Deep Feature-preserving Normal Estimation for Point Cloud Filtering

Dening Lu\textsuperscript{a,1}, Xuequan Lu\textsuperscript{b,1}, Yangxing Sun\textsuperscript{a}, Jun Wang\textsuperscript{a}

\textsuperscript{a}Nanjing University of Aeronautics and Astronautics, P.R. China
\textsuperscript{b}Deakin University, Australia

Abstract
Point cloud filtering, the main bottleneck of which is removing noise (outliers) while preserving geometric features, is a fundamental problem in 3D field. The two-step schemes involving normal estimation and position update have been shown to produce promising results. Nevertheless, the current normal estimation methods including optimization ones and deep learning ones, often either have limited automation or cannot preserve sharp features. In this paper, we propose a novel feature-preserving normal estimation method for point cloud filtering with preserving geometric features. It is a learning method and thus achieves automatic prediction for normals. For training phase, we first generate patch based samples which are then fed to a classification network to classify feature and non-feature points. We finally train the samples of feature and non-feature points separately, to achieve decent results. Regarding testing, given a noisy point cloud, its normals can be automatically estimated. For further point cloud filtering, we iterate the above normal estimation and a current position update algorithm for a few times. Various experiments demonstrate that our method outperforms state-of-the-art normal estimation methods and point cloud filtering techniques, in terms of both quality and quantity.

Keywords: Normal estimation, Feature preserving, Point cloud filtering, Deep learning

1. Introduction
Point cloud filtering has attracted noticeable attentions recently. It has a wide range of applications, including further geometry processing, computer animation, rendering, computer aided design and so on. Point cloud filtering has remained a challenge to date, due to the main bottleneck of noise/outliers removal as well as the preservation of geometric features.

Existing point cloud filtering techniques can be generally categorized into position based and normal based methods. Position based methods \cite{1,2,3,4,5,6}, including learning techniques \cite{7,8,9,10}, are often not designed to preserve sharp features. Normal based approaches usually contain two common steps: normal estimation and position update. Some recent methods attempted to filter point cloud data by utilizing normal information, such as \cite{7,8,9,10}. Normals are usually estimated by either the $L_0/L_1/L_2$ nuclear minimization \cite{7,8,11,10}, or deep learning \cite{12}. Regarding position update based on estimated normals, the $L_0/L_1/L_2$ optimization have been often employed \cite{7,8,10}. However, the $L_0/L_1/nuclear$ minimization is usually complicated and slow due to their mathematical nature. The $L_2$ optimization such as bilateral normal smoothing \cite{11,13} may sometimes oversharpen geometric features. Moreover, the optimization methods usually involve multiple parameters, and require tedious tuning to achieve decent results. Learning methods indeed provide alternatives for automatic learning and prediction. Nevertheless, to our knowledge, none of the existing learning methods are introduced to estimate feature-preserving normals which are of particularly great importance to CAD-like models. Motivated by the above issues, in this paper we propose a novel feature-preserving normal estimation method for point cloud filtering. Our core idea is to estimate feature-preserving normals in a deep learning way, and then update point positions with the estimated normals. To achieve this, we propose a framework consisting of three steps for training. We first generate training data, and then classify points into feature points (i.e., anisotropic surface points) and non-feature points (i.e., isotropic surface points) via the classification network. The normal estimation network is then trained on feature points and non-feature points, respectively. In testing phase, we can automatically get normal estimations given an input point cloud. For filtering purpose, the estimated normals are then used to update point positions \cite{10}. We iterate the normal estimation and position update for a few times to obtain better filtering results. To our knowledge, it is the first method to achieve feature-preserving normal estimation via deep learning. Figure 1 shows an overview for the training and testing phases of the proposed approach.

Various experiments validate our method, and demonstrate that it outperforms the state-of-the-art normal estimation methods and point cloud filtering approaches both visually and quantitatively. The main contributions of this paper are:

- a novel point cloud filtering framework with preserving geometric features;
- a novel method for feature-preserving normal estimation;
- a novel method for the classification of feature and non-feature points.

2. Related Work

Normal Estimation. The early research on normal estimation was based on Principal Component Analysis (PCA) \cite{14}, which computes a tangent plane for each point according to its neighborhood. To improve the robustness and accuracy of PCA normals, several variants have been proposed \cite{15,16,17,18}. Other approaches based on Voronoi cells or Voronoi-PCA \cite{19,20,21,22} were also introduced to estimate normals. To preserve sharp features, \cite{7,8} were proposed to minimize the
Point Cloud Filtering. [27, 28] introduced Moving Least Squares (MLS) methods, which are designed for smooth surfaces. Such methods may blur sharp features of edges and corners. Owing to the ambiguity of the points which locate at the intersection of multi surfaces, recently a pair consistency voting strategy was presented by [26] to compute multiple normals for each feature point. By measuring how close each point is to sharp features, [26] screened the candidate points whose neighbors may be from several different surfaces. Then, a series of reliable tangent planes were fitted to describe the structure of the neighborhood. Conceivably, the normals of these tangent planes were used as the candidate point’s multiple normals.

