EC-Net: an Edge-aware Point set Consolidation Network

Lequan Yu[1,3⋆][0000−0002−9315−6527], Xianzhi Li[1⋆], Chi-Wing Fu[1,3],
Daniel Cohen-Or[2], Pheng-Ann Heng[1,3]

[1]The Chinese University of Hong Kong [2]Tel Aviv University
[3]Shenzhen Key Laboratory of Virtual Reality and Human Interaction Technology,
Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, China
{lqyu,xzli,cwfu,pheng}@cse.cuhk.edu.hk dcor@mail.tau.ac.il

Abstract. Point clouds obtained from 3D scans are typically sparse, irregular, and noisy, and required to be consolidated. In this paper, we present the first deep learning based *edge-aware* technique to facilitate the consolidation of point clouds. We design our network to process points grouped in local patches, and train it to learn and help consolidate points, deliberately for edges. To achieve this, we formulate a regression component to simultaneously recover 3D point coordinates and point-to-edge distances from upsampled features, and an edge-aware joint loss function to directly minimize distances from output points to 3D meshes and to edges. Compared with previous neural network based works, our consolidation is *edge-aware*. During the synthesis, our network can attend to the detected sharp edges and enable more accurate 3D reconstructions. Also, we trained our network on virtual scanned point clouds, demonstrated the performance of our method on both synthetic and real point clouds, presented various surface reconstruction results, and showed how our method outperforms the state-of-the-arts.

Keywords: point cloud, learning, neural network, edge-aware

1 Introduction

Point cloud consolidation is *a process of "massaging" a point set into a surface* [1], for enhancing the surface reconstruction quality. In the past two decades, a wide range of techniques have been developed to address this problem, including denoising, completion, resampling, and many more. However, these techniques are mostly based on *priors*, such as piecewise smoothness. Priors are typically over-simplified models of the actual geometry behavior, thus the prior-based techniques tend to work well for specific class of models rather than being general.

To implicitly model and characterize the geometry behavior, one common way is to take a data-driven approach and model the complex behavior using explicit

⋆ indicates equal contributions.

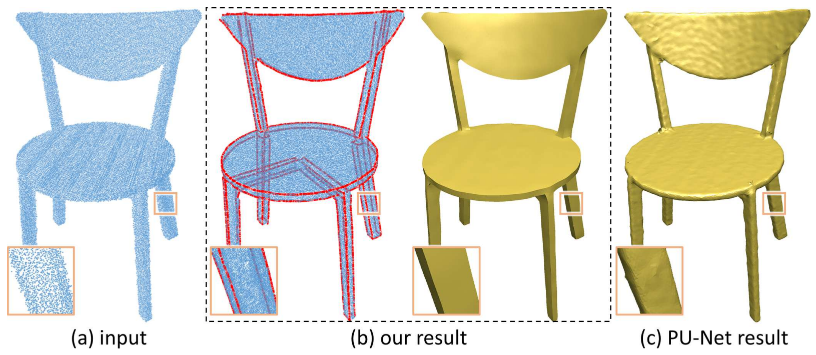

Fig. 1: Given a point cloud (a) with noisy samples in inhomogeneous distribution, our method consolidates it and reconstructs a plausible surface (b). Compared with PU-Net (c), our method is edge-aware and can preserve sharp features.

examples. Data-driven surface reconstruction techniques [2–5] are based on matching local portions (often denoted as patches) to a set of examples. Particularly, the emergence of neural networks and their startling performance provide a new means for 3D reconstruction from point sets by data-driven learning [6–8]. One of the main limitations of these neural network based methods is that they are oblivious to sharp features on 3D objects, where undersampling problems are typically more severe, making it challenging for an accurate object reconstruction.

In this paper, we present the first *edge-aware consolidation network*, namely EC-Net, for point cloud consolidation. The network is designed and trained, such that the output points admit to the surface characteristic of the 3D objects in the training set. More importantly, our method is *edge-aware*, in the sense that the network learns the geometry of edges from the training set, and during the test time, it identifies edge points and generates more points along the edges (and over the surface) to facilitate a 3D reconstruction that preserves sharp features.

Generally speaking, scanned point sets are irregular and non-uniform, and thus, do not lend themselves to be learned by common convolutional neural networks (CNN). Inspired by PointNet [9], we directly process 3D points by converting their coordinates into deep features and producing more points by feature expansion [7]. Then, for efficient learning of the edges, we design our network to process points grouped as local patches in the point cloud. To do so, we develop a patch extraction scheme that solely works on points, so that we can extract patches of points for use consistently in both training and testing phases.

In addition, to train the network to be edge-aware, we associate edge and mesh triangle information with the training patches, and train the network to learn features from the patches by regressing point-to-edge distances and then the point coordinates. More importantly, we design a novel edge-ware joint loss function that can be efficiently computed for directly comparison between the output points and ground truth 3D meshes. Our loss function encourages the output points to be located close to the underlying surface and to the edges, as well as distributed more evenly on surface. Then in the inference phase, the network can generate and find output points close to the edges. Since it is difficult to annotate edges directly in real scanned point clouds, we train our network on synthesized virtual scanned point clouds, and show the performance of our method on both real and virtual scanned point clouds. By using our trained network, we show through various experiments that we can improve not only the point cloud consolidation results (see Figures 1(b) & (c)), but also the surface reconstruction quality, compared to various state-of-the-art methods. All the code is available at the project webpage[1].

Related works. Consolidating scanned data and imperfect point clouds has been an active research area since the early 90's [10–12]. We briefly review some traditional geometric works and then discuss some recent related works that employ neural networks. For a more comprehensive survey, please refer to [13].

