - \frac{\|\boldsymbol{\delta}(\boldsymbol{X}_1, \boldsymbol{X}_{t_0})\|_2}{\|\boldsymbol{\delta}(\boldsymbol{X}_1, \boldsymbol{X}_0)\|_2} \right)^2 \right.$$
$$\left. + \lambda_2 \left\| \boldsymbol{\delta}(\tilde{\boldsymbol{X}}_1, \boldsymbol{X}_1) \right\|_2^2 \right], \quad (14)$$

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 [44]												
PCN [35]	3.515	1.148	7.467	3.965	13.067	8.737	1.049	0.346	1.447	0.608	2.289	1.285
PointFilter [45]	2.461	0.443	3.534	0.862	5.089	1.849	0.758	0.182	0.907	0.251	1.599	0.710
Score [28]	2.521	0.463	3.686	1.074	4.708	1.942	0.716	0.150	1.288	0.566	1.928	1.041
PDFlow [29]	2.126	0.381	3.246	1.010	4.447	1.999	0.651	0.164	1.173	0.581	1.914	1.210
DeepPSR [6]	2.353	0.306	3.350	0.734	4.075	1.242	0.649	0.076	0.997	0.296	1.344	**0.531**
IterativePFN [8]	2.056	**0.218**	3.043	**0.555**	4.241	1.376	0.605	**0.059**	0.803	**0.182**	1.971	1.012
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 [35]												
PCN [35]	3.847	1.221	8.752	3.043	14.525	5.873	1.293	0.289	1.913	0.505	3.249	1.076
Score [28]	3.369	0.830	5.132	1.195	6.776	1.941	1.066	0.177	1.659	0.354	2.494	0.657
PointFilter [45]	3.019	0.886	4.885	1.275	7.062	2.032	1.053	0.186	1.349	0.257	2.225	0.491
PDFlow [29]	3.243	0.606	4.545	0.966	5.934	1.441	0.969	0.152	1.646	0.424	2.450	0.569
DeepPSR [6]	2.873	0.783	4.757	1.118	6.031	1.619	1.010	0.146	1.515	0.340	2.093	0.573
IterativePFN [8]	**2.621**	0.698	4.439	1.011	6.026	1.560	0.913	**0.139**	1.251	**0.238**	2.529	0.716
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 recent state-of-the-art methods and our method on the synthetic PUNet and PCNet datasets. Note that our network is lightweight, with just $\sim 530K$ parameters (17% of IterativePFN). CD and P2M values are multiplied by 10^4.
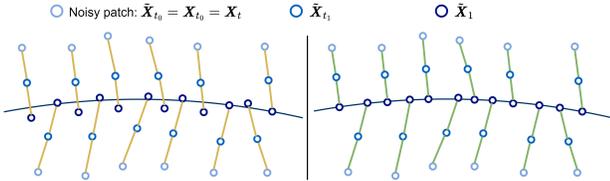
Figure 5. Left: Filtering by coupled VelocityModules only. Right: Coupled VelocityModules and DistanceModule. Scaled trajectories (green lines) lead to better convergence at the surface.

where $t_0 = t$ and $\tilde{\boldsymbol{X}}_{t_0} = \boldsymbol{X}_{t_0} = \boldsymbol{X}_t$. The first term of $\mathcal{L}_C$ encourages the DistanceModule to infer the relative distance of $\boldsymbol{X}_{t_0}$ from the clean surface, as compared to $\boldsymbol{X}_0$. The last term encourages points to return to the clean surface. We set $\lambda_2 = 2 \times 10^2$ to ensure the loss contribution of the last term is of the same order as that of the first one. Consequently, Eq. (11) becomes,

$$\tilde{\boldsymbol{X}}_{(\hat{t}+k+1)/T} = \tilde{\boldsymbol{X}}_{(\hat{t}+k)/T} + \frac{d_\phi(\tilde{\boldsymbol{X}}_{\hat{M}/T})}{T} \boldsymbol{v}_\theta^k(\tilde{\boldsymbol{X}}_{(\hat{t}+k)/T}). \tag{15}$$

5. Experiments

Next, we provide results on synthetic data under Gaussian noise and real-world Kinect [41] and Paris-Rue-Madame [38] data. The *supplementary document* contains additional results of both conventional and learning based methods on synthetic and real-world scanned data, as well as a comparison of testing times.