In terms of point cloud filtering, EC-Net [4], PointProNet [5], PointcleanNet [6] and Pointfilter [41] were proposed recently. In detail, EC-Net designed a joint loss function to realize the edge-aware effects. It is suitable for dense point clouds whose scale of structure is invariant. Moreover, EC-Net [4] needed tedious manual efforts in sharp edge labeling. PointProNet [5] used a heightmap generation network to convert the unordered points to regularly height maps, and then employed the CNN architecture to consolidate the point cloud. PointcleanNet [6] proposed a two-stage data-driven denoising approach, which discarded outlier samples firstly and then estimated the local properties for the remaining points. Pointfilter [41] is position based and depends on prior normal information for training. However, nearly none of these methods are designed to estimate feature-preserving normals and further point cloud filtering.

3. Overview

Figure 1 shows an overview of our proposed method, which consists of the training and testing phases. Training includes data generation, classification and normal estimation. In testing phase, apart from the three steps above, an additional step (i.e.,
position update) is included to match the estimated normals and achieve the goal of point cloud filtering.

**Data generation.** Taking as input a noisy point cloud, we represent each point and its neighbors as a 2D height map, to meet the classical CNN requirement. Specifically, we present a simple projection approach based on PCA. Since our method consists of two networks (i.e., classification and normal estimation), we define two different labels for each point: feature/non-feature label and normal label.

**Classification.** To improve the accuracy of the normal estimation network, we first distinguish the feature and non-feature points, which is formulated as a regression problem. Instead of the commonly used $l_2$ loss, we introduce a weighted $l_1$ loss, leading to a better classification result for geometric features.

**Normal estimation.** Feature points are assumed to own multiple normals with weights, inspired by [25]. We train the normal estimation network for feature points and non-feature points, respectively, to achieve better normal estimations than the mixed training.

**Position update.** At test phase, we first predict the normals of the input point cloud, and then employ a position update algorithm to update point positions [10]. To achieve better results, the normal prediction and position update are alternately called for a few iterations.

4. Data Generation

Given a noisy point cloud $\mathbb{P} = \{p_i\}_{i=1}^N \subset \mathbb{R}^3$, we define each point $p$ ($3 \times 1$ vector) with its local neighborhood (spherical search) as a local patch $\chi$. Since the classical CNN requires 2D input, $\chi$ needs to be transformed to a 2D representation. Height map is an intuitive choice which has been proven to be effective for deep learning networks [5], and thus we use it in this work.

![Figure 2: Height map generation based on PCA. (a) The local patch for the point $p$. (b) The projection procedure. (c) The original height map. (d) The final height map after the Gaussian-weight interpolation.](image)

**Height map generation.** As shown in Figure 2, we present a simple yet effective projection scheme based on PCA, for the generation of height maps. Denote the eigenvalues of the position covariance matrix $C_p$ (Eq. (1)) for point $p$ by $\lambda_1, \lambda_2$ and $\lambda_3$ ($\lambda_1 \geq \lambda_2 \geq \lambda_3$), with the corresponding eigenvectors $\mu_1, \mu_2$ and $\mu_3$.

$$C_p = \frac{1}{|\chi|} \sum_{p \in \chi} (p - p_i)(p - p_i)^T,$$

where $|\chi|$ is the number of $p$’s neighboring points. We take the plane $P_{(p, \mu_3)}$ as the projection plane with a 2D regular grid (size $M = m \times m = 48 \times 48$ cells), where the $x$-axis and $y$-axis are $\mu_1$ and $\mu_2$, respectively. Given a point $p_i$ in $\chi$, we denote its corresponding cell in the the projection plane as $c_i$, and the coordinates $(x, y)$ and value $H(c_i)$ of $c_i$ are defined as:

$$c_i = (x, y) = \left(\frac{(p_i - p) \cdot \mu_1 + r_m}{2r}, \frac{(p_i - p) \cdot \mu_2 + r_m}{2r}\right).$$

Given the position update (Eq. (7) and Eq. (8)).

$$H(c_i) = (p_i - p) \cdot \mu_3,$$

where $r$ is the radius of the patch $\chi$, and is empirically set to 5 times the average distance $r_{\text{average}}$ of the input point cloud $\mathbb{P}$. This enables an adaptive radius based on the average distance for each input point cloud, as suggested by [5]. If a cell has more than one points, we simply choose the one with the shortest projection distance.

Since the number of the points in $\chi$ is generally less than $M$, we apply the Gaussian-weight interpolation to fill up the vacant cells (i.e., no points). Other types of interpolation may be possible and we found the Gaussian-weight interpolation works very well.

$$H_g(c_i) = \left\{ \begin{array}{ll} \frac{1}{\eta} \sum_{c_k \in \chi} g(c_i, c_k)H(c_k), & \text{if } ||c_i|| \neq 0 \\ 0, & \text{otherwise} \end{array} \right.$$  

where $w_i = \sum_{c_k \in \chi} g(c_i, c_k)$, and $g(c_i, c_k) = e^{-\frac{||c_i - c_k||^2}{\sigma^2}}$. $N_{c_i} = \{c_k ||c_i - c_k|| < \eta\}$, where $\eta$ means the radius of the Gaussian-weight interpolation. $\eta$ and $\sigma$ are set to $\frac{\xi}{2}$ and $\frac{\xi}{2}$, respectively.