Point cloud consolidation. Early works in this area assumed smooth surface [1, 14, 15]. In [14], the parameterization-free local projection operator (LOP) was devised to enhance the point set quality. However, these methods are oblivious to sharp edges and corners. To consolidate a point set in an edge-aware manner, some methods detected/sensed the sharp edges and arranged points deliberatively along edges to preserve their sharpness [16–19]. Huang *et al.* [20] developed the edge-aware resampling (EAR) algorithm; it computes reliable normals away from edges and then progressively upsamples points towards the surface singularities. Despite its promising results, EAR depends on the accuracy of the given/estimated normal. Preiner *et al.* [21] developed CLOP, a continuous version of the LOP, for fast surface construction using the Gaussian mixture model to describe the point cloud density. To sum up, these geometric approaches either assume strong priors or rely on extra geometric attributes for upsampling point sets.

Neural networks for mesh and point cloud processing. Motivated by the promising results that deep learning methods have achieved for image and video problems, there has been increasing effort to leverage neural networks for geometry and 3D shape problems. To do so, early works extracted low-level geometric features as inputs to CNNs [22, 23]. Other works converted the input triangular meshes or point clouds to regular voxel grids [24–29] for CNN to process. However, pre-extracting low-level features may bring bias, while a volume representation demands a high computational cost and is constrained by its resolution.

Recently, point clouds have drawn more attention, and there are some works to utilize neural networks to directly process point clouds. Qi *et al.* [9] firstly developed the PointNet, a network that takes a set of unordered points in 3D as inputs and learns features for object classification and segmentation. Later, they proposed the PointNet++ to enhance the network with a hierarchical feature learning technique [30]. Subsequently, many other networks have been proposed for high-level analysis problems with point clouds [31–40]. However, they all focus

[1] `https://yulequan.github.io/ec-net/index.html`

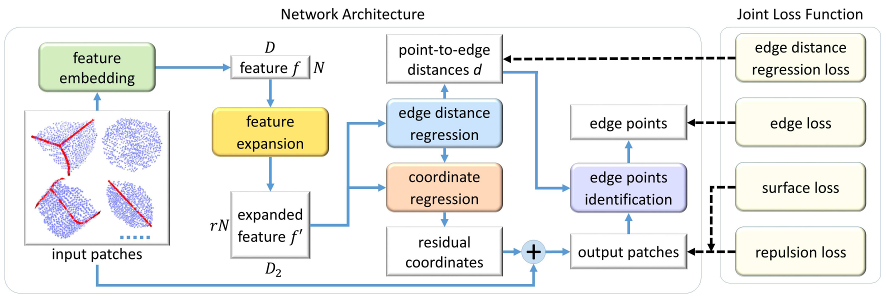

Fig. 2: The pipeline of EC-Net. For each point in an input patch, we first encode its local geometry into a feature vector f (size: $N \times D$) using PointNet++, and expand f into f' (size: $rN \times D_2$) using a feature expansion mechanism. Then, we regress the residual point coordinates and also the point-to-edge distances (d) from the expanded features, and form the output point coordinates by adding the original point coordinates to the residual. Finally, the network identifies points on edges and yields output points. The network was trained with an edge-aware joint loss function that has four terms; see the yellow boxes on the right.

on analyzing global or mid-level attributes of point clouds. In some other aspects, Guerrero *et al.* [6] proposed a network to estimate the local shape properties in point clouds, including normal and curvature. 3D reconstruction from 2D images has also been widely studied [8, 41, 42]. Our work is most related to PU-Net [7], which presented a network to upsample a point set. However, our method is edge-aware, and we extract local patches and train the network to learn edges in patches with a novel edge-aware joint loss function.

2 Method

In this section, we first present the training data preparation (Sec. 2.1) and the EC-Net architecture (Sec. 2.2). Then, we present the edge-aware joint loss function (Sec. 2.3) and the implementation details (Sec. 2.4). Figure 2 shows the pipeline of EC-Net; see the supplemental material for the detailed architecture.

2.1 Training data preparation

We train our network using point clouds synthesized from 3D objects, so that we can have ground truth surface and edge information. To start, we collect 3D meshes from ShapeNet [43] and other online repositories, including simple 3D shapes, mechanical parts, and everyday objects such as chairs. Since we train the network with patches as inputs, we prepare a large amount of patches on the 3D meshes and do not require many meshes. Moreover, we manually sketch polylines on each 3D mesh to annotate sharp edges on the meshes; see Figure 3(a).

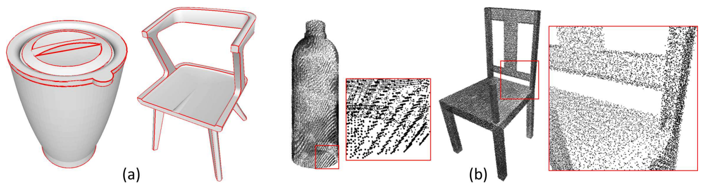

Fig. 3: Example annotated edges (in red) on some of our collected 3D meshes (a). Example point clouds produced from our virtual 3D scans (b). The point density varies and the zoom-in windows also reveal the synthetic noise.

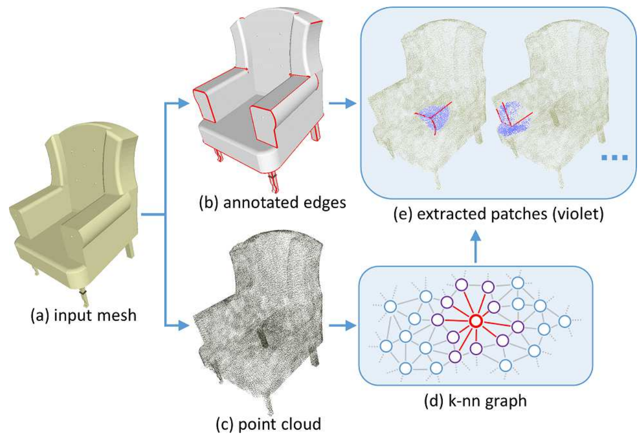

Fig. 4: Procedure to extract patches (a local group of points) from a point cloud; note that (a) and (b) are available only in the training (but not inference) phase.