5.1. Training and evaluation details

We follow the training procedure of ScoreDenoise [28], and train our model on the PUNet dataset [44] consisting of 40 point clouds for training and 20 point clouds for testing. To ensure consistency with ScoreDenoise's training settings, we add Gaussian noise sampled with standard deviation $\sigma_H = 2\%$ of the bounding sphere radius to each clean point cloud. Our training procedure only requires the high noise variants and the clean ground truth targets. All intermediate states at noise scales $\sigma = (1 - t)\sigma_H$, with $t \sim \mathcal{U}(0, 1)$, are created as linear interpolations between these two, initial and final states, as per Eq. (4). For testing, we also consider 10 test point clouds from the PCNet dataset [35] provided by ScoreDenoise. These synthetic point clouds contain Gaussian noise at scales of 1%, 2% and 3% of the point cloud's bounding sphere radius. We use two different sampling densities of 10K and 50K points to evaluate filtering ability across different sparsity settings. Moreover, to assess filtering results on real-world noisy point clouds, we compare methods on the Kinect v1 dataset that comprises of 71 point clouds [41] and 4 scenes extracted from the Paris-Rue-Madame dataset [38]. Finally, we provide a comparison on 4 scenes of the Kitti-360 dataset [20], in the *supplementary document*. Following [8, 28], all methods are only trained on PUNet with Gaussian noise.

Implementation. We train and test StraightPCF on a NVIDIA GeForce RTX 3090 GPU using PyTorch. We use the Adam optimizer with a learning rate of 1×10^{-4}. Similar to [8, 28], we use PyTorch3D [36] to compute Chamfer Distance (CD) and Point2Mesh (P2M) metric values.
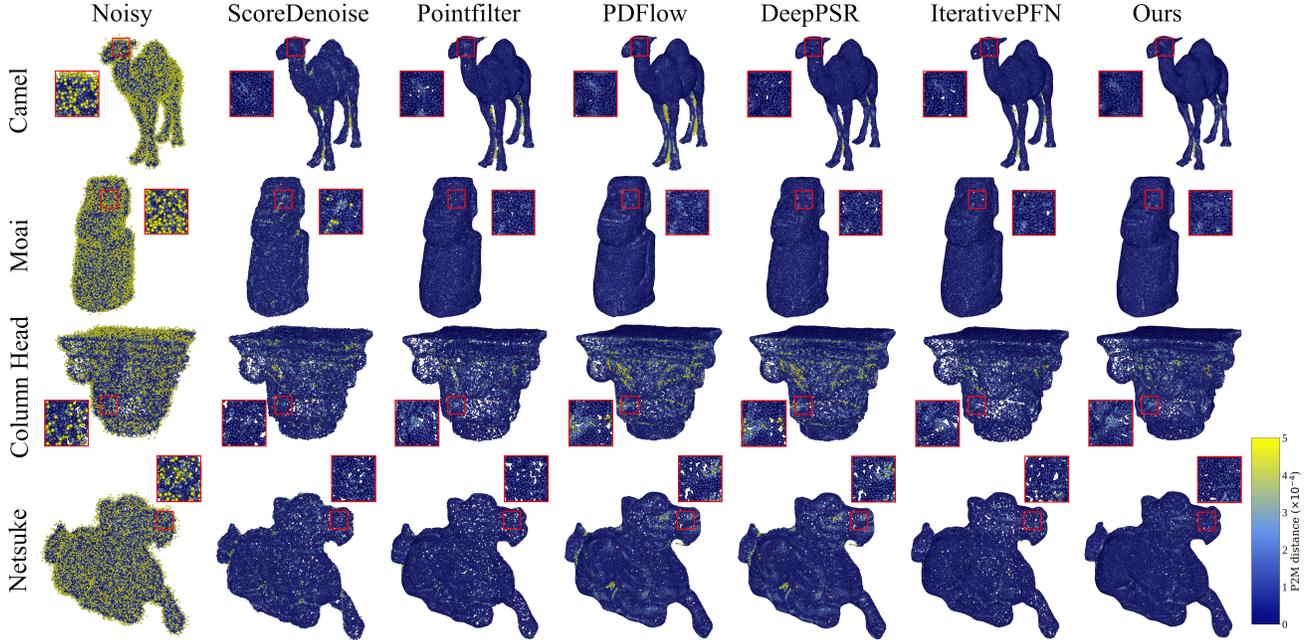
Figure 6. Visual filtering results for 50K resolution shapes ($\sigma = 2\%$) within the PUNet and PCNet datasets. The darker (i.e., more blueish) the better. We achieve both strong point-wise P2M results while also ensuring well distributed points (illustrated by close-ups) unlike DeepPSR or IterativePFN which have holes indicating clustering.