![Figure 3: Multiple normals for a feature point. (a) The local patch for the feature point $p_f$. (b) The multiple normals of $p_f$, where different color arrows represent different normals. (c) The priority calculation for the multiple normals, where the plane $P_1$ is defined by $p_f$ and the blue normal $n_1$, and the plane $P_2$ is defined by $p_f$ and the green normal $n_2$. The smaller distance $d_1$ would lead to a greater contribution. (d) The dominant and secondary normal of $p_f$ based on the priority calculation (Eq. (7) and Eq. (8)).](image)

**Training labels.** We need to define the “label” for each point, for training purposes. To improve the accuracy of the normal estimation network, we first classify all points in $\mathbb{P}$ into feature points (anisotropic surface points) and non-feature points (isotropic surface points), and feed them into the normal estimation network for separate training. For labeling feature points and non-feature points, we define the feature point set $\mathbb{R}$ and non-feature point set $\mathbb{S}$ from $\mathbb{P}$ as:

$$\mathbb{R} = \left\{ p_i \in \mathbb{P} \mid \| p_i - p_g \| < r_f, \forall p_g \in \Psi \right\},$$

$$\mathbb{S} = \left\{ p_i \in \mathbb{P} \mid \| p_i - p_g \| \geq r_f, \forall p_g \in \Psi \right\},$$

where $\Psi$ is the ground-truth feature points set corresponding to $\mathbb{P}$, $p_g$ is a ground-truth feature point, and $r_f$ is the distance threshold which defaults to 2 times the average distance $r_{\text{average}}$. The ground-truth feature points ($\Psi$) are located easily and automatically. We generate the ground-truth point clouds and corresponding normals from the triangle meshes (Sec. 8). We employ two approaches to compute the normals of vertices (i.e., points), to identify feature points. One is based on Thurmer et al. [42], which computes the normal as a weighted sum of the incident face normals, and the other takes one of the incident face normals as the normal of the vertex directly, as shown in Figure 4. For the non-feature points (i.e., isotropic area), the two computed normals are very similar. However, the resulting normals are noticeably different for the feature points, such as edge or corner points, which is illustrated in Figure 4(b) and 4(c). As with [19], we regard the normals by the latter method as feature-preserving.
normals (Figure 4(c)), which is also to facilitate the position update (Sec. 7) in terms of point cloud filtering (Figure 5(c)). By contrast, the former method would blur the sharp features, as shown in Figure 5(b). As a result, we can easily detect and extract the ground-truth feature points by setting a threshold (18° by default) for the angle between the two normals calculated by the two above schemes. After this, we can simply extract the feature and non-feature points for classification training via Eq. (5) and Eq. (6).

![Figure 4: Ground-truth feature points extraction. (a) The triangle mesh model. (b) normals by L2. (c) Feature-preserving normals by the latter scheme. (d) The extraction of ground-truth feature points.](image)

Since our framework involves two networks (classification network and normal estimation network), we need to define labels for each of them. Regarding the classification network, we take vectors \( \gamma_1 = (1, 0) \) and \( \gamma_2 = (0, 1) \), to represent the labels of an extracted feature point and an extracted non-feature point, respectively. Note that the vector as the label accounts for the incidences of a point for being both a feature point and a non-feature point simultaneously, and works well in this work. Regarding the normal estimation network, we introduce the normal label for the non-feature points and feature points, respectively. For a non-feature point, we take the normal of its nearest point in ground truth as the normal label. The normals of feature points, such as edge points and corner points, are undefined or ambiguous, while the multiple normals [26] have been proposed for a clear definition. Inspired by the multi-normal concept [26], we define the multiple normals (default to two normals) for each feature point, as demonstrated in Figure 3. Given a feature point \( p_f \) and its defined local neighbors in \( P \), we select two corresponding ground-truth normals with the largest angle, as the multiple normals of \( p_f \) (denoted as \( n_1 \) and \( n_2 \)). These two normals also act as the normal label (6 × 1 vector) for this feature point.

In testing phase, we need to get a normal (3 × 1 vector) for a feature point, rather than a 6 × 1 vector. To do so, we first define the priorities of the two normals for training. Specifically, for the feature point \( p_f \) and its neighborhood \( \chi_f \), we define two planes \( P_1 = P_{(p_f, n_1)} \) and \( P_2 = P_{(p_f, n_2)} \) based on the multiple normals \( n_1 \) and \( n_2 \). Based on this, we can calculate the priorities \( \omega_1 \) and \( \omega_2 \) of \( n_1 \) and \( n_2 \) as:

\[
\omega_1 = \sum_{p \in \chi_f} \frac{d(p, P_1)}{\sigma_{\gamma_1}} + \frac{d(p, P_2)}{\sigma_{\gamma_2}},
\]

(7)

\[
\omega_2 = \sum_{p \in \chi_f} \frac{d(p, P_2)}{\sigma_{\gamma_1}} + \frac{d(p, P_1)}{\sigma_{\gamma_2}},
\]

(8)

where \( d(p, P) \) represents the distance between point \( p \) and plane \( P \), and \( \sigma_T \) is a scaling parameter, which is empirically set to 2 times \( r_{\text{average}} \).