Virtual scanning. To obtain point clouds from the 3D mesh objects, we use the following virtual scanning procedure. First, we normalize the mesh to fit in $[-1, +1]^3$, and evenly arrange a circle of 30 virtual cameras (50° field of view) horizontally around (and to look at) the object. We then put the cameras two units from the object center and randomly perturb the camera positions slightly upward, downward or sideway. After that, we produce a point cloud for each camera by rendering a depth image of the object, adding quantization noise (see Sec. 3) to the depth values and pixel locations, and backprojecting each foreground pixel to obtain a 3D point sample. Then, we can compose the 3D point clouds from different cameras to obtain a virtual scanned data. Such sampling procedure mimics a real scanner with surface regions closer to the virtual camera receiving more point samples; see Figure 3(b) for two example results.

Patch extraction. From a point cloud (see Figure 4(c)), we aim to extract local groups of points (patches), such that the points in a patch are *geodesically* close to one another over the underlying surface. This is very important, since using Euclidean distances to select points could lead to points on opposite sides of a thin surface, e.g., see the thin plates in the chair shown in Figure 3(b). Compared with [7], our patch extraction procedure directly operates on point clouds, *not* meshes, so we need a *consistent* extraction procedure for *both* network training and inference, where ground truth meshes are not available during the inference.

To this end, we first construct a weighted graph by considering each point as a node and creating an edge from each point to its k-nearest neighboring (k-nn) points, where k=10; see Figure 4(d). The edge weight is set as the Euclidean distance between the two points. Then, we randomly select m=100 points as the patch centroids; from each selected point, we use the Dijkstra algorithm to find the 2048 nearest points in terms of shortest path distances in the graph. Hence, we can find points that are approximately within a geodesic radius from the centroid. Further, we randomly select $\hat{N}$=1024 points out of the 2048 points to introduce randomness into the point distribution, and normalize the 3D coordinates of the points to have zero mean inside a unit ball. For patches used for training, we also find the associated mesh triangles and annotated edge segments near the patches as the ground truth information for training the network; see Figure 4(e).

2.2 Edge-aware Point set Consolidation Network

In this subsection, we present the major components of EC-Net; see Figure 2.

Feature embedding and expansion. This component first maps the neighboring information (raw 3D coordinates of nearby points) around each point into a feature vector using PointNet++ [30] to account for the fact that the input points are irregular and unordered. The output is a D-dimensional multi-scale feature vector for each input point, where D is 256 in our experiments. In this step, we make the following adaptation for our problem. By design, PointNet++ processes a full point cloud of an object, while EC-Net processes local patches. Since patches have open boundary, points near the boundary have neighbors mostly on one of its side only, so we found that the extracted features of these points are less accurate. Hence, out of the $\hat{N}$ feature vectors, we retain the N=$\frac{\hat{N}}{2}$ feature vectors (denoted as f) corresponding to points closer to the patch centroid. Next, the component synthesizes points by expanding features directly in feature space using the feature expansion module in [7], since points and features should be interchangeable. After this module, feature f (dimension: $N \times D$) are expanded to be f' (dimension: $rN \times D_2$), where r is the upsampling rate and D_2 is the new feature dimension, which is set as 128; see again Figure 2.

Edge distance regression. This component regresses a point-to-edge distance for each expanded feature (or point, equivalently) later for edge points identification. The regressed distance is an estimated shortest distance from the output point to the nearest annotated edge segment among all annotated edge segments associated with the patch. To do this, we extract a distance feature f_{dist} from the expanded feature f' via a fully connected layer, and then regress the point-to-edge distance d from f_{dist} via another fully connected layer. We do this in two steps, so that we can feed f_{dist} also to the coordinate regression component.

Coordinate regression. This component reconstructs the 3D coordinates of the output points; see Figure 2. First, we concatenate the expanded feature f' with the distance feature f_{dist} (from previous component) to form another feature, since f_{dist} contains certain point-to-edge distance information. Then, we regress the point coordinates from the concatenated feature by applying two fully connected layers. Note that we only regress the residual 3D coordinates, and the network output 3D coordinates of the output points by adding the original 3D coordinates of the input points to the regressed residual 3D coordinates.

Edge points identification. Denoting d_i as the regressed point-to-edge distance of output point x_i, we next find a subset of output points, namely *edge points* (denoted as $\mathcal{S}_{\Delta_d}$ with threshold Δ_d) that are near the edges: $\mathcal{S}_{\Delta_d} = \{x_i\}_{d_i < \Delta_d}$. Note that this component is performed in both training and inference phases.