5.2. Performance on synthetic data

Table 1 and Fig. 6 illustrate filtering results on the synthetic PUNet and PCNet datasets. Our method has a clear edge over others in recovering the underlying clean point distribution demonstrated by superior CD results across all noise settings. We also achieve strong P2M results on the PUNet dataset. While IterativePFN [8] has marginally better P2M results on the PUNet data, this does not ensure a good distribution of points as evidenced by visual results in Fig. 6. Here, for noisy point clouds at 50K resolution and 2% noise, IterativePFN exhibits small holes, indicating clustering. Moreover, our network consists of only 530K parameters and is roughly 83% smaller than that of IterativePFN, given the latter's 3.2M parameters. DeepPSR [6], despite using a post-processing regularization step, also suffers from holes and poor point distributions. We also note that IterativePFN generalizes poorly to higher noise scales outside the training noise scales (where maximum noise is $\sigma = 2\%$). The ability of our method to generalize well across different data is demonstrated by the superior performance on the PCNet dataset. The PCNet shapes are unseen during training. Our method obtains both superior CD and P2M results over all other methods. At 10K resolution and $\sigma = 3\%$ noise, our method offers a 17.1% reduction in CD error and 24.1% reduction in P2M error. Our method also yields state-of-the-art results at 50K resolution where, for $\sigma = 3\%$, the CD error improvement is 13.2% and P2M error

improvement is 9.4%. Visual filtering results in Fig. 6 indicate that our method outperforms others in recovering complex details such as the legs of the Camel which are closely situated. By contrast, methods such as Pointfilter [45] and PDFlow [29] flatten points between these close surfaces while IterativePFN does not recover the body well. Furthermore, on a complex shape as Netsuke, we outperform other methods which either cause clustering (e.g., PDFlow, IterativePFN) at this high noise scale or leave behind noise due to limited filtering ability (e.g., ScoreDenoise).

5.3. Performance on real-world scanned data

Next we consider filtering results on scanned data. Table 2 provides quantitative results on the Kinect dataset and Fig. 7 illustrates visual results on the Paris-Rue-Madame dataset. Our method performs favorably on the Kinect data, obtaining a 1.82% reduction on CD errors, as compared to IterativePFN [8]. As this data is very sparse, our P2M result is marginally higher than Pointfilter [45] and Score-Denoise [28]. These methods focus on returning points to the surface yet succumb to clustering artifacts whereas our method both recovers surfaces while retaining good point distributions. The Paris-Rue-Madame dataset contains high noise scans due to outdoor environmental factors. In Fig. 7, we visualize filtering results for several most recent, state-of-the-art methods. In general, methods such as PDFlow [29] and DeepPSR [6] perform poorly in removing noisy artifacts while IterativePFN [8] causes points to
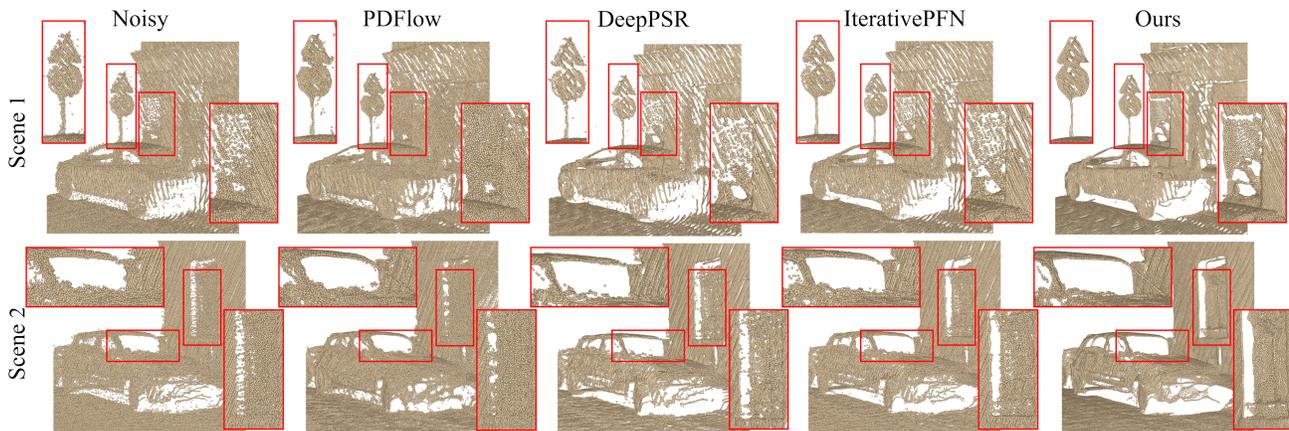
Figure 7. Visual filtering results on two scenes of the RueMadame dataset. We show results for the most recent state-of-the-art methods.