![Figure 5: (a) Input (1.5% noise). (b) normals by L2 and filtering result. (c) feature-preserving normals by the latter method and filtering result.](image)

![Figure 6: The balance points in the feature point set, whose multiple normals have similar weights. (d) shows the comparison of normal estimation network training with and without balance points.](image)

As described in Eq. (7) and (8), the priorities of the multiple normals are based on the contributions of all points in \( \chi_f \), and that a smaller distance would lead to a greater contribution. Therefore, the normal with a larger priority is defined as the dominant normal, while the other is chosen as the secondary normal. In testing, the dominant normal is simply selected as the final prediction. For corners, we still randomly choose two normals with the largest priorities as the multiple normals, even though the corner point may have more than a pair of normals with the same largest angle. Notice that there are some balance points in the feature point set, whose multiple normals have similar weights, as shown in Figure 6. Since the normal priority of a balance point is indistinguishable, it may hinder the training of the normal estimation network, as illustrated in Figure 6(d). To solve this issue, we simply remove such points from the feature point set.

5. Classification

We notice that the proportion of feature points is often small, especially for CAD-like models. As a result, the direct training of the normal estimation network based on all points would result in limited accuracy for feature points, as shown in Figure 7.

![Figure 7: Normal estimation with or without the classification step.](image)
Therefore, to improve the accuracy, we first classify the feature and non-feature points by training the classification network (this section), and then respectively train the normal estimation network for feature points and non-feature points (Sec. 6).

LeNet [43], which has been proven to be adaptable to various regression problems in image processing, is chosen for this classification task. It consists of three convolutional layers, two poolings and two fully connected layers. In addition, we resize our height maps (48 × 48) to the input size (32 × 32) of LeNet [43]. We introduce the following weighted $l_2$ loss to train this network:

$$L_c = \sum_{p_i \in \mathcal{T}} g(\theta_i) \cdot ||\gamma_i - \hat{\gamma}_i||^2,$$

where $\Gamma$ includes all points in the training set for classification, $\gamma_i$ and $\hat{\gamma}_i$ are the output of the classification network and the classification label for point $p_i$, respectively. $g(\theta_i) = e^{t - (\frac{\theta_i}{\sigma_\theta})^2}$ and produces greater weights to feature points for better recognition. $\sigma_\theta$ is empirically set to 30°, and $\theta_i$ is defined as:

$$\theta_i = \begin{cases} 
\arccos (\frac{n_{i1} \cdot n_{i2}}{|n_{i1}| |n_{i2}|}), & \text{if} \ p_i \in \mathcal{R} \\
0, & \text{if} \ p_i \in \mathcal{F}
\end{cases} \quad (10)$$

where $n_{i1}$ and $n_{i2}$ are the multiple normals of $p_i$. $L_c$ is differentiable, so the gradients can be back-propagated through the network. In testing time, by setting a threshold $\sigma_\tau$ (default to 0.85), the feature and non-feature point set $\mathcal{R}_f$ and $\mathcal{R}_{non-f}$ of the test model $\mathcal{R}$ can be naturally obtained:

$$\mathcal{R}_f = \{p_i \in \mathcal{R} | \sigma_i > \sigma_f\}, \quad (11)$$

$$\mathcal{R}_{non-f} = \{p_i \in \mathcal{R} | \sigma_i \leq \sigma_f\}, \quad (12)$$

where $\sigma_i$ is the classification score, defined as a proportion style:

$$\sigma_i = \frac{a_i}{a_i + b_i}, \quad (13)$$

where $(a_i, b_i)$ is the output of the classification network for $p_i$. $\sigma_i$ accounts for the ratio between $a_i$ and the total $(a_i + b_i)$, and a greater $\sigma_i$ means a greater probability of being a feature point.

We choose a weighted $l_2$ loss, rather than the commonly used $l_2$ loss, because the latter is inferior to the former in recognizing feature points and non-feature points. Figure 8 shows the comparisons of the two losses.

Figure 8: Classification results with two different losses, the commonly used $l_2$ loss and our weighted $l_2$ loss. Feature points are colored in red. Classification accuracy (%): (b) 88.5, (c) 96.6, (d) 84.8 and (e) 92.7.

6. Normal Estimation

Network architecture. Our normal estimation network takes ResNet18 [44] as the backbone. It takes a height map as input, and outputs a 6 × 1 vector or 3 × 1 vector for a feature point and a non-feature point, respectively. For training, the $l_2$ penalization is adopted as the loss:

$$L_n = \sum_{p_i \in \mathcal{Q}} ||n_{p_i} - \hat{n}_{p_i}||^2, \quad (14)$$

where $\mathcal{Q}$ contains feature points or non-feature points in the training set for normal estimation. $n_{p_i}$ and $\hat{n}_{p_i}$ are the estimated normal and the normal label for point $p_i$, respectively. It should be noted that the normal label needs to be transformed to the eigen space of the corresponding local patch in the noisy point cloud before training. It is achieved by multiplying the normal vector by the eigen matrix (formed by the three eigen vectors of $\sigma_i$). We do this since we found the original normal label would lead to strong oscillation and even non-convergence during training. For testing, the estimated normal can be easily computed with a corresponding inverse matrix to the eigen matrix.