2.3 Edge-aware joint loss function

The loss function should encourage the output points to be (i) located close to the underlying object surface, (ii) edge-aware (located close to the annotated edges), and (iii) more evenly distributed on the object surface. To this end, we guide the network's behavior by designing an *edge-aware joint loss function* with the following four loss terms (see also the rightmost part of Figure 2):

Surface loss encourages the output points to be located close to the underlying surface. When extracting each training patch from the input point clouds, we find triangles and edge segments associated with the patch; see Figure 4. Hence, we can define surface loss using the minimum shortest distance from each output point x_i to all the mesh triangles T associated with the patch: $d_T(x_i, T) = \min_{t \in T}\ d_t(x_i, t)$, where $d_t(x_i, t)$ is the shortest distance from x_i to triangle $t \in T$. It is worth noting that to compute d_t in 3D, we need to consider seven cases, since the point on t that is the closest to x_i may be located at triangle vertices, along triangle edges, or within the triangle face. Experimentally, we found that the algorithm [44] for calculating d_t can be implemented using TensorFlow to automatically calculate the gradients when training the network. With d_T computed for all the output points, we can sum them up to compute the surface loss:

$$L_{surf} = \frac{1}{\tilde{N}} \sum_{1 \le i \le \tilde{N}} d_T^2(x_i, T) \ , \tag{1}$$

where $\tilde{N} = rN$ is the number of output points in each patch.

Edge loss encourages the output points to be edge-aware, i.e., located close to the edges. Denoting E as the set of annotated edge segments associated with a patch, we define edge loss using the minimum shortest distance from each *edge point* to all the edge segments in the patch: $d_E(x_i, E) = \min_{e \in E}\ d_e(x_i, e)$, where $d_e(x_i, e)$ is the shortest distance from edge point x_i to any point on edge segment $e \in E$. Again, we implement the algorithm in [45] to calculate d_e for different shortest distance cases using TensorFlow to automatically calculate the gradients. Then, we sum up d_E for all the edge points and obtain the edge loss:

$$L_{edge} \;=\; \frac{\sum_{x_i \in \mathcal{S}_{\Delta_d}} d_E^2(x_i, E)}{|\mathcal{S}_{\Delta_d}|} \;, \quad \text{where } \mathcal{S}_{\Delta_d} \text{ is the edge point set .} \tag{2}$$

Repulsion loss encourages the output points to be more evenly distributed over the underlying surface. Given a set of output points $x_i, i = 1...\tilde{N}$, it is defined as

$$L_{repl} \;=\; \frac{1}{\tilde{N} \cdot K} \sum_{1 \le i \le \tilde{N}} \; \sum_{i' \in \mathcal{K}(i)} \eta(\; || \; x_{i'} \; - \; x_i \; || \;) \;, \tag{3}$$

where $\mathcal{K}(i)$ is the set of indices for the K-nearest neighborhood of x_i (we set K=4), $|| \cdot ||$ is the L2-norm, and $\eta(r) = max(0, h^2 - r^2)$ is a function to penalize x_i if it is too close to some other nearby points, where h is empirically set as 0.03 (which is the mean separation distance between points estimated from the number of points and bounding box diagonal length according to [20]). It is worth noting that we only want to penalize x_i when it is too close to some neighborhood points, so we only consider a few nearest neighboring points around x_i; moreover, we remove the repulsion effect when the point-to-point distance is above h.

Edge distance regression loss aims to guide the network to regress the point-to-edge distances d for the rN output points; see Figure 2. Considering that it is difficult for the network to regress d, since not all output points are actually close to the annotated edges. Hence, we design a truncated regression loss:

$$L_{regr} \;=\; \frac{1}{\tilde{N}} \sum_{1 \le i \le \tilde{N}} \left[\; \mathcal{T}_b(d_E(x_i, E)) \; - \; \mathcal{T}_b(d_i) \; \right]^2 \;, \tag{4}$$

where $\mathcal{T}_b(x) = max(0, min(x, b))$ is a piecewise linear function with parameter b. Empirically, we found the network training not sensitive to b, and set it as 0.5.

End-to-end training. When training the network, we minimize the combined edge-aware joint loss function below with balancing weights α and β:

$$\mathcal{L} \;=\; L_{surf} \;+\; L_{repl} \;+\; \alpha L_{edge} \;+\; \beta L_{regr} \;. \tag{5}$$

In our implementation, we set α and β as 0.1 and 0.01, respectively.

2.4 Implementation Details

Network training. Before the training, each input patch is normalized to fit in $[-1, 1]^3$. Then, we augment each patch on-the-fly in the network via a series of operators: a random rotation, a random translation in all dimensions by -0.2 to 0.2, a random scaling by 0.8 to 1.2, adding Gaussian noise to the patch points with σ set as 0.5% of the patch bounding box size, and randomly shuffling the ordering of points in the patch. We implement our method using TensorFlow, and train the network for 200 epochs using the Adam [46] optimizer with a minibatch size of 12 and a learning rate of 0.001. In our experiments, the default upsampling rate r is set as 4. For threshold $\varDelta_d$, we empirically set it as 0.15, since it is not sensitive to slight variation in our experiments. Overall, it took around 5 hours to train the network on an NVidia TITAN Xp GPU.

Network inference. We apply a trained network to process point clouds also in a patch-wise manner. To do so, we first find a subset of points in a test point cloud, and take them as centroids to extract patches of points using the procedure in Sec. 2.1. For the patches to distribute more evenly over the point cloud (say with N_{pt} points), we use farthest point sampling to randomly find $M = \beta\frac{N_{pt}}{N}$ points in the test point cloud with parameter β, which is empirically set as three. Hence, each point in the point cloud should appear roughly in β different patches on average. After extracting the patches, we feed them into the network and apply the network to produce 3D coordinates and point-to-edge distances, as well as to identify edge points (see Sec. 2.3). Unlike the training phase, we set a smaller $\varDelta_d$, which is 0.05. We use a larger $\varDelta_d$ in the training because training is an optimization process, where we want the network to consider more points to learn to identify the points near the edges.

