Refine-Net: Normal Refinement Neural Network for Noisy Point Clouds

Haoran Zhou®, Honghua Chen®, Yingkui Zhang®, Mingqiang Wei®, Senior Member, IEEE, Haoran Xie®, Senior Member, IEEE, Jun Wang, Tong Lu®, JIng Qin®, and Xiao-Ping Zhang®, Fellow, IEEE

Abstract—Point normal, as an intrinsic geometric property of 3D objects, not only serves conventional geometric tasks such as surface consolidation and reconstruction, but also facilitates cutting-edge learning-based techniques for shape analysis and generation. In this paper, we propose a normal refinement network, called Refine-Net, to predict accurate normals for noisy point clouds. Traditional normal estimation wisdom heavily depends on priors such as surface shapes or noise distributions, while learning-based solutions settle for single types of hand-crafted features. Differently, our network is designed to refine the initial normal of each point by extracting additional information from multiple feature representations. To this end, several feature modules are developed and incorporated into Refine-Net by a novel connection module. Besides the overall network architecture of Refine-Net, we propose a new multi-scale fitting patch selection scheme for the initial normal estimation, by absorbing geometry domain knowledge. Also, Refine-Net is a generic normal estimation framework: 1) point normals obtained from other methods can be further refined, and 2) any feature module related to the surface geometric structures can be potentially integrated into the framework. Qualitative and quantitative evaluations demonstrate the clear superiority of Refine-Net over the state-of-the-arts on both synthetic and real-scanned datasets.

Index Terms—Refine-net, normal estimation, noisy point clouds, multi-scale and -feature modules, point cloud denoising

1 INTRODUCTION

Point cloud is a set of unstructured points to efficiently represent 3D surfaces/scenes. Each point in the point cloud possesses a spatial position (x, y, z) and potentially a vector of attributes, such as the normal, color or material reflection.
designed regularities (e.g., $l_0$ minimization, low rank) or noise distributions (e.g., Gaussian/impulsive noise) to preserve geometric features. These prior-based approaches, though achieving notable successes in certain surface types or noise models, may not generalize well to resolve diverse inputs. In addition, they rely on parameters tuning heavily, like the neighborhood scale for plane fitting [18], [19], which tends to over-smooth or over-sharpen normal estimation results.

Recent years have witnessed considerable research efforts of learning-based normal estimation. Existing wisdom, either using the convolutional neural network (CNN) architecture [20], [21], or the PointNet architecture [22], attempts to design a complete form of feature representation to predict point normals. However, two core problems are adverse to accurate normal estimation. First, while many methods [20], [21], [22], [23] exploit the multi-scale neighborhood size, they may ignore the limited ability of networks that are built on a single type of feature inputs. We exhibit in Fig. 1 that several kinds of features are explored from a noisy input, such as the point coordinates of a local patch and the grid-like structures describing height distances, etc. Previous methods [6], [22] have proved that each of the extracted feature representations can be independently modelled for the point geometric properties. We validate that these features can mutually support each other, from different perspectives, and can be jointly incorporated into the estimation of point normals. This incorporation of multiple features is an effort in this work. Besides, we still apply the multi-scale scheme following [23] to promote learning robustness. These are the two main schemes adopted in this work (see Fig. 1). Second, for severely degraded data, it is open to recover sharp features and fine details by a general mapping learned from deep neural networks without any support of geometric information. We refer the readers to Fig. 2b that small handles are blurred in the red bounding box where the input domain is similar to planar areas. Over-smoothed results that ignore sharp edges (in the black bounding box of Fig. 2) are not allowed in the accuracy-required tasks like robotic grasp and surface reconstruction.

Motivated by the aforementioned challenges, we propose a normal refinement network, namely Refine-Net, which fuses important geometric information based on a multi-feature scheme. Our main contributions are summarized as follows:

- The proposed Refine-Net is a normal-to-normal system, designed to refine the initial normal of each point. We introduce two additional feature representations into the refinement steps in the context of neural networks.
- We present a novel connection module, which applies a flexible transformation to integrate useful information learned from additional features with the initial normal.
- We further extend Refine-Net into a normal refinement framework, with the generalization ability of both feature and normal modules.
- We propose a geometric method, namely multi-scale fitting patch selection (MFPS), for the initial normal estimation. It serves as our own initial normal estimator, which is better at capturing geometric supports by using an elaborately-designed patch selection scheme.
- A new point update algorithm is designed for the downstream construction tasks.

**Difference From our Conference Paper.** This work covers and extends our conference version [24] as follows: (i) we exploit more feature modules in the system, making a breakthrough from single-feature to a multi-feature refinement framework. Thus, our previous work can be seen as part of the network (normals&HMPs version in Section 6); (ii) we extend the connection module to three alternative choices including the newly proposed one which is more flexible in the high-dimensional space; (iii) the original geometric method is further improved, by using a novel selection solution, to determine normal directions in challenging regions of sharp edges; (iv) we demonstrate that Refine-Net can also improve previous normal estimation networks [21], [22], by directly taking their results as input; (v) we conduct more experiments on the outdoor scene dataset [25]; (vi) our results show clear improvements over Zhou et al. [24] in terms of normal estimation accuracy and noise-robustness.

2 RELATED WORK

Normal estimation for point clouds is a long-standing problem in academic. We will review previous researches from
traditional normal estimators to recent prevalent learning-based techniques.

2.1 Traditional Normal Estimators

Traditional approaches mainly focus on exploiting low-level geometric priors. The most well-known one is the Principal Component Analysis (PCA) [26], by analyzing covariance in a local structure around a point and defining its normal as the eigenvector corresponding to the smallest eigenvalue. Following this work, a lot of improvements have been proposed [27], [28], [29] to increase estimation precision. In particular, Mitra et al. [27] analyze the effects of neighborhood size, curvature, sampling density, and noise level for estimating normals. By carefully analyzing these local information, this method can find a suitable neighborhood size for each point. Another kind of normal estimation methods is based on Voronoi cells [30], [31], [32], [33]. However, this kind of methods cannot well estimate the normals of points near/on sharp features. Some other works are based on the improvement of a preliminary normal result. Typical approaches use adaptive moving least squares [34] and robust local kernel regression [35] to estimate normals as the gradient of an implicit surface which fits the local points and their preliminary normals. Later, based on the observation that neighbors belonging to different surface patches often result in large fitting errors, recent works are dedicated to selecting a plane approximating points from the same surface patch to estimate normals [18], [36], [37]. Under the assumptions that: i) surfaces are commonly composed of piecewise flat patches, and ii) geometric features are sparsely distributed over the entire shape, sparsity-based methods [38], [39], [40] show impressive results for those CAD-like models, especially in sharp feature preservation. Low-rank based methods [40], [41], [42], [43] explore the non-local geometric similarity to generate better normal fields on 3D surfaces. Some other methods, like Hough Transform [44], also yield pleasing results.

Overall, the geometric methods often resolve the normal estimation problem under certain assumptions or specific observations. Besides, these approaches contain a number of parameters, which require careful trial-and-error tuning to obtain decent results, especially for complex models.

2.2 Learning-Based Normal Estimators

Recently, learning-based methods have gradually shown their power in the task of normal estimation. We will review the related prior works in two main types: CNN-based and PointNet-based networks.

2.2.1 CNN-Based Methods

Boulch et al. [20] propose to project a discrete Hough space representing normal directions onto a structure amenable to CNN-based deep networks. Similarly, Roveri et al. [45] define a grid-like regular input to learn ideal normal results. Ben-Shabat et al. [21] propose to parameterize the local field using a multi-scale 3D modified Fisher vector which serves as the input to a deep 3D CNN architecture. In addition, a scale manager module is designed for selecting a suitable neighborhood size in a data-driven manner. As we investigate above, how to map the unstructured point data into a regular domain, is the key to these CNN-based methods.

2.2.2 PointNet-Based Methods

Another point cloud learning framework, namely PointNet [46], becomes increasingly popular in 3D domains, thanks to its ability to directly manipulate raw point cloud data. Inspired by this, Guerrero et al. [22] apply the PointNet architecture in a patch-based multi-scale form to predict normals. With a data-driven manner, this method allows to replace the difficult and error-prone manual tuning of parameters presented in the majority of existing traditional techniques. More recently, Zhou et al. [47] introduce an extra feature constraint mechanism and a multi-scale neighborhood selection strategy to estimate normals for 3D point clouds. Considering the typical problem that PointNet is not suitable for encoding local structures of a point cloud, Hashimoto et al. [48] propose a two-branch network that extracts both local and spatial features required for learning. The aforementioned learning-based deep networks try to directly regress point normals from a large dataset. However, they have downsides in terms of robustness to small input changes and interpretability. Contrary to these methods, Lenssen et al. [49] propose to embed deep learning in least-squares plane fitting, which leads to a fast and accurate algorithm. Specifically, a graph neural network is utilized to iteratively parameterize an adaptive anisotropic kernel that produces point weights for least-squares plane fitting.

In general, the aforementioned learning-based methods have two main drawbacks: 1) the weak ability of detail perceiving in/near challenging feature regions; 2) an existing learning-based method often exploits one single type of feature representation, neglecting the useful information from multiple types of features that can be combined together to co-support the normal estimation task. In this work, we focus on deploying a multi-feature scheme to explore different geometric supports in our normal refinement system.

3 Refine-Net

In this section, we first describe the overall architecture of Refine-Net (Section 3.1), followed by detailed elaboration of the component modules (Sections 3.2, 3.3, 3.4, and 3.5). Furthermore, we show that Refine-Net can be well extended to a generic normal refinement framework in Section 3.6.