Method	Kinect	
	CD	P2M
ScoreDenoise	1.322	0.652
Pointfilter	1.377	**0.644**
PDFlow	1.334	0.699
DeepPSR	1.431	0.715
IterativePFN	1.320	0.685
Ours	**1.296**	0.664

Table 2. Quantitative results on Kinect data. Our network is lightweight, with just $\sim 530K$ parameters (17% of IterativePFN). CD and P2M values are multiplied by 10^4.

Ablation	10K points					
	1% noise		2% noise		3% noise	
	CD	P2M	CD	P2M	CD	P2M
V1) VM w/o DM	2.16	0.42	3.06	0.92	3.73	1.42
V2) VM w/ DM	2.00	0.32	3.06	0.96	3.81	1.55
V3) Large VM	2.17	0.41	2.99	0.84	3.54	1.29
V4) CVM w/o DM	1.97	0.32	3.01	0.92	3.71	1.45
V5) CVM w/ DM	**1.87**	**0.24**	**2.64**	**0.60**	**3.29**	**1.13**

Table 3. Ablation results for different VelocityModule (VM) and DistanceModule (DM) configurations.

cluster near the original scan lines. Our method, however, recovers the underlying clean surface and ensures a much better point distribution, as evidenced by the close-ups in Scene 1. In Scene 2, PDFlow and IterativePFN are not able to clean the surface of the parked vehicle whereas our method recovers the underlying shape with fewer noisy artifacts.

6. Ablation Study and Discussion

We train and test several variant architectures in order to ascertain the impact of the proposed VelocityModule (VM) and DistanceModule (DM). The results are given in Table 3. Our coupled VM + DM architecture (V5) increases the parameter number to $\sim 530K$. Therefore, we train another Large VM with $\sim 530K$ parameters (V3). We find that directly increasing the parameter number (V3) leads to very limited performance gain while V5 exhibits far superior performance. Furthermore, the coupled VM + DM architecture (V5) significantly outperforms the single VM (V1 and V2) architectures. Finally, there is a noticeable difference in performance with and without a DM as evidenced by the dis-

parity between V4 and V5. We provide further ablations, including higher VM couplings, in the *supplementary document*.

Limitation. While our StraightPCF yields state-of-the-art results across multiple datasets, we observe relatively low performance on low density or high sparsity data, which is similar to current methods. We provide the visual results in the *supplementary document* due to space limit.

7. Conclusion

Recent deep learning based filtering methods focus on moving noisy points along stochastic paths to remove noise from input point clouds. We propose the first study to consider filtering points along straight paths, leading to smaller discretization errors and fewer filtering iterations. This lightweight method, while being parameter efficient, delivers filtered point distributions closer to that of the ground truth distributions without requiring any regularization in loss function or post-processing. Our method achieves state-of-the-art performance on multiple synthetic and real-world datasets across standard filtering metrics, showcasing its superiority and effectiveness.

References

[1] A. Adamson and M. Alexa. Point-sampled cell complexes. *ACM Trans. Graph.*, 25:671–680, 2006. 2

[2] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and Cláudio T. Silva. Computing and rendering point set surfaces. *IEEE Trans. Vis. Comput. Graph.*, 9:3–15, 2003. 1, 2

[3] Ma Baorui, Han Zhizhong, Liu Yu-Shen, and Zwicker Matthias. Neural-pull: Learning signed distance functions from point clouds by learning to pull space onto surfaces. In *International Conference on Machine Learning (ICML)*, 2021. 3

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

[5] Honghua Chen, Zeyong Wei, Xianzhi Li, Yabin Xu, Mingqiang Wei, and Jun Wang. Repcd-net: Feature-aware recurrent point cloud denoising network. *International Journal of Computer Vision*, 130(3):615–629, 2022. 3

[6] 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. 2, 4, 6, 7

[7] Dasith de Silva Edirimuni, Xuequan Lu, Gang Li, and Antonio Robles-Kelly. Contrastive learning for joint normal estimation and point cloud filtering. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–15, 2023. 2, 3

[8] 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. 2, 3, 4, 6, 7

[9] Julie Digne. Similarity based filtering of point clouds. *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 73–79, 2012. 2