Surface reconstruction. First, we build a k-nn graph for the output points from network. Then, we filter edge points by fitting line segments using RANSAC, and filter surface points (not near edges points) by finding small groups of nearby points in the k-nn graph in an edge-stopping manner and fitting planes using PCA. Edge stopping means we stop the breath-first growth at edge points; this avoids including irrelevant points beyond the edges. These steps are iterated several times. Lastly, we fill the tiny gap between edge and surface points by including some original points in the gap, and by applying dart throwing to add new points. To further reconstruct the surface, we follow the procedure in EAR [20] to downsample the point set and compute normals, use ball pivoting [47] or screened Poisson surface reconstruction [48] to reconstruct the surface, and use a bilateral normal filtering [49] to clean the resulting mesh.

3 Experiments

Dataset overview. Since most models in [7] are manifolds without sharp edges, we collected 24 CAD models and 12 everyday objects as our training data set, and manually annotate sharp edges on them; see supplemental material. Then, we randomly crop 2,400 patches from the models (see Figure 2) to train our network; see the procedure in Sec. 2.1. To perform the experiments presented in this section, we do not reuse the models in the training data set but download additional 3D models from ShapeNet [43]. For each testing model, we also use the procedure in Sec. 2.1 to generate the virtual scanned point clouds as input.

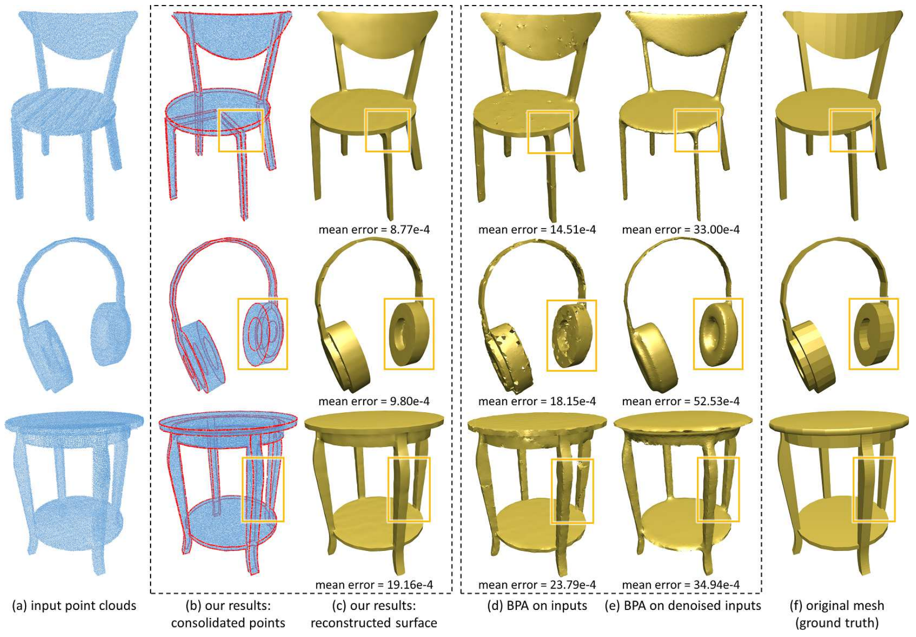

Fig. 5: Our method produces consolidated point clouds (b) that lead to higher quality surface reconstruction results (c). Predicted edge points are shown in red.

Surface reconstruction results. We demonstrate the quality of our method by applying it to consolidate point sets and reconstruct surfaces. Figure 5(a) shows the input point clouds generated from the testing models shown in Figure 5(f), while Figure 5(b) & (c) show our results (see supplemental material for additional reconstruction results). To reconstruct the surfaces, we follow the procedure in Sec. 2.4 and employ the ball pivoting algorithm (BPA) [47] to reconstruct the surfaces. To show the effect of our network, we apply the same procedure with BPA to reconstruct surfaces directly from (i) the input point clouds (see Figure 5(d)), and from (ii) the input point clouds with PCA plane fitting for denoising (see Figure 5(e)), without using our network to process the point clouds.

Comparing the resulting reconstructed surfaces with the ground truth meshes shown in Figure 5(f), we can see that our method achieves the best visual quality, particularly on preserving the edges. In addition, we compute the mean Hausdorff distance between the reconstructed surfaces and their corresponding ground truth surfaces; see the mean errors shown in Figure 5(c), (d) & (e); our method consistently has the lowest mean errors among all the testing models. These quantitative results, together with the visual comparison, give evidence that our method produces consolidated point sets and improves the surface reconstruction quality. Note that without the knowledge of edges learnt and recognized by the network, using PCA alone to denoise point clouds is not edge-aware; the sharp edges would be wiped away, if we directly apply PCA to the raw point cloud,

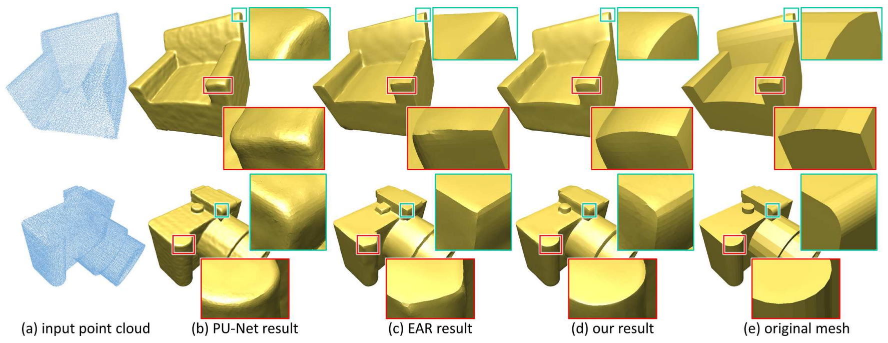

Fig. 6: Comparing surface reconstruction results produced by using PU-Net (b), EAR (c), and our method (d) to consolidate points with the original meshes (e).