3.1 Architecture Overview

The overall architecture of Refine-Net is shown in Fig. 3, which is designed to refine the initial normals by introducing multiple types of feature representations from an input point cloud. The initial normal of each point (Section 4) serves as the main feature in the network, and is processed by a two-step refinement for the final normal. First, we use bilateral filtering to extend the initial normal into a multi-scale version of normal vectors in order to expand the receptive field of local points (Section 3.2). Then, the filtered normals are fed into individual branches for a multi-feature refinement. Based on the extracted information, an additional point feature is introduced, which fuses with the normal in the connection module, in order to apply a rough refinement (Section 3.3). Similarly, in the second step, we
We consider the initial normal of each point as the main feature. Given the noisy inputs, previous attempts [23], [50], [51], [52], [53] that apply deep neural networks in the normal estimation later in Section 4. Previous attempts [23], [50], [51], [52], [53] that apply deep neural networks in the normal field have shown impressive results of recovering normal set of \( p_i \), denoted as \( \tilde{N}_i = \{ n_1, n_2, ..., n_X \} \). Each filtered normal will be fed into an individual branch for the following refinement operations.

Point-Wise Normal Reorientation. Before diving into these branches, we consider the point-wise normal reorientation in order to lower the learning difficulty. To this end, we apply a global rotation to the filtered normals to make them invariant to rigid transformation. Specifically, we first compute the normal tensor of point \( p_i \):

\[
T_i = \sum_{j=1}^{X} n_j T n_j^T, \quad n_j \in \tilde{N}_i. \tag{2}
\]

\( T_i \) fuses the normal characteristics of \( p_i \)'s local neighbors and can be decomposed as:

\[
T_i = \lambda_1 e_1 e_1^T + \lambda_2 e_2 e_2^T + \lambda_3 e_3 e_3^T, \tag{3}
\]

where \( \lambda_1 \leq \lambda_2 \leq \lambda_3 \) are the eigenvalues and \( e_1, e_2, e_3 \) are the corresponding eigenvectors. Then, a rotation matrix \( R_i \) can be constructed as [\( e_1, e_2, e_3 \)]. \( R_i \) rotates all normals in \( \tilde{N}_i \) to a local frame near the \( Z \)-axis of the global frame. Furthermore, if they are in the direction of negative \( Z \)-axis, we reverse them to the positive one. The ground-truth normal is also applied by the same rotation matrix to match the input. Such a pre-processing operation explicitly reorients the normals of each point to the same direction in order to tackle the inconsistency of global directions between input samples.

Normal-Based Clustering. Before training, we partition the training samples into \( K \) clusters via the k-means algorithm [23]. It is performed on the filtered normals \( \tilde{N}_i \) to gather similar inputs together in one group. For each cluster, we train a separate Refine-Net to recover the ground-truth normals. In this way, the network can focus on a specific kind of geometric features and intensively refine this type of normals. In the runtime test stage, a single point sample is distributed into one of these clusters in terms of \( ||\tilde{N}_i - C_k|| \leq ||\tilde{N}_i - C_l||, \forall k \). Here, \( C_k \) is the center of the \( k \)th cluster. We evaluate the performance of this cluster scheme in Section 6.

\[
\sum_{p|p \in N(i)} W_s(||p_i - p_j||) W_r(||\tilde{n}_i - \tilde{n}_j||)||n_j\), \tag{1}
\]

where \( \Lambda(*) \) is the vector normalization function, and \( W_s \) and \( W_r \) are Gaussian weight functions, i.e., \( W_s(x) = \exp(-x^2/(2\sigma_s^2)) \), representing the spatial similarity and normal similarity respectively between a pair of points. Here, standard deviations \( \sigma_s \) and \( \sigma_r \) are used. Instead of carefully tuning these parameters, we apply a multi-scale filtering [23] and obtain filtered normals from two designed parameter sets \( P_s = \{ \sigma_1, \sigma_2, ... \} \) and \( P_r = \{ \sigma_1, \sigma_2, ... \} \) by using every combination of the two parameters. Thus, together with the initial one, we get a total of \( X \) normals in the normal set of \( p_i \), denoted as \( \tilde{N}_i = \{ n_1, n_2, ..., n_X \} \). Each filtered normal will be fed into an individual branch for the following refinement operations.

Point-Wise Normal Reorientation. Before diving into these branches, we consider the point-wise normal reorientation in order to lower the learning difficulty. To this end, we apply a global rotation to the filtered normals to make them invariant to rigid transformation. Specifically, we first compute the normal tensor of point \( p_i \):

\[
T_i \sum_{j=1}^{X} n_j T n_j^T, \quad n_j \in \tilde{N}_i. \tag{2}
\]

\( T_i \) fuses the normal characteristics of \( p_i \)'s local neighbors and can be decomposed as:

\[
T_i = \lambda_1 e_1 e_1^T + \lambda_2 e_2 e_2^T + \lambda_3 e_3 e_3^T, \tag{3}
\]

where \( \lambda_1 \leq \lambda_2 \leq \lambda_3 \) are the eigenvalues and \( e_1, e_2, e_3 \) are the corresponding eigenvectors. Then, a rotation matrix \( R_i \) can be constructed as [\( e_1, e_2, e_3 \)]. \( R_i \) rotates all normals in \( \tilde{N}_i \) to a local frame near the \( Z \)-axis of the global frame. Furthermore, if they are in the direction of negative \( Z \)-axis, we reverse them to the positive one. The ground-truth normal is also applied by the same rotation matrix to match the input. Such a pre-processing operation explicitly reorients the normals of each point to the same direction in order to tackle the inconsistency of global directions between input samples.

Normal-Based Clustering. Before training, we partition the training samples into \( K \) clusters via the k-means algorithm [23]. It is performed on the filtered normals \( \tilde{N}_i \) to gather similar inputs together in one group. For each cluster, we train a separate Refine-Net to recover the ground-truth normals. In this way, the network can focus on a specific kind of geometric features and intensively refine this type of normals. In the runtime test stage, a single point sample is distributed into one of these clusters in terms of \( ||\tilde{N}_i - C_k|| \leq ||\tilde{N}_i - C_l||, \forall k \). Here, \( C_k \) is the center of the \( k \)th cluster. We evaluate the performance of this cluster scheme in Section 6.
3.3 Point Module

We introduce point features to prepare for the refinement. For a target point \( p_i \), a local patch \( P_i \) is defined as the neighboring point set \( \{ p_1, p_2, \ldots, p_n \} \) which is centered at \( p_i \). We propose to extract useful information from the patch points to refine the normal. To simplify the input space, the patch \( P_i \) is translated to origin and normalized by the patch radius. In order to match the filtered normals obtained above, \( P_i \) is also applied by the corresponding matrix \( R_i \), rotating the patch to the same direction. The rotation constraint is nontrivial [22], and in this case we can use the global directions extracted from the initial normals instead of computing directly from raw points.

To process the point set, we use the PointNet architecture [46] which is denoted as Point Module in Fig. 3. PointNet suggests to apply a set of shared functions independently on each point and collect the global feature using a symmetric function:

\[
f(P_i) = g(h(p_1), \ldots, h(p_n)),
\]

where \( g(*) \) is a symmetric function, i.e., max-pooling or sum, and \( h(*) \) denotes a multi-layer perceptron network (MLP). Here, the input \( p_j \in \mathbb{R}^3 \) only contains the \((x, y, z)\) coordinates. Then, our point module takes the global feature and processes it using several fully-connected layers which output a \( d_1 \) dimensional feature vector. The output dimension \( d_1 \) is determined according to the design of connection module, and we will discuss several choices later in Section 3.5.

3.4 Height-map Module

The height-map (HMP) feature is constructed related to both the point normals and coordinates which maps the local points onto a grid-like structure. The point \( p_i \) itself as well as one of its filtered normals \( n_i \in \bar{N} \) defines a tangent plane, and the associated HMP is built on this plane. Suppose that we build a matrix with \( m \times m \) bins that describes the local point positions, and the center of the matrix is located at \( p_i \). Similar as [6], we fill each bin by weightedly averaging the height distances of points in a ball neighborhood \( N_{\text{ball}}(b_j) \) of bin center \( b_j \):

\[
v_{b_j} = \frac{\sum_{p_k \in N_{\text{ball}}(b_j)} w(b_j, p_k) H(T(p_i, n_i), p_k)}{\sum_{p_k} w(b_j, p_k)},
\]

where \( T(p_i, n_i) \) is the tangent plane defined by \( p_i \) and \( n_i \), \( H(T(p_i, n_i), p_k) \) returns the signed distance from \( p_k \) to the plane and \( w(b_j, p_k) = \exp\left(-\frac{||b_j - p_k||^2}{\sigma_d^2}\right) \) is a spatial Gaussian weight function in which \( \sigma_d \) is the residual bandwidth. Assembling all HMPs yields each point’s height-map features \( \{ \text{HMP}_1, \text{HMP}_2, \ldots, \text{HMP}_X \} \). Note that, for the defined planes associated with HMPs, their Z-axes of local coordinate systems are also rotated by the matrix \( R_i \). Thus, the directions of X-axes and Y-axes are consistent among all samples.

The computed representation is fed into the height-map module (refinement 2 in Fig. 3). The \( X \) HMPs are concatenated and processed using several convolutional layers and max-pooling layers, followed by fully-connected layers (see Table 6). Similarly, the output of this module is a \( d_2 \) dimensional vector prepared for refinement.