Perturbation strategy. The perturbation strategy is designed for the testing phase only. After classification, we can obtain the feature point set of the testing point cloud. However, the height maps of the feature points are not expected to be directly fed into the normal estimation network, due to the balance points (Sec. 4). Because it is impossible to calculate the priorities (Eq. 7 and 8) to judge the balance points, we simply introduce a tiny perturbation to each feature point $p_f$, to push it a bit away from the equilibrium status, as demonstrated in Figure 9. The feature point after perturbation ($\hat{p}_f$) can be defined as:

$$\hat{p}_f = p_f + \epsilon = \bar{\omega} \cdot r_{average}. \quad (15)$$

where $\nu$ represents the perturbation direction for $p_f$, and is defined as the orientation pointing to the neighboring point with the smallest classification score $\bar{\omega}$. $\epsilon = \bar{\omega} \cdot r_{average}$. Based on $\hat{p}_f$,
we re-generate the height map, and feed it to the normal estimation network. Note that $\tilde{p}_f$ is only used to re-generate a height map and $p_f$ is thus not changed. To further demonstrate the performance of the perturbation strategy, we estimate normals with $\tilde{p}_f$ (Figure 9(b)) and without perturbation (Figure 9(c)). Figure 9(d) and 9(e) show the estimation results, respectively. It is obvious that the latter achieves a better result than the former both visually and quantitatively, since there are balance points in edge and corner parts in Figure 9(b).

We do not perform the perturbation strategy for the training phase, since we found the training data already involves rich similar samples to the samples perturbed on balance points. As a result, the decent normal estimations can be achieved.

7. Position Update

In testing phase, we adopt an efficient point update algorithm [10] to match the estimated normals output by Sec. 6. Assuming that the point $p_i$ and its neighboring information $N_i$ are known and unchanged [10], its new position can be calculated as:

$$\hat{p}_i = p_i + \alpha_i \sum_{p_k \in N_i} (p_k - p_i) (n_k^T n_k + n_i^T n_i),$$ (16)

where $\alpha_i$ is the step size that is default to $\frac{1}{|N_i|}$, as suggested by the authors [10].

Figure 10: The classification results of the Octahedron point set (1.5% noise) in different iterations.

Figure 11: The normal estimation results in different iterations. MSAE: mean square angular error [10]. PGP: the proportion of good points [13]. Visual results describe the angle errors.

To achieve better results for the testing phase, we alternately call the normal estimation and position update for a few iterations. The number of iterations $\kappa$ is empirically set to 6, to mitigate the gap near sharp edges which has been pointed out in the original work [10]. As shown in Figure 10, 11 and 12 as the number of iterations increases, the results of classification and normal estimation are getting better, which lead to better filtering results. This is because the classification, normal estimation and position update work collaboratively. In other words, classification facilitates normal estimation and normal estimation facilitates position update, and position update further facilitates classification in next iteration. Therefore, the noise level of the input point cloud decreases with increasing iterations, and better results can be achieved.

8. Results

We test our approach on a variety of raw and synthetic point clouds, and compare it with the state-of-the-art normal estimation approaches, including principal component analysis (PCA) [14], bilateral normal filtering (BF) [11], Hough-based method (Hough) [38] and Nesti-Net [40]. Besides, we also compare our method with current point cloud filtering techniques, including EC-Net [4], PointCleanNet [6], CLOP [3], RIMLS [13] and GPF [9].

For the sake of fair comparisons, the parameters of each method are tuned carefully, based on the suggested ranges or values. Similar to [9], we also upsample the denoised point cloud of the same input to a similar number of points for visualization purposes, and set the same parameter values for each method when reconstructing surfaces based on the upampling results of the same model.
8.1. Dataset and Parameter Setting

**Dataset.** Our collected dataset consists of 31 models which have been split into 18 training models and 13 test models, shown in Figure 13. The training set contains 11 CAD-like models and 7 smooth models which are all with synthetic noise. For testing, we have 7 CAD-like models and 6 smooth models with different noise levels. We also test the real-world data captured by Kinect-v2 and Handy700 (included in the test set), to demonstrate the generalization capability of our method. We generate the ground truth point clouds and corresponding normals from the triangle meshes via randomly sampling $10K - 30K$ points. Each point cloud is corrupted with different levels of noise (Gaussian standard deviation 0.5%, 1%, 1.5%, 2% of the bounding box diagonal length). We also augment the training data by applying a random rotation to each point cloud and swapping the axles of the eigenspace of each patch. During training, the same number (2K in our experiments) of patches of each model are extracted randomly in an epoch, to ensure that the training can cover all types of models evenly. We train 100 and 300 epoches for the classification and normal estimation networks, respectively.

**Network implementation and parameters.** We implemented the networks with Pytorch and trained them on a NVIDIA Geforce GTX 1080 GPU. The classification and normal estimation networks are both trained with the Adam Optimizer, with a momentum of 0.9 and starting with an initial learning rate of 0.001 decreased by a factor of 0.9 at every 100 epoches. All the involved parameters of our method are empirically set (refer to Sec. 4-7), to show the robustness of our approach.

8.2. Normal Estimation

In general, PCA [14] estimates the normal of a point by fitting a local plane to its neighbors. Thus, the estimated normals tend to smooth sharp features. Bilateral normal smoothing [11] is suitable for CAD-like models, but prone to over-sharpen some regions to reach the overall smoothing effect. The Hough-based method [38] has challenges in predicting normals of sharp feature points, since it does not consider the distinctions of feature and non-feature points. Given fisher vectors as input, the Nesti-Net [40] relies on a data driven approach for selecting the optimal scale around each point and encourages sub-network specialization. It works relatively poorly in a single scale. Compared with these methods, our normal estimation approach treats feature and non-feature points separately, and achieves the best results (i.e., lowest errors), as shown in Table 1.