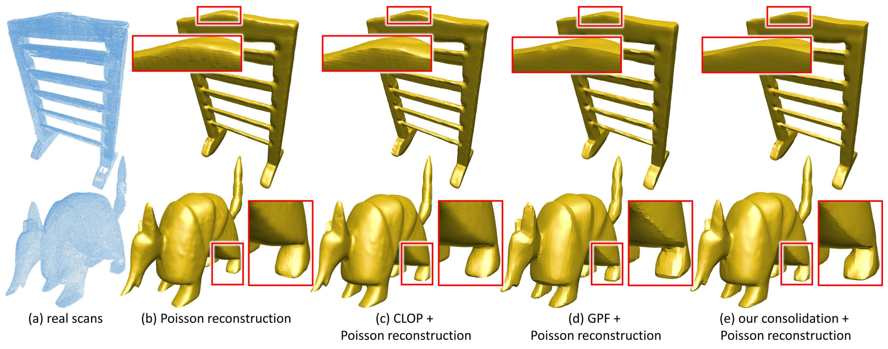
(a) real scans
(b) Poisson reconstruction
(c) CLOP + Poisson reconstruction
(d) GPF + Poisson reconstruction
(e) our consolidation + Poisson reconstruction

Fig. 7: Comparing reconstruction results on real scans produced by direct reconstruction (b) and with consolidation: CLOP, GPF, and our method (c-e).

leading to the inferior surface reconstructions shown in Figure 5(e). To sum up, our consolidation facilitates high-quality reconstruction not only on sharp objects but also on usual smooth objects. It is worth to note that with our consolidation, the overall reconstruction quality also improves over the state-of-the-art surface reconstruction methods on the benchmark models in [50]; due to page limit, please see the supplemental material for more details.

Comparing with other methods. In the experiment, we compare our method with state-of-the-art methods, EAR [20], CLOP [21], GPF [51], and PU-Net [7], by applying them to process and consolidate point clouds before the screened Poisson surface reconstruction [48]. As for PU-Net, we train a new model using our training objects and code released by the author. For better comparison, we also apply patch-based manner in testing phase, as this can achieve better results. Figures 6(b), (c) & (d) present the comparison with EAR and PU-Net. We can see from the results that the sharp edges in the original mesh are mostly smoothed out if we take the point clouds from PU-Net for surface construction. EAR better preserves the sharp features on the reconstructed surfaces, but in

Table 1: Quantitative comparison: our method, PU-Net, and EAR.

Model	Mean (10^{-3})			RMS (10^{-3})		
	Our	PU-Net	EAR	Our	PU-Net	EAR
Camera	**1.31**	1.91	2.43	**1.99**	2.74	3.75
Sofa	1.72	3.20	**1.56**	**2.40**	4.68	2.87
Chair	**0.88**	1.70	1.93	**1.27**	2.50	3.54
Fandisk	**1.09**	2.86	2.33	**1.77**	4.50	5.63
Switch-pot	**1.36**	2.00	1.76	**1.83**	3.07	2.44
Headphone	**0.81**	1.83	3.71	**1.19**	2.89	6.93
Table	**1.15**	2.14	2.74	**1.62**	3.12	5.34
Monitor	**0.61**	2.01	2.58	**0.97**	2.71	5.73

case of severe noise or under-sampling, it may over-smooth the geometric details and sharp features, or over-sharpen the edges. It is because the method depends on the quality of the estimated normals; see the limitation paragraph in [20]. Our method, which is based on an edge-aware neural network model, is able to learn and capture edge information with high learning generalization, our point cloud consolidation results can lead to surface reconstructions that are closer to the original meshes. Figure 7 also demonstrates that our method helps enhance the Poisson reconstruction quality on real scans in terms of preserving sharp edges compared with CLOP [21] and GPF [51].

We also quantitatively compare with EAR and PU-Net by calculating the minimum distances from output points to the associated original mesh (as ground truth) in the test dataset. Table 1 lists the mean and root mean square (RMS) values of different methods on the test models; see supplemental material for visual comparison. We can see from the table that points generated from our method are closer to the original meshes compared to others. The PU-Net uses the EMD loss to encourage output points to be located on the underlying surfaces, so this comparison shows that the results are sub-optimal compared with our method, which directly minimizes the distances from output points to surfaces.

Our results under varying quantization noise. In real scans, the acquired depth values are generally quantized nonlinearly depending on the distance from the scanner. In short, objects far from the scanner would have less precise depth values in fewer quantization levels. During the virtual scanning process, we mimic real scans by quantizing the depth values in N_q levels. This means that there are only N_q unique depth values over all the foreground pixels. In this subsection, we investigate the robustness of our method under different amount of quantization noise by producing three sets of point cloud data using virtual scan on a testing model with N_q=120 (low noise), N_q=80 (medium noise), and N_q=50 (high noise) and then applying our method to consolidate the resulting point clouds.