3.5 Connection Module

In the aforementioned cases, we introduce additional features to refine the normal in each branch. The main discussion behind is how to establish a connection, e.g., between the normal vector \( V \in \mathbb{R}^3 \) and the point feature \( F \) (connection module of refinement 1 in Fig. 3), since they belong to distinct data types. The key idea is to use the point feature \( F \) to learn an intermediate transformation matrix \( T \), which should match the size of \( V \), and apply matrix multiplication to obtain the higher-level feature \( Y = T \cdot V \) (a new normal feature). Several choices to construct the transformation target \( T \) are discussed below.

**Rotation Matrix.** The simplest solution is to apply 3D rotation to normal vectors for the refinement. In this way, we can learn a rotation matrix from the input feature \( F \) which parameterizes the local points related to \( V \). Thus, the vector \( V \) can be rotated in the guidance of the point feature. To be more specific, we use the point module to learn a quaternion, which means \( d_1 = 4 \) is set in the output dimension, and construct a \( 3 \times 3 \) rotation matrix \( T \) from the unit quaternion.

**Transformation Matrix.** Another option that generates a transformation matrix does not require a quaternion for rotating. Intuitively, we set the output dimension as \( d_1 = 9 \) and simply reshape it to a \( 3 \times 3 \) matrix, which is a more direct solution. Note that, this does not act the same as a rotation operator. This construction can be seen as a common 3D transformation applied to the input normal vector \( V \) which is not designed strictly for rotating.

**Weight Matrix.** More generally, we construct a weight matrix \( T = [t_1, t_2, \ldots, t_p] \in \mathbb{R}^{p \times 3} \), in which each \( t_i = [w_{x,i}, w_{y,i}, w_{z,i}]^T \) contains learned weights corresponding to the 3D coordinate of \( V \). Thus, the transformation can be more flexible, producing output features with arbitrary dimensions. Through matrix multiplication, a weight matrix can extract more information from the input feature compared with a single \( 3 \times 3 \) matrix that is applied within the limits of 3D space. We set \( d_1 = p \times 3 \) in the output dimension, and reshape it to a \( (p, 3) \) matrix \( T \). This is the option that we adopt in this work.
To be more specific, in refinement 1 of Fig. 3, we set \(d_1 = 64 \times 3\) which converts the filtered normal into a new normal feature of dimension 64 (see the top part of Fig. 4). Similarly, a second refinement is applied by the height-map module with \(d_2 = 64 \times 64\) (see the bottom part of Fig. 4). The two features together with their filtered normal are stacked for the output.

By inference, the choices designed above can also provide a solid connection in other tasks with similar inputs. The weight matrix option receives two inputs with different sizes and can produce a high-dimensional output. We find in our experiments that the way of simply concatenating two distinct feature vectors leads to an incompatible connection which may oppositely limit the learning ability. The attempt to use matrix multiplication in the connection module shows its advantages compared with other learning schemes (see Section 6).

Output Module and Loss Function. The refined features from all branches are collected for the output. Refine-Net applies several fully-connected layers and outputs the final predicted normal \(N^* = (N_x, N_y, N_z)\). Batch normalization and ReLU are included in each layer of this module.

We train the network by minimizing the MSE loss between the output normal \(N^*\) and its ground truth \(\bar{N}\):

\[
Loss = ||\Lambda(N^*) - \bar{N}||^2 + \lambda E_{\text{reg}}, \tag{6}
\]

where \(\Lambda(\cdot)\) is the vector normalization function, and \(E_{\text{reg}}\) is the commonly-used \(L_2\) regularization to avoid overfitting, where \(\lambda = 0.02\) is used.

3.6 Normal Refinement Framework
Refine-Net uses two feature modules to refine the initial normal (normal module). When extended to a general normal refinement framework, these modules can be potentially replaced. That is, the normals from other methods can be further refined, and other feature modules can be developed and incorporated in the network. They are included by the connection modules in order to obtain the predicted results.

In this work, we introduce both point and height-map features in the network. We show that a single feature module is already enough for a promising result in Section 6. Further, more information related to the point geometric details can be absorbed by appending other feature modules in the branches.

The initial normal choice has a significant influence on the recovery results. We choose to use a traditional estimator, namely MFPS in Section 4, which can focus more on the geometric information compared with learning-based techniques. It is designed to preserve sharp features and fine details which are hard to recover by learning a straightforward mapping. Refine-Net that is trained on other typical initial normals can also produce a well-improved normal result.

4 Multi-Scale Fitting Patch Selection
As aforementioned, we choose to design a traditional geometric estimator for the initial normal estimation. Before computing the initial normal for each point, we first apply the local covariance analysis (Section 4.2 in [19]), to classify all points into candidate points (near sharp features) and smooth points (far from sharp features). The normals of smooth points can be simply computed by PCA.

To estimate the normals of candidate points, one common strategy [18], [36] is to randomly select three non-collinear points to construct a set of candidate planes and pick the one that best describes the underlying surface patch. Typically, let’s define the local patch \(Q^k_j\) (the \(k\)-nearest points of the point \(p_j\)) as the neighborhood of \(p_j\). Then, the selected plane on this patch can be determined by the following objective function:

\[
E_{Q^k_j}(\theta) = \frac{1}{|Q^k_j|} \sum_{p_k \in Q^k_j} W_\sigma(p_k, \theta), \tag{7}
\]

\[
\theta^*_Q^k \triangleq \arg \max_\theta E_{Q^k_j}(\theta), \tag{8}
\]

where \(W_\sigma(p_k, \theta) = \exp(-r_{k,0}^2/\sigma^2)\) is the Gaussian function in which \(r_{k,0}\) denotes the distance from the point \(p_k\) to the plane \(\theta\), and \(\sigma\) is the residual bandwidth of \(p_j\). Thus, \(\theta^*_Q^k\) is the selected plane, and the normal of this plane can be seen as the patch normal of \(Q^k_j\).

4.1 Multi-Scale Scheme
If \(p_j\) lies in the neighborhood of an edge/intersection, the corresponding patch \(Q^k_j\) might be corrupted by points from other sides, over-smoothing the resulting estimated normal of \(p_j\) (see Fig. 5 left). Considering this, we propose to select a more consistent and suitable neighborhood which can make it easier to determine the normal. For each candidate point \(p_j\), our method tries to find the best fitting patch that contains \(p_j\). Please note that normals of smooth points are already computed by PCA.

First, the patch size is extended to a multi-scale parameter set \(K = \{k_1, k_2, \ldots, k_d\}\). For each point \(p_j\) in a point cloud, every scale size \(k_i \in K\) defines a patch \(Q^j_i\) similarly as above, and we compute the plane of this patch by Eq. (8). Then, for a candidate point \(p_j\), we denote \(S_j = \{Q^j_i | p_i \in Q^j_i\}\) as all patches containing \(p_j\). We aim to search for, among all patches in \(S_j\), the most isotropic one and to use the normal of the plane of this patch as the estimated initial normal of \(p_j\). Fig. 5 (right) shows a better neighboring patch to determine the normal on sharp edges, rather than the one
centered at the target point (Fig. 5 left). Moreover, the multi-scale scheme provides more choices for complicated regions where a single-scale size cannot fit in structures with varying point densities (Fig. 5 middle). The scale parameter is set as $K = 50, 100, 150$ by default in our experiments. The consistency of any patch $Q^i_j$ can be measured as:

$$D(Q^i_j) = E_Q^i(\theta_Q^i)\eta(k_i),$$

$$\eta(k_i) = \beta + (1 - \beta) \frac{k_i - k_{\min}}{k_{\max} - k_{\min}},$$

where $E_Q^i(\theta_Q^i)$ comes from Eq. (7) and $k_i \in K$. The larger $E_Q^i(\theta_Q^i)$ is, the smaller the plane fitting error is, and $\eta(k_i)$ is a tradeoff parameter used to punish relatively small patches because larger ones are preferred in noisy areas. We set $\beta = 0.9$ empirically. The measurement values are sorted, preparing for the following patch selection.

### 4.2 Fitting Patch Selection

Since $S_i$ may contain patches sampled on different sides of the intersection, a decision should be made to abandon undesired ones (Fig. 6b). To this end, we first select several anisotropic patches from $S_i$ and pick the one that is fitting to the candidate point $p_i$ (Fig. 6c). Specifically, starting from the largest $D(Q^i_j)$, our method adds a new patch $Q^i_{\ast} \in S_i$ and its corresponding normal $n_{\ast}$ into the anisotropic patch set $A_i = \{(Q_{1}^{ani}, n_{1}^{ani}), (Q_{2}^{ani}, n_{2}^{ani}), \ldots\}$, if $D(n_{\ast}, n^{ani}_j) > w_j$, $\forall j$ where the metric $D$ is calculated as the angular error of two normals. $w_j$ is an angular threshold, and we set $w_j$ to $60^\circ$ in our experiments.

The final step is to choose one patch from $A_i$, which can best describe the underlying surface of $p_i$. Following [56], for each $Q^{ani}_j \in A_i$, we find a reference point $p^\text{ref}_j$ by projecting $p_i$ onto its plane $\theta^\text{ani}_Q$ (see the black arrows in Figs. 6b and 6c). Second, each normal $n^{ani}_j$ is reoriented to the exterior of the surface, which means that it must satisfy the following condition:

$$\sum_{p_k \in N_i} n^{ani}_j \cdot (p^\text{ref}_j - p_k) < 0,$$

where $N_i$ is the neighborhood of $p_i$. The values $\{n^{ani}_j \cdot (p^\text{ref}_j - p_k) \}|j = 1, 2, \ldots\}$ are computed, and the minimum value corresponds to the initial normal of the candidate point $p_i$. Note that, $p^\text{ref}_j$ is only used for patch selection, which is not a new point in the point cloud. We refer the reader to [56] for a detailed explanation.