8.3. Point Cloud Filtering

Besides the above normal estimation, we also demonstrate the proposed approach in the point cloud filtering application.

**Synthetic point clouds.** We apply the position update algorithm [10] to match the estimated normals output by the above normal estimation methods. Generally, a better normal estimation result will lead to a more accurate denoised model. Figures 17-18 and 20 show that our normal estimation method can enable better point cloud filtering results than state-of-the-art normal estimation approaches. Apart from the filtering comparisons with the normal estimation methods, we also compare our method with the state-of-the-art denoising methods, including EC-Net [4], PointCleanNet [6], CLOP [3], RIMLS [13] and GPF [9].

In Figure 19, RIMLS tends to smooth out some small-scale features. GPF generates an over-sharpened result near the cylindrical features, and leads to an uneven distribution in the de-
Figure 17: Filtering results on a noisy mechanical point cloud (1% noise). First row: filtering results with upsampling. Second row: corresponding surface reconstruction results. Third row: corresponding distance errors. The root mean square errors are (×10^{-3}): (b) 3.7, (c) 2.4, (d) 3.1, (e) 2.7 and (f) 1.9.

Table 1: Normal errors (mean square angular error, in radians) of the point clouds in Figure 16.

| Models      | Figure 16 1st row | Figure 16 2nd row | Figure 16 3rd row | Figure 16 4th row | Figure 16 5th row | Figure 16 6th row | Figure 16 7th row |
|-------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| PCA [14]    | 0.051             | 0.086             | 0.097             | 0.017             | 0.091             | 0.040             | 0.052             |
| BF [11]     | 0.006             | 0.024             | 0.022             | 0.006             | 0.009             | 0.007             | 0.018             |
| Hough [38]  | 0.009             | 0.046             | 0.043             | 0.007             | 0.013             | 0.033             | 0.026             |
| Nesti-Net [40] | 0.004         | 0.040             | 0.015             | 0.005             | 0.007             | 0.011             | 0.020             |
| Ours        | **0.002**         | **0.007**         | **0.011**         | **0.002**         | **0.005**         | **0.005**         | **0.009**         |

Figure 18: Filtering results on a noisy Leg point cloud (1% noise). First row: noisy input and filtering results with upsampling. Second row: distance errors. The root mean square errors are (×10^{-3}): (b) 6.9, (c) 4.5, (d) 2.5, (e) 1.9, (f) 1.3.

Figure 19: Filtering results on a noisy Trim-star point set (1% noise). First row: filtering results with upsampling. Second row: surface reconstruction results.

Figure 20: Filtering results on a noisy Monkey point cloud (0.5% noise). First row: filtering results with upsampling. Second row: distance errors. The root mean square errors are (×10^{-3}): (b) 2.0, (c) 4.1, (d) 1.9, (e) 1.8, (f) 1.5.

noised model due to the edge absorption. Since the EC-Net and PointCleanNet are independent of normals, we use the bilateral normal smoothing [11] before their reconstruction, by setting the same parameters. This actually enhances their reconstruction results. Figure 20 shows the comparisons of the deep learning methods on a complicated Monkey point cloud, and Figure 21 shows the results of different point cloud filtering techniques over six different models, which also confirms the outstanding
Figure 21: Additional comparison of filtering results on seven point cloud models: Octahedron point set (1.5% noise), raw Girl point set captured by Kinect, Leg point set (1% noise), Block point set (1.5% noise), Julius point set (1% noise) and Joint point set (1% noise). From left to right: noisy input and the filtering results of EC-Net [4], PointCleanNet [6], CLOP [3], RIMLS [13], GPF [9] and Ours.
performance of our method. Since we classify points to features and non-feature points and handle them separately, our method achieves the best results in the preservation of both straight and curved sharp edges among all methods.

**Raw point scans.** Besides the synthetic models, we further contrast our approach with these methods on the scanned data with raw noise. Figure 22, 23 and 24 show the filtering results by updating positions based on normal estimation and the state-of-the-art point set filtering methods. We can see that the results by our approach have higher fidelity to the ground truth than other methods, with regard to preserving features.

### 8.4. Quantitative Results

To quantitatively evaluate the normal estimation and denoising quality, we utilize several published metrics to measure the results output by the involved methods, against the corresponding ground truth. We measure the accuracy for classification, which is described as the proportion of the true feature points in the predicted feature point set. Figure 8, 9, 10 and 14 show the classification results both visually and quantitatively. It demonstrates that classification is useful and necessary to the normal estimation, and a higher accuracy would typically lead to a better normal estimation result. For normal estimation, we evaluate the normal errors by adopting the MSAE (mean square angular error) [10], as listed in Table 1. Figure 15 visualizes the normal errors for some test models, and our normal estimation result is better than those by the state-of-the-art techniques. This is also confirmed by the comprehensive normal estimation results shown in Figure 16. For point cloud filtering, we calculate the mean distances between points of the ground truth and their closest points of the upsampled point cloud [9], and compute the RMSE metric of all mean distances in a point cloud to measure the geometric fidelity. In addition, we also calculate the Chamfer distance [6] in Table 2. Figures 17, 18, 20, 23 and 24 include a visualization of the distance errors, respectively. The distance errors show
Table 2: Chamfer measure ($\times 10^{-5}$) of the point clouds in Figure 21.