[10] Julie Digne and C. D. Franchis. The bilateral filter for point clouds. *Image Process. Line*, 7:278–287, 2017. 1, 2

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

[12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012. 1

[13] Gaël Guennebaud and M. Gross. Algebraic point set surfaces. In *SIGGRAPH 2007*, 2007. 1, 2

[14] 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

[15] Hui Huang, Dan Li, Hongxing Zhang, U. Ascher, and D. Cohen-Or. Consolidation of unorganized point clouds for surface reconstruction. *ACM SIGGRAPH Asia 2009 papers*, 2009. 1, 2

[16] Youngki Kim, Kiyoun Kwon, and Duhwan Mun. Mesh-offset-based method to generate a delta volume to support the maintenance of partially damaged parts through 3d printing. *Journal of Mechanical Science and Technology*, 35(7):3131–3143, 2021. 1

[17] D. Levin. The approximation power of moving least-squares. *Math. Comput.*, 67:1517–1531, 1998. 2

[18] Qing Li, Huifang Feng, Kanle Shi, Yue Gao, Yi Fang, Yu-Shen Liu, and Zhizhong Han. NeuralGF: Unsupervised point normal estimation by learning neural gradient function. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023. 3

[19] Qing Li, Huifang Feng, Kanle Shi, Yue Gao, Yi Fang, Yu-Shen Liu, and Zhizhong Han. SHS-Net: Learning signed hyper surfaces for oriented normal estimation of point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13591–13600, Los Alamitos, CA, USA, 2023. IEEE Computer Society. 3

[20] 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. 1, 6

[21] Y. Lipman, D. Cohen-Or, D. Levin, and H. Tal-Ezer. Parameterization-free projection for geometry reconstruction. *ACM SIGGRAPH 2007 papers*, 2007. 1, 2

[22] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. 3

[23] Dening Lu, Xuequan Lu, Yangxing Sun, and Jun Wang. Deep feature-preserving normal estimation for point cloud filtering. *Computer-Aided Design*, 125, 2020. 2

[24] Xuequan Lu, S. Schaefer, Jun Luo, Lizhuang Ma, and Y. He. Low rank matrix approximation for 3d geometry filtering. *IEEE transactions on visualization and computer graphics*, PP, 2020. 1, 2

[25] Chenxu Luo, Xiaodong Yang, and A. Yuille. Self-supervised pillar motion learning for autonomous driving. In *CVPR*, 2021. 1

[26] 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. 1, 2

[27] Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2837–2845, 2021. 3

[28] 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, 2, 3, 4, 6, 7

[29] 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. 3, 4, 6, 7

[30] E Mattei and A Castrodad. Point cloud denoising via moving rpca. *Computer Graphics Forum*, 36:123–137, 2017. 1, 2

[31] 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

[32] R. Preiner, O. Mattausch, Murat Arikan, R. Pajarola, and M. Wimmer. Continuous projection for fast l1 reconstruction. *ACM Transactions on Graphics (TOG)*, 33:1 – 13, 2014. 1, 2

[33] C. Qi, Hao Su, Kaichun Mo, and L. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017. 2

[34] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017. 2

[35] 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, 6

[36] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *ArXiv*, 2020. 6

[37] Riccardo Roveri, A. Cengiz Öztireli, Ioana Pandele, and Markus Gross. Pointpronets: Consolidation of point clouds with convolutional neural networks. *Computer Graphics Forum*, 37(2):87–99, 2018. 2

[38] Andrés Serna, Beatriz Marcotegui, François Goulette, and Jean-Emmanuel Deschaud. Paris-rue-madame database - a 3d mobile laser scanner dataset for benchmarking urban detection, segmentation and classification methods. In *ICPRAM*, 2014. 1, 6

[39] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2019. 3

[40] Philipp R. W. Urech, M. Dissegna, C. Girot, and A. Grêt-Regamey. Point cloud modeling as a bridge between landscape design and planning. *Landscape and Urban Planning*, 203:103903, 2020. 1

[41] Peng-Shuai Wang, Yang Liu, and Xin Tong. Mesh denoising via cascaded normal regression. *ACM Trans. Graph.*, 35(6): Article 232, 2016. 6

[42] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019. 2

[43] L Wu, D Wang, C Gong, X Liu, Y Xiong, R Ranjan, R Krishnamoorthi, V Chandra, and Q Liu. Fast point cloud generation with straight flows. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9445–9454. IEEE Computer Society, 2023. 3

[44] 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. 6

[45] 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. 2, 4, 6, 7