Although they may be sensitive to high-level noise, geometric estimators are often more effective to predict point normals around tiny details. Such initial normals which perform better in sharp feature regions improve the final Refine-Net results significantly by preserving geometric information that is difficult to recover from learning. We give a better understanding of our geometric method in Section 7, compared with those learning-based solutions.

### 5 Evaluation

In this section, we evaluate our Refine-Net on both synthetic and real-scanned datasets. Detailed network architectures and comparisons are provided.

#### 5.1 Experimental Setup

**Network Architecture.** The network architecture of Refine-Net is illustrated in Fig. 3. In the first refinement, following [22], the patch radius of input points is set related to the length of the bounding box of the point cloud. The size is 0.05 and a maximum of 300 points is used. For neighborhoods with more than 300 points, we apply a random sampling, and for those with fewer points, we pad the rest with zeros (the patch center). Here, we use the PointNet architecture [46]. The T-Net module in PointNet is not included since the input points are already reoriented according to their initial normals. First, a shared MLP ($64, 64, 64, 128, 1024$) is applied to get a 1024 dimensional feature per point. Then, a maxpooling layer is used to get the global feature for the entire patch, after which fully-connected layers ($256, 128$) are applied for the output $d_1$-dimensional vector. Dropout with a keep probability of 0.3 is introduced in the last two fully-connected layers. The output of the point module follows the connection designs we discussed in Section 3.5.

In the second refinement, for the HMP construction, we use a $7 \times 7 (n = 7)$ height-map grid, and the neighborhood size is the same as the patch radius above. Then, we process these HMPs using several convolutional layers followed by fully-connected layers (see Table 6 left). In connection modules, we use the weight matrix option, i.e., $d_1 = 64 \times 3$ and $d_2 = 64 \times 64$ as discussed in Section 3.5. Other connection choices will be evaluated later in Section 6. As a consequence, two normal features ($64$) and the original filtered normal, extended by FC ($64, 64$), are concatenated as the output of each branch. Finally, for the output module, fully-connected layers ($512, 256, 3$) are used to obtain the predicted normal. Dropout is included similarly in all fully-connected layers. All layers use the batch normalization and ReLU.

**Initial Normal and Filtering.** The initial normal is computed for each point (Section 4) before the refinement steps. The proposed geometric normal estimation method plays an important role in capturing detailed structures and sharp features. We will also evaluate these normal results and demonstrate the effectiveness of Refine-Net based on several different initial normals. In MFPS, the multi-scale neighborhood size $K$ is the most practical parameter that could be tuned. We set $K = 50, 100, 150$ empirically for the synthetic models. The parameters used in bilateral filtering are set as $P_3 = \{0.025d_4, 0.05d_3\}$ and $P_r = \{0.1, 0.2, 0.35, 0.5\}$.
where \( l_d \) is the diagonal length of the bounding box. Thus, together with the initial normal (non-filtered normal), Refine-Net contains 9 individual branches.

5.2 Experiments on the PCPNet Dataset

Dataset. First, we compare our model with state-of-the-art deep networks on the PCPNet [22] dataset. This dataset consists of 30 shapes from a mixture of figurines, man-made objects and differentiable surfaces, with 8 shapes for training and 22 for testing. All shapes are uniformly sampled with 100k points. We follow the experimental settings of [21], [22]. Each shape is augmented by Gaussian noise with standard deviations of 0.12\%, 0.6\% and 1.2\% of the diagonal length of the bounding box. In addition, two categories with different sampling densities (Gradient and Stripes) are generated for each shape. For evaluation, we use the same subset of 5k points from each point cloud, following the protocol of Guerrero et al. [22].

Training. We use the SGD optimizer with a learning rate of 0.0001, and we do not use the learning rate decay in all epochs. The batch size is 512 for all training clusters and the momentum for batch normalization is 0.9. We implement Refine-Net using PyTorch and train it on a RTX 2080 Ti GPU.

Results. We report the root mean square error (rmse) in Table 1. Here, we show both the initial normal estimation (MFPS) and Refine-Net results with MFPS initial normals (our full pipeline). Following [21], we give the single-scale (ss) and multi-scale (ms) versions for HoughCNN [20] and PCPNet [22]. For traditional PCA [26] and Jet [28], results for medium neighborhood sizes are shown. It can be seen that our method achieves state-of-the-art performances on most of the categories. For shapes with high-level noise, general directions of point normals are predicted correctly, while fine details can hardly be observed for all of the methods. Also, we can see that our geometric method already performs promisingly against other models on point clouds with varying densities, thanks to the multi-scale scheme for patch selection.

5.3 Experiments on More Synthetic Data

Dataset. In order to conduct a more comprehensive evaluation with both the traditional geometric and deep learning-based methods, we use another synthetic dataset from

### Table 1
Normal Estimation Results on the Synthetic PCPNet Dataset [22], Evaluated as Angular RMS Errors

| Category      | Noise \( \delta \) | Density | Average |
|---------------|--------------------|---------|---------|
|               | 0.00125 | 0.006 | 0.012 |
| PCA [26]      | 12.29   | 12.87 | 18.38 | 27.5 | 12.81 | 13.66 | 16.25 |
| Jet [28]      | 12.23   | 12.84 | 18.33 | 27.68 | 13.39 | 13.13 | 16.29 |
| HoughCNN-ss [20] | 10.23   | 11.62 | 22.66 | 33.39 | 12.47 | 11.02 | 16.9 |
| HoughCNN-ms [20] | 10.02   | 11.51 | 23.36 | 36.7 | 10.67 | 11.95 | 17.37 |
| PCPNet-ss [22] | 9.68    | 11.46 | 18.26 | 22.8 | 11.74 | 13.42 | 14.56 |
| PCPNet-ms [22] | 9.62    | 11.37 | 18.87 | 23.28 | 11.7 | 11.16 | 14.34 |
| Nesti-Net [21] | 6.99    | 10.11 | 17.63 | 22.28 | 8.47 | 9.00 | 12.41 |
| Lenssen et al. [49] | 6.72 | 9.95 | 17.18 | 21.96 | 7.75 | 7.51 | 11.84 |
| DeepFit [55]  | 6.51    | 9.21 | 16.72 | 23.12 | 7.31 | 7.92 | 11.8 |
| MFPS          | 7.22    | 11.19 | 17.91 | 24.07 | 7.27 | 7.87 | 12.58 |
| Our full pipeline | 6.27   | 9.18 | 16.59 | 22.57 | 6.61 | 7.02 | 11.37 |

### Table 2
Comparison of Normal Estimation Errors on the Synthetic Dataset [23], Evaluated as Mean Angular Error (Mean) and Root Mean Square Error (Rmse)

| Category      | BigNoise | SharpFeature | RichFeature | SmoothSurface | Average |
|---------------|----------|--------------|-------------|--------------|---------|
|               | mean     | rmse         | mean        | rmse         | mean    | rmse   |
| PCA [26]      | 8.53     | 13.80        | 8.70        | 13.61        | 6.94    | 9.84   | 8.04    | 6.83   | 8.05    | 11.02  |
| HF [44]       | 11.73    | 16.23        | 5.26        | 9.75         | 5.81    | 8.34   | 3.87    | 5.54   | 6.67    | 9.96   |
| HoughCNN1s [20] | 8.76     | 15.68        | 5.40        | 11.20        | 6.00    | 9.05   | 5.05    | 7.44   | 6.30    | 10.84  |
| HoughCNN3s [20] | 8.61     | 15.16        | 5.52        | 11.23        | 5.61    | 8.49   | 5.89    | 8.55   | 6.41    | 10.86  |
| HoughCNN5s [20] | 10.78   | 16.39        | 6.48        | 12.22        | 6.46    | 9.51   | 7.40    | 10.24  | 7.28    | 12.09  |
| PCPNet-ss [22] | 9.61    | 14.26        | 8.12        | 16.23        | 6.84    | 9.78   | 12.84   | 15.87  | 10.53   | 14.04  |
| PCPNet-ms [22] | 9.09    | 13.52        | 7.90        | 11.09        | 5.97    | 8.15   | 5.27    | 6.97   | 7.06    | 9.93   |
| LRR [19]      | 5.77     | 12.08        | 4.39        | 8.37         | 4.86    | 7.18   | 4.66    | 8.22   | 4.92    | 8.96   |
| PCV [18]      | 5.89     | 11.92        | 4.52        | 8.50         | 4.80    | 6.87   | 3.97    | 6.24   | 4.80    | 8.38   |
| Nesti-Net [21] | 5.10    | 10.86        | 4.28        | 7.89         | 4.62    | 6.37   | 4.20    | 5.76   | 4.55    | 7.72   |
| MFPS          | 5.72     | 11.72        | 4.36        | 7.86         | 4.82    | 6.97   | 4.10    | 6.32   | 4.75    | 8.22   |
| PCPNet-ms + Refine-Net | 7.11   | 12.08        | 6.08        | 8.91         | 5.23    | 7.02   | 4.13    | 5.15   | 5.64    | 8.29   |
| Nesti-Net + Refine-Net | 4.79  | 10.63        | 3.76        | 6.93         | 4.25    | 5.74   | 3.64    | 4.78   | 4.11    | 7.02   |
| Our full pipeline | 4.74   | 10.73        | 3.37        | 6.39         | 4.07    | 5.65   | 3.28    | 4.56   | 3.87    | 6.83   |
Wang et al. [23]. The training set contains 21 synthetic triangular mesh models with varying sampling densities and is manually divided into three categories: CAD-like models, smooth models and feature-rich models. Each type is collected from shapes representing typical challenging geometric features. We simply extract the mesh vertices as point samples and the mesh normals as ground-truth normals in order to preserve the original features for training. To generate noisy inputs, we introduce Gaussian noise for each point cloud with standard deviations of 0.1%, 0.2% and 0.3% of the diagonal length of the bounding box. The final training dataset contains 1.5M points from 63 noisy point clouds, in which 20% of point samples are used for validation.