| Models         | Figure 21 1st row | Figure 21 2nd row | Figure 21 3rd row | Figure 21 4th row | Figure 21 5th row | Figure 21 6th row |
|----------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| EC-Net[4]      | 4.59              | 10.72             | 1.25              | 12.41             | 9.73              | 1.68              |
| PointCleanNet[6] | 5.38             | 12.20             | 1.54              | 14.73             | 8.41              | 1.84              |
| CLOP[3]        | 5.75              | 14.15             | 2.37              | 16.92             | 13.11             | 2.26              |
| RIMLS[13]      | 2.16              | 13.30             | 1.47              | 5.94              | 11.37             | 1.44              |
| GPF[9]         | 1.73              | 16.88             | 1.10              | 4.73              | 11.82             | 1.24              |
| Ours           | 1.14              | 6.33              | 0.59              | 3.37              | 5.81              | 0.77              |

Figure 25: Filtering results of our method and PointCleanNet[6] over the Block point clouds with different densities. From the first row to the third row: 10K, 50K and 100K points, respectively.

Figure 26: Distance errors of our method and PointCleanNet[6] over the Block point clouds with different densities.

Figure 27: Filtering results of our method over the Cube point clouds with different levels of outliers. First row: the inputs with 3%, 6% and 9% outliers. Second row: the filtering results with upsampling.

that our normal estimation method can produce obviously better filtering results than those by state-of-the-art normal estimation and point cloud filtering methods.

8.5. Robustness to Density Variation

Both our train set and test set contains noisy data with different densities. We note that some methods like PointCleanNet[6] are designed to process dense point clouds, which would usually produce less pleasant results on the sparse-density data. In comparison, our method employs an adaptive radius and Gaussian-weight interpolation for the height maps generation, and the train set comprises of point clouds with different densities. As such, it is able to flexibly deal with different densities. As shown in Figures 25 and 26, taking the block as an example, we provide the visual and quantitative comparisons on varied densities for PointCleanNet[6] and our method. We can see that both methods can obtain better results for the dense point clouds (50K/100K points), compared to the less dense data (10K points). Our method generates consistently better results than PointCleanNet[6] for all densities, which manifests that our method is more robust to density variation.

8.6. Limitations

Despite the above advantages of our method, it still has a few limitations. Similar to existing works[9, 10], our approach has difficulty in handling exceptionally severe noise. This is because such noise would significantly destroy the underlying surfaces, and lead to many ambiguities for feature and non-feature points. Second, our method is not particularly designed to deal with significant outliers. Figure 27 shows that our approach is capable of handling the 3% and 6% outliers, and generally produce less desired results in the presence of heavier outliers (9%). Our method and PointCleanNet[6] might complement each other, in terms of features preservation and outliers removal. In the future, we would like to investigate and solve the above issues by introducing innovative techniques. Points and height maps are two different representations which satisfy the PointNet form and 2D CNNs, respectively. We would also like to exploit the two representations further in the future.

9. Conclusion

In this paper, we have proposed a novel method for feature-preservation normal estimation and further point cloud filtering. Taking a noisy point cloud as input, our approach can automatically estimate its normals. Point positions are then updated to match the estimated normals. To achieve decent filtering results, we iterate the normal estimation and position update for
References

[1] Y. Lipman, D. Cohen-Or, D. Levin, H. Tal-Ezer, Parameterization-free projection for geometry reconstruction, in: ACM Transactions on Graphics (TOG), Vol. 26, ACM, 2007, p. 22.

[2] H. Huang, D. Li, H. Zhang, U. Ascher, D. Cohen-Or, Consolidation of unorganized point clouds for surface reconstruction, ACM Transactions on graphics (TOG) 28 (5) (2009) 176.

[3] R. Preiner, O. Mattausch, M. Arikian, R. Pajarola, M. Wimmer, Continuous projection for fast 11 reconstruction., ACM Trans. Graph. 33 (4) (2014) 47–1.

[4] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, P.-A. Heng, Ec-net: an edge-aware point set consolidation network, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 386–402.

[5] R. Roveri, A. C. Otzirel, J. Pandele, M. Gross, Pointprontets: Consolidation of point clouds with convolutional neural networks, in: Computer Graphics Forum, Vol. 37, Wiley Online Library, 2018, pp. 87–99.

[6] M.-J. Rakotosaona, V. La Barbera, P. Guerrero, N. J. Mitra, M. Ovsjanikov, Pointcleannet: Learning to denoise and remove outliers from dense point clouds, in: Computer Graphics Forum, Wiley Online Library, 2019.

[7] X. Chen, H. Ma, J. Yang, X. D. H. Zhang, Y. He, Low rank matrix approximation to retain quality of computation 67 (224) (1998) 1517–1531.

[8] S. Wu, H. Huang, M. Gong, M. Zwicker, D. Cohen-Or, Deep points consolidation, ACM Transactions on Graphics (ToG) 34 (6) (2015) 176.