Figure 8 presents the results, where the left, middle and right columns correspond to low, medium and high noise levels, respectively. From the blown-up views, we

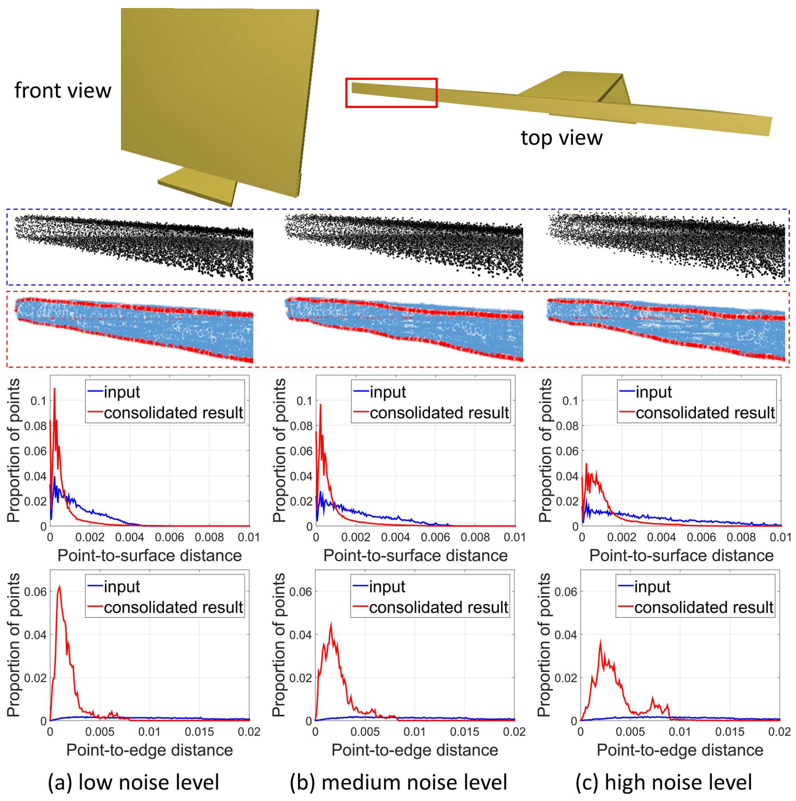

Fig. 8: Results of using our method to consolidate point clouds with different amount of quantization noise. Top row: the testing Monitor model; 2nd and 3rd rows: blown-up views of the input point clouds with different noise level (low to high) and our consolidated results, respectively; 4th and 5th rows: statistics of point-to-surface and point-to-edge distances, respectively.

can see more points above the object surface for the point cloud with high noise. The statistics about the point-to-surface distances also reveal the noise level; see the blue plots in the 4th row. After the consolidation, the points are steered to the left (see the red plots in 4th row), meaning that the points are now all closer to the ground truth surface under different noise levels. Similar pattern can also be observed from the statistical results for the point-to-edge distances.

Results on real scans. We also apply our method to point clouds produced from real scans downloaded from Aim@Shape and obtained from the EAR project [20]. Figure 7 has shown some results on real scans, and Figure 9 shows more consolidation and reconstruction results. As we can see, real scan point clouds are often noisy and have inhomogeneous point distribution. Comparing with the input point clouds, our method is still able to generate more points near the edges and on the surface, while better preserving the sharp features.

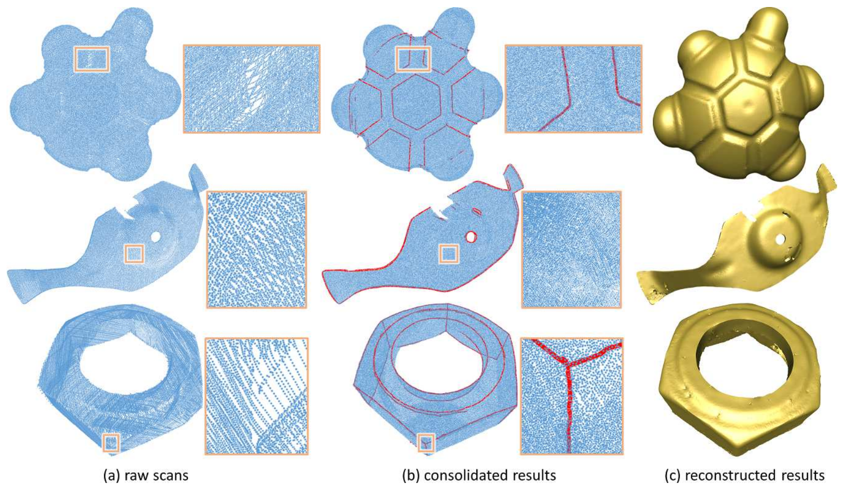

Fig. 9: Three results from real scans (a). Note the diversity of the geometry, the sparseness of the inputs (a), and how well the network locates the edges (in red); see (b). The reconstruction results (c) are yet imperfect due to tiny regions that are severely undersampled (see the blown-up views in (a)).

4 Discussion and future works

We presented EC-Net, the first edge-aware network for consolidating point clouds. The network was trained on synthetic data and tested on both virtual and real data. To be edge-aware, we design a joint loss function to identify points along edges, and to encourage points to be located close to the underlying surface and edges, as well as being more evenly distributed over surface. We compared our method with various state-of-the-art methods for point cloud consolidation, showing improvement in the 3D reconstruction quality, especially at sharp features. Our method has a number of limitations. First, it is not designed for completion, so filling large holes and missing parts would be a separate and different problem. In future, we plan to investigate how to enhance the network and the training process for point cloud completion. For tiny structures that are severely undersampled, our network may not be able to reconstruct the sharp edges around them. With insufficient points, the patch could become too large compared to the tiny structure. Moreover, our current implementation considers patches with a fixed number of points, and it cannot adapt structures of varying scales. It would be an interesting problem to try to dynamically control the patch size and explore the use of larger context in the training and inference.

Acknowledgments. We thank anonymous reviewers for the comments and suggestions. The work is supported by the Research Grants Council of the Hong Kong Special Administrative Region (Project no. GRF 14225616), the Shenzhen Science and Technology Program (No. JCYJ20170413162617606 and No. JCYJ20160429190300857), and the CUHK strategic recruitment fund.