Table 3

| Category       | BigNoise | SharpFeature | RichFeature | SmoothSurface | Average |
|----------------|----------|--------------|-------------|---------------|---------|
|                | PGP5     | PGP10        | PGP5        | PGP10         | PGP5    | PGP10    | PGP5    | PGP10    | PGP5    | PGP10    |
| PCA [26]       | 0.591    | 0.754        | 0.564       | 0.711         | 0.533   | 0.764     | 0.648   | 0.889     | 0.584   | 0.780    |
| HF [44]        | 0.316    | 0.608        | 0.739       | 0.893         | 0.601   | 0.853     | 0.757   | 0.937     | 0.603   | 0.823    |
| HoughCNN1s [20]| 0.624    | 0.782        | 0.774       | 0.882         | 0.618   | 0.828     | 0.682   | 0.875     | 0.675   | 0.842    |
| HoughCNN3s [20]| 0.610    | 0.784        | 0.756       | 0.868         | 0.637   | 0.847     | 0.634   | 0.825     | 0.659   | 0.831    |
| HoughCNN5s [20]| 0.468    | 0.691        | 0.695       | 0.835         | 0.576   | 0.806     | 0.530   | 0.748     | 0.567   | 0.770    |
| PCPNet-ss [22] | 0.473    | 0.732        | 0.381       | 0.620         | 0.527   | 0.791     | 0.261   | 0.537     | 0.410   | 0.670    |
| PCPNet-ms [22]| 0.464    | 0.737        | 0.508       | 0.772         | 0.564   | 0.840     | 0.613   | 0.887     | 0.537   | 0.809    |
| LRR [19]       | 0.732    | 0.885        | 0.793       | 0.901         | 0.675   | 0.895     | 0.744   | 0.902     | 0.736   | 0.896    |
| PCV [18]       | 0.725    | 0.882        | 0.767       | 0.919         | 0.672   | 0.895     | 0.765   | 0.930     | 0.732   | 0.906    |
| Nesti-Net [21] | 0.767    | 0.888        | 0.764       | 0.901         | 0.682   | 0.911     | 0.730   | 0.921     | 0.736   | 0.905    |
| MFPS           | 0.732    | 0.884        | 0.779       | 0.918         | 0.674   | 0.896     | 0.751   | 0.919     | 0.734   | 0.904    |
| PCPNet-ms + Refine-Net | 0.623 | 0.808 | 0.608 | 0.852 | 0.616 | 0.880 | 0.703 | 0.945 | 0.637 | 0.871 |
| Nesti-Net + Refine-Net | 0.783 | 0.895 | 0.796 | 0.923 | 0.709 | 0.927 | 0.763 | 0.948 | 0.763 | 0.923 |
| Our full pipeline | **0.788** | **0.903** | **0.842** | **0.948** | **0.722** | **0.931** | **0.802** | **0.959** | **0.788** | **0.935** |

Higher is better.

Fig. 7. Visual comparison of estimated normals with other methods: PCA [26], HF [44], PCV [18], PCPNet [22] and Nesti-Net [21]. We also give our MFPS and Refine-Net results. The 1st and 3rd rows denote the normal results rendered in RGB colors. The average normal angular errors (mean) are: (1st row) 4.34, 6.37, 1.63, 1.49, 5.37, 1.66, 0.88; (3rd row) 8.14, 6.27, 5.60, 5.71, 6.22, 6.16, 4.99. The 2nd and 4th rows denote the normal errors, mapped to a heatmap according to the color bar on the right. It can be seen that more errors occur near sharp edges (2nd row) and facial details (4th row). Our method suffers the least from these challenges.
Gaussian noise with a standard deviation of

To test the normal results on synthetic data, we use the test set from [18], [23]. Similarly, the test set includes 4 categories: SharpFeature, SmoothSurface, RichFeature and BigNoise, in which there are 11, 8, 8 and 8 models respectively. For data augmentation, each point cloud in the BigNoise category is perturbed by Gaussian noise with a standard deviation of 0.05%, 0.1% and 0.15% of the diagonal length of the bounding box. For the RichFeature category, 0.05%, 0.1%, 0.15% and 0.2% noise are introduced and 0.2%, 0.3%, 0.4% and 0.5% for the BigNoise category. Our test set contains totally 121 point clouds with over 3M point samples.

**Results.** The evaluation metrics [21] for normal estimation are mean angular error (mean), root mean square error (rmse) and proportion of good points metric PGP\(\alpha\) (percentage of points with angular error below threshold \(\alpha\)) where \(\alpha \in [5^\circ, 10^\circ]\).

Tables 2 and 3 show the results on the synthetic models. We compare our method, including MFPS initial normal results and the full pipeline, with several traditional and deep-learning state-of-the-arts: PCA [26], HF [44], LRR [19], PCV [18], HoughCNN [20], PCPNet [22] and Nesti-Net [21]. We retrain HoughCNN, PCPNet and Nesti-Net on this dataset for a fair comparison. In addition, HoughCNN has three versions with different scales and PCPNet has the single-scale and multi-scale versions. We evaluate all these versions. For the above methods, we set a same neighborhood size, i.e., \(K = 100\) for the three categories: SharpFeature, RichFeature and SmoothSurface. In particular, the HoughCNN3s considers 3 scales, \(K = 50, 100, 200\), which is recommended in their paper. For the BigNoise category, we double the neighborhood size, i.e., \(K = 100, 200, 400\) for HoughCNN3s, and \(K = 200\) for other single-scale methods. Besides, HoughCNN5s considers 5 scales, \(K = 32, 64, 128, 256, 512\), for all categories. All other parameters are set using default values. From the above comparison, our method outperforms others in all categories, especially on models with high-level noise.

For visual comparisons, we show normal results and corresponding angular errors (mapped to a heatmap) of different methods in Fig. 7. For PCPNet, we display the best results out of its two versions. We can see that Refine-Net produces more accurate normals on sharp edges and facial details in the 1st and 3rd rows respectively.

**Using Other Initial Normals.** On the other hand, Refine-Net is able to refine the results of other deep networks. We show in Tables 2 and 3 that normals predicted by PCPNet [22] and Nesti-Net [21] can be improved significantly by replacing the initial normal in our system. In this experiment, we train the networks separately, and then Refine-Net takes the Nesti-Net/PCPNet outputs as initial normals and generates the final predicted normals. It is also practical to design an end-to-end network where, however, the filtering for multi-scale normals in the refinement should be abandoned since it requires complete normal results of the point cloud. We prefer to keep this multi-scale design in the network, which considers different normal directions in the neighborhood and performs better on sharp features. Moreover, Fig. 8 depicts a visual comparison by using Nesti-Net/PCPNet normal results as the initial normals in our framework. Clear improvements are seen from regions of sharp features and details in the bounding box.

### 5.4 Experiments on Real Scans

**Dataset.** We extend our Refine-Net to recover truth surface normals on scanned point clouds. In this task, the real-scanned data reveals more challenges, such as the fluctuation on flat surface, originated from the projection process of 3D sensors. The non-Gaussian and discontinuous noise...
heavily interferes both traditional and learning-based estimators against recovering the underlying surface.

The training set comes from [23]. This dataset contains scanned meshes of seven real models using Microsoft Kinect v1. There are 71 scans in the training set and the total number of samples is about 2.6M. Further, for each scan, they use another high-resolution scanned surface to help building ground truth normals. The benchmark test set of [23] contains 73 scans and 930k samples with ground truth normals.

Moreover, we test our trained network on NYUD-V2 [57] and Paris-rue-Madame [25] datasets. NYUD-V2 captures a variety of indoor scenes recorded by both the RGB and

Fig. 9. Visual comparison of normal estimation results on the scanned point clouds from the NYU Depth V2 dataset [57]. We show the scanned depth map (noisy input) in (a) and the corresponding RGB image in (g). PCPNet [22] and Nesti-Net [21] tend to smooth the captured small objects and edges. Our method clearly produces better normal results with nice geometric details.

Fig. 10. Visual comparison of estimated normals on two real-scanned models. The average normal angular errors (mean) are: (1st row) 8.3, 10.2, 8.7, 8.7, 7.8, 7.0; (2nd row) 10.0, 11.9, 10.0, 10.2, 10.2, 9.5, 8.6. Our Refine-Net is better at recovering facial details of the two models and is able to remove noise on flat areas.
depth cameras. We extract 3D point clouds from the depth channel as the noisy input. Missing pixels of raw depth maps are filled in using [58]. For large-scale outdoor scenes, we employ the Paris-rue-Madame dataset [25].