[9] J. Digne, Similarity based filtering of point clouds, in: 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, IEEE, 2012, pp. 73–79.

[10] O. Remil, Q. Xie, X. Xie, K. Xu, J. Wang, Data-driven sparse priors of 3d shapes, in: Computer Graphics Forum, Vol. 36, Wiley Online Library, 2017, pp. 63–72.

[11] D. Levin, The approximation power of moving least-squares, Mathematics of computation 67 (224) (1998) 1517–1531.

[12] X. Chen, H. Ma, J. Wai, B. Li, T. Xia, Multi-view 3d object detection network for autonomous driving, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1907–1915.

[13] Y. Lipman, D. Cohen-Or, D. Levin, H. Tal-Ezer, Parameterization-free projection for geometry reconstruction, in: ACM Transactions on Graphics (TOG), 2007, p. 22.

[14] H. Huang, D. Li, H. Zhang, U. Ascher, D. Cohen-Or, Consolidation of unorganized point clouds for surface reconstruction, ACM Transactions on graphics (TOG) 28 (5) (2009) 176.

[15] R. Preiner, O. Mattausch, M. Arikian, R. Pajarola, M. Wimmer, Continuous projection for fast 11 reconstruction., ACM Trans. Graph. 33 (4) (2014) 47–1.

[16] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, P.-A. Heng, Ec-net: an edge-aware point set consolidation network, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 386–402.

[17] R. Roveri, A. C. Otzirel, J. Pandele, M. Gross, Pointprontets: Consolidation of point clouds with convolutional neural networks, in: Computer Graphics Forum, Vol. 37, Wiley Online Library, 2018, pp. 87–99.

[18] M.-J. Rakotosaona, V. La Barbera, P. Guerrero, N. J. Mitra, M. Ovsjanikov, Pointcleannet: Learning to denoise and remove outliers from dense point clouds, in: Computer Graphics Forum, Wiley Online Library, 2019.

[19] X. Sun, S. Schaefer, W. Wang, Denoising point sets via l0 minimization, Computer Aided Geometric Design 35 (2015) 2–15.

[20] H. Avron, A. Sharf, C. Greif, D. Cohen-Or, 11-sparse reconstruction of sharp point set surfaces, ACM Transactions on Graphics (TOG) 29 (5) (2010) 135.

[21] X. Lu, S. Wu, H. Chen, S.-K. Yeung, W. Chen, M. Zwicker, Ggf: Gmm-inspired feature-preserving point set filtering, IEEE transactions on visualization and computer graphics 20 (6) (2014) 2315–2326.

[22] X. Lu, S. Schaefer, J. Luo, L. Ma, Y. He, Low rank matrix approximation for geometry filtering, arXiv preprint arXiv:1803.06783.

[23] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, H. R. Zhang, Edge-aware point set resampling, ACM transactions on graphics (TOG) 32 (1) (2013) 9.

[24] P. Guerrero, Y. Kleinman, M. Ovsjanikov, N. J. Mitra, Pcpnet learning local shape properties from raw point clouds, in: Computer Graphics Forum, Vol. 37, Wiley Online Library, 2018, pp. 75–85.

[25] A. C. Otzirel, G. Guennebaut, M. Gross, Feature preserving point set surfaces based on non-linear kernel regression, in: Computer Graphics Forum, Vol. 29, Wiley Online Library, 2009, pp. 493–501.

[26] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, Surface reconstruction from unorganized points, Vol. 26, ACM, 2007, p. 23.

[27] K. Klausing, D. Altho , H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, Surface reconstruction from unorganized points, Vol. 26, ACM, 2007, p. 22.

[28] N. J. Mitra, A. Nguyen, Estimating surface normals in noisy point cloud data, in: Proceedings of the nineteenth annual symposium on Computational geometry, ACM, 2003, pp. 322–328.

[29] M. Pauly, M. Gross, L. P. Kobbelt, Efficient simplification of point-sampled surfaces, in: Proceedings of the conference on Visualization’02, IEEE Computer Society, 2002, pp. 163–170.

[30] M. Yoon, Y. Lee, S. Lee, I. Ivriissimtzis, H.-P. Seidel, Surface and normal ensembles for surface reconstruction, Computer-Aided Design 39 (5) (2007) 408–420.

[31] N. Amenta, M. Bern, Surface reconstruction by voronoi filtering, Discrete & Computational Geometry 22 (4) (1999) 481–504.

[32] Q. Mériot, M. Ovsjanikov, L. J. Guibas, Voronoi-based curvature and feature estimation from point clouds, IEEE Transactions on Visualization and Computer Graphics 17 (6) (2010) 743–756.

[33] T. K. Dey, S. Goswami, Provable surface reconstruction from noisy samples, Computational Geometry 35 (1-2) (2006) 124–141.

[34] P. Alliez, D. Cohen-Steiner, Y. Tong, M. Desbrun, Voronoi-based variational reconstruction of unoriented point sets, in: Symposium on Geometry processing, vol. 7, 2007, pp. 39–48.

[35] B. Li, R. Schnabel, R. Klein, Z. Cheng, G. Dang, S. Jin, Robust normal estimation for point clouds with sharp features, Computers & Graphics 34 (2) (2010) 94–106.