Architecture. We use a similar Refine-Net architecture as in our synthetic experiments. The filtering parameters are the same, but the non-filtered normal branch is not included. Thus, 8 branches are developed in the network. Since the real-scanned point cloud tends to be dense and large, in the point module, the patch size is set to be 0.03, and the maximum number of points is extended to 500. The neighborhood in HMP construction is also resized similarly and the height-map grid size is $m = 7$.

Results. Quantitative comparisons of normal estimation results from [23] are shown in Table 4. Please note that we train all networks on the same dataset for a fair comparison. Our method outperforms the state-of-the-arts according to all the metrics. Visual comparisons on two real models are illustrated in Fig. 10. Refine-Net produces more faithful results on, e.g., the eyes and nose of the boy and girl model respectively while removing undesired noise on the model bases.

For the NYUD-V2 dataset, we show visual comparisons of several real scans of indoor scenes in Fig. 9. Traditional geometric estimators (HF and our MFPs) can preserve tiny details but with the price of retaining noise (see Figs. 9b and 9c). Nesti-Net and PCPNet can well smooth the noisy surface, but over-smooth geometric features (see Figs. 9e and 9d). Our Refine-Net can well handle both challenges. For instance, the produced normal results of detailed objects are more faithful on, e.g., the toys and bottles from the 3rd and 4th rows respectively. Meanwhile, our method is able to remove the scanner noise on the flat surfaces. Fig. 12 depicts more visual comparisons. It can be seen clearly that better normal results can be recovered when the predicted normals from other networks are refined in our framework. We refer the reader to look closely at the drawers and handles in the 1st and 2nd rows respectively. Other methods tend to smooth these details as their local regions are similar to planar areas. Refine-Net, however, produces results with nice details which are more faithful to the ground-truth surfaces. Our method can utilize additional information extracted from two feature modules and outperforms other networks on real-scanned point clouds.

Finally, we show more results on the Paris-rue-Madame [25] dataset of large-scale scenes in Fig. 11. Compared with PCPNet and Nesti-Net, our network produces better normal

---

**Fig. 11.** Visual comparison of normal estimation results on the large-scale outdoor scenes from Paris-rue-Madame [25] dataset. Our method produces more faithful results on the vehicle. Details on the wall are clearly recovered.

---

**Fig. 12.** Visual comparison of normal estimation results on the scanned point clouds from the NYU Depth V2 dataset [57]. “PCP+Ours” indicates that Refine-Net takes PCPNet [22] results as initial normals. “Nesti+Ours” indicates the Nesti-Net [21] results as initial normals.
results on the vehicle in a real-world environment. Also, the
details on the wall are clearly recovered, which are ignored
by other methods.

5.5 Efficiency
We test our Refine-Net and other network models on a sin-
gle RTX 2080 Ti GPU and report the execution times (per 1k
points) and model sizes in Table 5. The reported time of our
method includes the initial normal estimation and Refine-Net
forward time.

5.5 Efficiency

The computing time of our geometric method depends largely on the
shape geometric structures, with about 150s per shape on
the PCPNet dataset (100k points for each shape). It is run on
the CPU and can be further sped up by parallel computing.
The forward pass time of Refine-Net is 0.8s (per 1k points),
which is relatively fast among the related normal estimation
networks. This can be contributed to the smaller network size and our efficient connection modules.

6 Ablation Studies
In this section, we explain some of the architecture choices and
evaluate the effectiveness of several components in Refine-Net.

Feature Modules. Compared with similar works [22], [23],
we introduce two additional feature modules into the
Refine-Net framework apart from the main normal feature.
Since the local points and HMPs are both generated related
to the normal, Refine-Net can utilize the extracted informa-
tion and improve the initial normal. To demonstrate the
advantages of this combination (normals&points&HMPs),
we design several ablations and evaluate their performan-
ces on the synthetic dataset [23] as shown in Table 7:

- normals - We first consider using normals of each
  point (from multi-scale bilateral filtering) to recover
  the ground-truth normal. The filtered normals are
  concatenated and fed directly to several fully-con-
nected layers (256,128,3). The network setting fol-
lows our output module.
- normals&points - This design solely uses point fea-
ture for a refinement. That is, the second refinement
  is abandoned, and the transformed feature after
  point module is exported from each branch. The out-
put dimensions are the same as in our Refine-Net.
- normals&HMPs - Similarly, we test the network
  which uses normals and HMPs as input. The height-
map module outputs a $T$ transformation matrix
  applied to the normal, which uses the architecture
  in Table 6 (right). The first refinement (point module)
is not included.

Connection Module Choices. The key idea behind our
approach is to learn a transformation term $T$ and to combine
the two feature inputs by matrix multiplication. Apart from
the adopted choice (weight matrix), we test another two
options discussed (rotation matrix and transformation
matrix), which are also alternative in other situations. Results
are shown in Table 7. Further, we explore other solutions to
combine on two different data inputs:

- concat - The feature module outputs a vector $T$ with
  the same dimension as $V$. Then, in the connection

| Category | BigNoise | SharpFeature | RichFeature | SmoothSurface | Average |
|----------|----------|--------------|-------------|---------------|---------|
|          | mean     | rmse         | mean        | rmse          | mean    | rmse    |
| MFPS (initial normal) | 5.72 | 11.72 | 4.36 | 7.86 | 4.82 | 6.97 | 4.10 | 6.32 | 4.75 | 8.22 |
| simple MFPS | 7.28 | 16.75 | 5.86 | 12.47 | 5.01 | 7.91 | 4.89 | 8.49 | 5.76 | 11.40 |
| normals | 5.51 | 12.53 | 3.86 | 7.46 | 4.52 | 6.55 | 3.72 | 5.64 | 4.40 | 8.04 |
| normals&HMPs | 5.50 | 12.48 | 3.74 | 7.26 | 4.40 | 6.32 | 3.61 | 5.38 | 4.31 | 7.86 |
| normals&points | 4.83 | 10.86 | 3.55 | 6.67 | 4.14 | 5.79 | 3.37 | 4.78 | 3.97 | 7.03 |
| concat | 4.83 | 10.98 | 3.61 | 6.91 | 4.23 | 5.99 | 3.48 | 5.06 | 4.04 | 7.23 |
| residual | 4.93 | 11.10 | 3.59 | 6.77 | 4.19 | 5.87 | 3.40 | 4.88 | 4.03 | 7.15 |
| Refine-Net-Rot | 4.84 | 11.03 | 3.56 | 6.78 | 4.19 | 5.90 | 3.43 | 4.93 | 4.00 | 7.16 |
| Refine-Net-Trans | 4.87 | **10.71** | 3.54 | 6.49 | 4.10 | 5.67 | 3.29 | 4.59 | 3.95 | 6.87 |
| Refine-Net-Weight | **4.74** | 10.73 | **3.37** | **6.39** | **4.07** | **5.65** | **3.28** | **4.56** | **3.87** | **6.83** |
module of each refinement, the two different feature inputs are concatenated and connected through fully-connected layers (64, 64). The output dimensions in each branch are identical to our Refine-Net.

- residual - We replace the matrix $T$ by a single vector with the same size as $V$ ($1 \times 3$ normal), which learns the residual (noise on point normal) between the input and the ground truth. The intermediate feature $Y$ is obtained by $V$ plus the residual.

Experimental results in Table 7 show that the multi-feature scheme significantly improves the normal estimation accuracy compared with using single features as input. Note that, our previous work [24] can be seen as the normals&HMPs version, and Refine-Net clearly achieves better performance on top of this. On the other hand, the practice that combines two features using matrix multiplication shows its advantages over other learning schemes. The weight matrix choice is better since it captures more information in a high-dimensional vector.

Initial Normals With Fitting Patch Selection. For the initial normal estimation, we propose a fitting patch selection method to choose one from the multi-scale candidate patches. To quantitatively evaluate this method, we compute the initial normal results without fitting patch selection (simple MFPS), as shown in Table 7. We can see that the proposed MFPS clearly performs better in all categories, especially on shapes with sharp edges. Without such a selection scheme, it would simply choose the patch with highest score within the neighborhood. However, the most consistent patch may reflect a normal directing the wrong side as illustrated previously in Fig. 6.

$L1$ Loss versus $L2$ loss. The $L1$ loss is known to be less sensitive to outliers and better at preserving sharp estimations. In this experiment, we compare the training performances using $L1$ loss and $L2$ loss in the task of normal estimation. We use the PCPNet dataset which is categorized according to noise levels and report the rms errors in Table 8. It shows that the $L1$ loss performs better on the noise-free point clouds while it is more sensitive to high-level noise. Furthermore, in Fig. 15, we show colormaps reflecting the error distances where a specific color (blue: $L1$, red: $L2$) is used to represent points which perform better with $L1$ or $L2$ on shapes with different noise levels. The $L2$ loss performs clearly better on sharp features and tiny details, especially at a lower noise level. Therefore, we choose to use the $L2$ loss which is more robust to noise and better at recovering normals in challenging regions with sub-optimal initial normals.

### 7 LEARNING TO REFINE NORMALS

To achieve a better understanding of Refine-Net, we provide two insights exploring deeper into the proposed model.

*Learning to Refine Normals.* Refine-Net is a generic normal refinement framework to “refine” initial normals. When inputting better initial estimations, it is expected to also achieve better final normal results. However, we find it is

---

**TABLE 8**

| Noise $\delta$ | None | 0.0012 | 0.006 | 0.012 |
|---------------|------|--------|-------|-------|
| L1            | 6.05 | 9.58   | 17.13 | 23.50 |
| L2            | 6.27 | 9.18   | 16.59 | 22.57 |

**Fig. 13.** Left: The averaged angular error (mean) improves when the cluster number $K_c$ increases. We use $K_c = 4$ by default and a larger number improves the results very slightly. Right: We show points divided into different clusters when using $K_c = 2$ (a) and $K_c = 4$ (b). In (b), the 4 clusters are visualized in two parts, in which the left part (with red and blue colors) can capture points on geometric features. Color coding is given on the right.

---

**Fig. 14.** Visual comparison of normal results and error heatmaps between different methods. We show Nesti-Net and “Nesti-Net + Refine-Net” results in (a) and (b) respectively. (c) denotes our initial normals computed by MFPS, (d) denotes the Refine-Net with only the point module, and (e) is the result from our full pipeline. Zoom in to see clearer.
not always true. For example, it shows in Tables 2 and 3 that combining Refine-Net with MFPS achieves better results than combining Refine-Net with Nesti-Net, though Nesti-Net has better initial normals than MFPS in some of the categories.

In order to explain such phenomena, we explore deeper to see how Refine-Net improves the geometric normals by means of deep learning techniques. In this experiment, we train our full pipeline (MFPS + Refine-Net) and Nesti-Net [21] on the synthetic dataset from Wang et al. [23]. Then, the Nesti-Net normal results are further refined by our Refine-Net (see Fig. 14b). By absorbing prior geometric knowledge, initial normals computed by our MFPS can be more effective for detecting sharp edges and details, while they may also reflect discontinuity of the smooth surface caused by high-level noise (see Fig. 14c). On the other hand, Nesti-Net tends to produce piecewisely smooth normals, representing general directions of the underlying surfaces. Refine-Net can polish the initial normals from different methods while the improvements can be better on the MFPS normals (see Fig. 14e), by recovering the sharp edges (preserving details) and removing perturbations in the flat areas (denoising). Further, we show another result of refined normals using only point features in Fig. 14d. We can see that using a single feature module can already produce decent denoised normals on noisy point clouds and Refine-Net with multiple feature inputs has better performances on sharp features.

Cluster-Based Scheme. Inspired by [23], Refine-Net uses a cluster-based scheme to promote its learning ability, which divides input samples into $K_c$ clusters before training. In all our experiments, including the variants of the proposed network, we set $K_c = 4$ by default. A larger number of clusters slightly improves the performance but heavily expands the network (see Fig. 13 left). This clustering scheme can detect
similar features among all training samples especially for points on geometric features that are hard to process (see Fig. 13 right). Thus, the network can focus on a specific kind of input data.

8 APPLICATION
As known, well-estimated normals can boost the effect of many point cloud processing tasks, like denoising and surface reconstruction. To further verify the advantages of our method, we show denoising results using the same point updating algorithm, yet based on the normal results estimated by different methods.

Denote that \( N_i \) is the ball neighborhood of \( p_i \), we here provide an algorithm to update point positions under the guidance of estimated normals, which is able to remove noise and recover sharp features:

\[
P'_i = p_i + y_i \sum_{j \in N_i} (p_j - p_i)w_i(n_i, n_j)n_i^T n_j + \lambda n_j,
\]

(12)

where \( n_i \) is the normal of \( p_i \), \( w_i(n_i, n_j) = \exp\left(\frac{-||n_i - n_j||^2}{\sigma^2}\right) \) is a weight function, \( \lambda = 0.5 \) is a tradeoff parameter and \( y_i \) is the step size set to \( \frac{1}{12} \) by default. To prevent points from accumulating around edges, we keep the neighboring information unchanged in all iterations [42]. The iteration number is set to 20 in our experiment.

We demonstrate the visual quality of denoising results and their reconstruction results (see Fig. 16). Our normal results help produce the most faithful denoising and reconstruction results.

9 CONCLUSION
In this work, we propose a normal refinement network (Refine-Net) to estimate normals for noisy point clouds, by introducing multiple important feature representations into the refinement steps of the input initial normals. Our Refine-Net involves several feature modules which can capture different geometric information in order to jointly contribute to the normal recovery. By a novel connection module, our method effectively handles features from distinct domains and incorporates them into the produced normal feature. In addition to the overall architecture of Refine-Net, we propose a multi-scale fitting patch selection (MFPS) scheme to estimate our own initial normals with better geometric supports.

Refine-Net is a generic normal refinement framework. We show that Refine-Net is able to repair the suboptimal estimations of any initial normal input by recovering sharp edges (preserving details) and removing perturbations (denoising); thus, it can be applied to improve results from other state-of-the-art networks. More feature modules, representing typical geometric properties of the input point clouds, could be potentially developed to further explore the network ability of our framework. Extensive evaluations demonstrate the clear superiority of Refine-Net over the state-of-the-arts on both synthetic and real-scanned datasets. It also shows the promising prospect of applying our method to several downstream geometric tasks in future researches, such as surface reconstruction, consolidation, and semantic segmentation.

ACKNOWLEDGMENTS
The authors thank the anonymous reviewers for their careful reading and valuable comments.

REFERENCES
[1] Y. He, W. Sun, H. Huang, J. Liu, H. Fan, and J. Sun, “PVN3D: A deep point-wise 3D keypoints voting network for 6DoF pose estimation,” in Proc. IEEE/CVF Comput. Vis. Pattern Recognit., 2023, pp. 11 632–11 641.
[2] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao, “PVNet: Pixel-wise voting network for 6DoF pose estimation,” in Proc. IEEE/CVF Comput. Vis. Pattern Recognit., 2019, pp. 4561–4570.
[3] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, “Towards 3D point cloud based object maps for household environments,” Robot. Auton. Syst., vol. 56, no. 11, pp. 927–941, 2008.
[4] M. Liang, B. Yang, S. Wang, and R. Urtasun, “Deep continuous fusion for multi-sensor 3D object detection,” in Proc. Eur. Conf. Comput. Vis., 2018, pp. 641–656.
[5] S. Wang, S. Suo, W.-C. Ma, A. Pokrovsky, and R. Urtasun, “Deep parametric continuous convolutional neural networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 2589–2597.
[6] H. Chen, M. Wei, Y. Sun, X. Xie, and J. Wang, “Multi-patch collaborative point cloud denoising via low-rank recovery with graph constraint,” IEEE Trans. Vis. Comput. Graph., vol. 26, no. 11, pp. 3255–3270, Nov. 2020.
[7] C. Dinesh, G. Cheung, and I. V. Bajić, “Point cloud denoising via feature graph laplacian regularization,” IEEE Trans. Image Process., vol. 29, pp. 4143–4158, 2020.
[8] W. Hu, X. Gao, G. Cheung, and Z. Guo, “Feature graph learning for 3D point cloud denoising,” IEEE Trans. Signal Process., vol. 68, pp. 2841–2856, 2020.
[9] P. Hermosilla, T. Ritschel, and T. Ropinski, “Total denoising: Unsupervised learning of 3D point cloud cleaning,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2019, pp. 52–60.
[10] S. Fleishman, D. Cohen-Or, and C. T. Silva, “Robust moving least-squares fitting with sharp features,” ACM Trans. Graph., vol. 24, no. 3, pp. 544–552, 2005.
[11] W. Xie, M. Wang, M. Wei, J. Jiang, and J. Qin, “Surface reconstruction from normals: A robust DGP-based discontinuity preservation approach,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2019, pp. 5328–5336.
[12] E. Che and M. J. Olson, “Multi-scan segmentation of terrestrial laser scanning data based on normal variation analysis,” J. Photogramm. Remote Sens., vol. 143, pp. 233–248, 2018.
[13] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, “Dynamic graph CNN for learning on point clouds,” ACM Trans. Graph., vol. 38, no. 5, pp. 1–12, 2019.
[14] J. Li, B. M. Chen, and G. H. Lee, “SO-Net: Self-organizing network for point cloud analysis,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 9397–9406.
[15] S. Te, W. Hu, A. Zheng, and Z. Guo, “RGCNN: Regularized graph CNN for point cloud segmentation,” in Proc. 26th ACM Int. Conf. Multimedia, 2018, pp. 746–754.
[16] Y. Qian, J. Hou, S. Kwong, and Y. Hu, “PUGeoNet: A geometrically-perturbed graph network for 3D point cloud upsampling,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 752–769.
[17] J. Zhang, J. Cao, X. Liu, H. Chen, B. Li, and L. Liu, “Multi-normal estimation via pair consistency voting,” IEEE Trans. Vis. Comput. Graphics, vol. 25, no. 4, pp. 1693–1706, Apr. 2019.
[18] J. Zhang, J. Cao, X. Liu, J. Wang, J. Liu, and X. Shi, “Point cloud normal estimation via low-rank subspace clustering,” Comput. Graph., vol. 37, no. 6, pp. 697–706, 2013.
[19] A. Boulch and R. Marlet, “Deep learning for robust normal estimation in unstructured point clouds,” Comput. Graph. Forum, vol. 35, no. 5, 2016, pp. 281–290.
[20] Y. Ben-Shabat, M. Lindenbaum, and A. Fischer, “Nesti-net: Normal estimation for unstructured 3D point clouds using convolutional neural networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2019, pp. 10 112–10 120.
[22] P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra, “PCPNet: Learning local shape properties from raw point clouds,” Comput. Graph. Forum, vol. 37, no. 2, pp. 75–85, 2018.
[23] P.-S. Wang, Y. Liu, and X. Tong, “Mesh denoising via cascaded normal regression,” ACM Trans. Graph., vol. 35, no. 6, pp. 232–1, 2016.
[24] H. Zhou et al., “Geometry and learning co-supported normal estimation for unorganized point clouds,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 13238–13247.
[25] A. Serna, B. Marcotegui, F. Goulette, and J.-E. Deschaud, “Paris-Rue-Madame database: A 3D mobile laser scanner dataset for benchmarking urban detection, segmentation and classification methods,” in Proc. 4th Int. Conf. Pattern Recognit. Appl. Methods, 2014, pp. 1–4.
[26] H. Hoppe, T. DeRose, T. Duchamp, J. A. McDonald, and W. Stuetzle, “Surface reconstruction from unorganized points,” in Proc. 19th Annu. Conf. Comput. Graph. Interactive Techn., 1992, pp. 71–78.
[27] N. J. Mitra and A. T. Nguyen, “Estimating surface normals in noisy point cloud data,” in Proc. 19th ACM Symp. Comput. Geom., 2003, pp. 322–328.
[28] F. Cazals and M. Pouget, “Estimating differential quantities using point-clouds,” Comput. Aided Geometric Des., vol. 22, no. 2, pp. 121–146, 2005.
[29] G. Guennebaud and M. H. Gross, “Algebraic point set surfaces,” ACM Trans. Graph., vol. 26, no. 3, p. 23, 2007.
[30] N. Amenta and M. Bern, “Surface reconstruction by voronoi filtering,” Discrete Comput. Geom., vol. 22, no. 4, pp. 481–504, 1999.
[31] T. K. Dey and S. Goswami, “Provable surface reconstruction from noisy samples,” Comput. Geometry, vol. 35, no. 1/2, pp. 124–141, 2006.
[32] P. Alliez, D. Cohen-Steiner, Y. Tong, and M. Desbrun, “Voronoi-based variational reconstruction of unoriented point sets,” in Proc. 5th Eurographics Symp. Geometry Process., 2007, pp. 39–48.
[33] Q. Mérigot, M. Ovsjanikov, and L. J. Guibas, “Voronoï-based curvature and feature estimation from point clouds,” IEEE Trans. Vis. Comput. Graph., vol. 17, no. 6, pp. 743–756, Jun. 2011.
[34] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, “Point set surfaces,” in Proc. Conf. Visualization, 2001, pp. 21–29.
[35] A. C. Öztişireli, G. Guennebaud, and M. Gross, “Feature preserving point set surfaces based on non-linear kernel regression,” Comput. Graph. Forum, vol. 28, no. 2, pp. 493–501, 2009.
[36] B. Li, R. Schnabel, R. Klein, Z. Cheng, G. Dang, and S. Jin, “Robust normal estimation or point clouds with sharp features,” Comput. Graph., vol. 34, no. 2, pp. 94–106, 2010.
[37] Y. Wang, H.-Y. Feng, F.-E. Delorme, and S. Engin, “An adaptive normal estimation method for scanned point clouds with sharp features,” Comput.-Aided Des., vol. 45, no. 11, pp. 1333–1348, 2013.
[38] H. Avron, A. Sharf, C. Greif, and D. Cohen-Or, “L1-sparse reconstruction of sharp point set surfaces,” ACM Trans. Graph., vol. 29, no. 5, 2010, Art. no. 135.
[39] Y. Sun, S. Schaefer, and W. Wang, “Denoising point sets via I0 minimization,” Comput. Aided Geometric Des., vol. 35, pp. 2–15, 2015.
[40] H. Chen et al., “Structure-guided shape-preserving mesh texture smoothing via joint low-rank matrix recovery,” Comput.-Aided Des., vol. 115, pp. 122–134, 2019.
[41] M. Wei, J. Huang, X. Xie, L. Liu, J. Wang, and J. Qin, “Mesh denoising guided by patch normal co-filtering via kernel low-rank recovery,” IEEE Trans. Vis. Comput. Graphics, vol. 25, no. 10, pp. 2910–2926, Oct. 2019.
[42] X. Lu, S. Schaefer, J. Luo, L. Ma, and Y. He, “Low rank matrix approximation for 3D geometry filtering,” IEEE Trans. Vis. Comput. Graphics, early access, Oct. 01, 2020, doi: 10.1109/TVCG.2020.3026785.
[43] Z. Li et al., “NormalFit-Net: Normal filtering neural network for feature-preserving mesh denoising,” Comput. Aided Des., vol. 127, 2020, Art. no. 102861.
[44] A. Bouch and R. Marlet, “Fast and robust normal estimation for point clouds with sharp features,” Comput. Graph. Forum, vol. 31, no. 5, pp. 1765–1774, 2012.
[45] R. Kovesi, A. C. Öztişireli, I. Pandele, and M. H. Gross, “PointProNets: Consolidation of point clouds with convolutional neural networks,” Comput. Graph. Forum, vol. 37, no. 2, pp. 87–99, 2018.
[46] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “PointNet: Deep learning on point sets for 3D classification and segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 77–85.
Mingqiang Wei (Senior Member, IEEE) received the PhD degree in computer science and engineering from the Chinese University of Hong Kong (CUHK), in 2014. He is currently a professor with the School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics (NUAA). Before joining NUAA, he served as an assistant professor with the Hefei University of Technology, and a postdoctoral fellow with CUHK. He was a recipient of the CUHK Young Scholar Thesis awards in 2014. He is currently an associate editor for the Visual Computer Journal, Journal of Electronic Imaging, and a guest editor for IEEE Transactions on Multimedia. His research interests include 3D vision, computer graphics, and deep learning.

Xiaoping Zhang (Fellow, IEEE) received BS and PhD degrees in electronic engineering from the Tsinghua University, in 1992 and 1996, respectively. He holds an MBA in Finance, Economics and Entrepreneurship with Honors from the University of Chicago Booth School of Business, Chicago, IL. Since Fall 2000, he has been with the Department of Electrical and Computer Engineering, Ryerson University, Toronto, ON, Canada, where he is currently a professor and the director of the Communication and Signal Processing Applications Laboratory. He has served as the program director of Graduate Studies. He is cross-appointed to the Finance Department at the Ted Rogers School of Management, Ryerson University. He was a visiting scientist with the Research Laboratory of Electronics, Massachusetts Institute of Technology, and cross-appointed to the Finance Department at the Ted Rogers School of Management, Ryerson University. His research interests include sensor networks and IoT, machine learning, statistical signal processing, image and multimedia content analysis, and applications in Big Data, finance, and marketing.

Haoran Xie (Senior Member, IEEE) received the PhD degree in computer science from the City University of Hong Kong, in 2014. He is currently an associate professor with the Department of Computing and Decision Sciences, Lingnan University, Hong Kong. His research interests include artificial intelligence, Big Data, and educational technology. He has published 240 research publications, including 114 journal articles. Among all 114 journal articles, there are 89 SCI/SSCI indexed and 12 SCOPUS indexed. He has obtained 14 research awards, including five best paper awards from WI 2020, ICBL 2020, DASFAA 2017, ICBL 2016 and SECOP 2015, the Golden Medal and the Special Award from International Innovation Innovation Competition in Canada and so on. He is the editor-in-chief of Computers & Education: Artificial Intelligence, associate editors of Array Journal, Australasian Journal of Educational Technology, Advances in Computational Intelligence, and International Journal of Mobile Learning and Organisation. He has successfully obtained more than 50 research grants; the total amount of these grants is more than HK $27 million. He is the senior member of ACM, and the Life Member of AAAI.

Jing Qin is currently an associate professor with the School of Nursing, The Hong Kong Polytechnic University and is a key member in the Centre for Smart Health. His research interests include creatively leveraging advanced virtual reality (VR) and artificial intelligence (AI) techniques in healthcare and medicine areas and his achievements in relevant areas has been well recognized by the academic community. He won the Hong Kong Medical and Health Device Industries Association Student Research Award for his PhD study on VR-based simulation systems for surgical training and planning. He won three Best Paper Awards for his research on AI-driven medical image analysis and computer-assisted surgery, including one of the most prestigious awards in this field: MIA-MICCAI best paper award in 2017. He served as a local organization chair for MICCAI 2019, technical program committee (TPC) members for many academic conferences, speakers for many invited talks, and referees for many prestigious journals in relevant fields.

Jun Wang received the bachelor’s and PhD degrees in computer-aided design from the Nanjing University of Aeronautics and Astronautics, in 2002 and 2007, respectively. He is currently a professor with the Nanjing University of Aeronautics and Astronautics (NUAA), China. He conducted research as a postdoctoral scholar with the University of Wisconsin, Madison from 2007 to 2010. From 2012 to 2013, he worked as a senior research engineer with Leica Geosystems Inc., USA. In 2013, he paid an academic visit to the Department of Mathematics, Harvard University. His research interests include geometry processing and geometric modeling, especially large-scale LiDAR point data capturing, management, processing and analysis.

Jing Qin

Tong Lu received the BSc degree from the Nanjing University, in 1997, and the PhD degree in computer science from Nanjing University, in 2005. He served as associate professor and assistant professor with the Department of Computer Science and Technology, Nanjing University from 2007 and 2005. He is currently a full professor with the Nanjing University. He also has served as visiting scholar with the National University of Singapore and Department of Computer Science and Engineering, Hong Kong University of Science and Technology, respectively. He is also a member of the National Key Laboratory of Novel Software Technology in China. He has published more than 130 papers and authored two books in his area of interest, and issued more than 20 international or Chinese invention patents. His current interests are in the areas of multimedia, computer vision and pattern recognition algorithms/systems.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.