References

1. Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., Silva, C.T.: Computing and rendering point set surfaces. IEEE Trans. Vis. & Comp. Graphics **9**(1) (2003) 3–15
2. Gal, R., Shamir, A., Hassner, T., Pauly, M., Cohen-Or, D.: Surface reconstruction using local shape priors. In: Symposium on Geometry Processing. Number EPFL-CONF-149318 (2007) 253–262
3. Sung, M., Kim, V.G., Angst, R., Guibas, L.J.: Data-driven structural priors for shape completion. ACM Trans. on Graphics (SIGGRAPH Asia) **34**(6) (2015) 175:1–175:11
4. Xu, K., Kim, V.G., Huang, Q., Kalogerakis, E.: Data-driven shape analysis and processing. Computer Graphics Forum **36**(1) (2017) 101–132
5. Remil, O., Xie, Q., Xie, X., Xu, K., Wang, J.: Surface reconstruction with data-driven exemplar priors. Computer-Aided Design **88** (2017) 31–41
6. Guerrero, P., Kleiman, Y., Ovsjanikov, M., Mitra, N.J.: PCPNet: Learning local shape properties from raw point clouds. arXiv preprint arXiv:1710.04954 (2017)
7. Yu, L., Li, X., Fu, C.W., Cohen-Or, D., Heng, P.A.: PU-Net: Point cloud upsampling network. arXiv preprint arXiv:1801.06761 (2018)
8. Groueix, T., Fisher, M., Kim, V.G., Russell, B.C., Aubry, M.: AtlasNet: A papier-mâché approach to learning 3D surface generation. In: IEEE CVPR. (2018) 216–224
9. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: PointNet: deep learning on point sets for 3D classification and segmentation. In: IEEE CVPR. (2017) 77–85
10. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W.: Surface reconstruction from unorganized points. In: Proc. of SIGGRAPH. (1992) 71–78
11. Turk, G., Levoy, M.: Zippered polygon meshes from range images. In: Proc. of SIGGRAPH. (1994) 311–318
12. Amenta, N., Bern, M., Kamvysselis, M.: A new voronoi-based surface reconstruction algorithm. In: Proc. of SIGGRAPH. (1998) 415–422
13. Berger, M., Tagliasacchi, A., Seversky, L.M., Alliez, P., Guennebaud, G., Levine, J.A., Sharf, A., Silva, C.T.: A survey of surface reconstruction from point clouds. In: Computer Graphics Forum. Volume 36., Wiley Online Library (2017) 301–329
14. Lipman, Y., Cohen-Or, D., Levin, D., Tal-Ezer, H.: Parameterization-free projection for geometry reconstruction. ACM Trans. on Graphics (SIGGRAPH) **26**(3) (2007) 22:1–22:5
15. Huang, H., Li, D., Zhang, H.R., Ascher, U., Cohen-Or, D.: Consolidation of unorganized point clouds for surface reconstruction. ACM Trans. on Graphics (SIGGRAPH Asia) **28**(5) (2009) 176:1–176:8
16. Pauly, M., Keiser, R., Kobbelt, L.P., Gross, M.: Shape modeling with point-sampled geometry. ACM Trans. on Graphics (SIGGRAPH) **22**(3) (2003) 641–650
17. Guennebaud, G., Barthe, L., Paulin, M.: Real-time point cloud refinement. In: Symposium on Point Based Graphics. (2004) 41–48
18. Fleishman, S., Cohen-Or, D., Silva, C.T.: Robust moving least-squares fitting with sharp features. ACM Trans. on Graphics (SIGGRAPH) **24**(3) (2005) 544–552
19. Öztireli, A.C., Guennebaud, G., Gross, M.: Feature preserving point set surfaces based on non-linear kernel regression. Computer Graphics Forum (Eurographics) **28**(2) (2009) 493–501
20. Huang, H., Wu, S., Gong, M., Cohen-Or, D., Ascher, U., Zhang, H.R.: Edge-aware point set resampling. ACM Trans. on Graphics **32**(1) (2013) 9:1–9:12

21. Preiner, R., Mattausch, O., Arikan, M., Pajarola, R., Wimmer, M.: Continuous projection for fast l1 reconstruction. ACM Trans. on Graphics (SIGGRAPH) **33**(4) (2014) 47:1–47:13
22. Guo, K., Zou, D., Chen, X.: 3D mesh labeling via deep convolutional neural networks. ACM Trans. on Graphics **35**(1) (2015) 3:1–3:12
23. Boulch, A., Marlet, R.: Deep learning for robust normal estimation in unstructured point clouds. Computer Graphics Forum (SGP) **35**(5) (2016) 281–290
24. Qi, C.R., Su, H., Niessner, M., Dai, A., Yan, M., Guibas, L.J.: Volumetric and multi-view CNNs for object classification on 3D data. In: IEEE CVPR. (2016) 5648–5656
25. Dai, A., Qi, C.R., Nießner, M.: Shape completion using 3D-encoder-predictor CNNs and shape synthesis. In: IEEE CVPR. (2017) 5868–5877
26. Han, X., Li, Z., Huang, H., Kalogerakis, E., Yu, Y.: High-resolution shape completion using deep neural networks for global structure and local geometry inference. In: IEEE ICCV. (2017) 85–93
27. Wang, P., Liu, Y., Guo, Y., Sun, C., Tong, X.: O-CNN: Octree-based convolutional neural networks for 3D shape analysis. ACM Trans. on Graphics **36**(4) (2017) 72:1–72:11
28. Riegler, G., Ulusoy, A.O., Geiger, A.: OctNet: Learning deep 3D representations at high resolutions. In: IEEE CVPR. (2017) 6620–6629
29. Liu, F., Li, S., Zhang, L., Zhou, C., Ye, R., Wang, Y., Lu, J.: 3DCNN-DQN-RNN: a deep reinforcement learning framework for semantic parsing of large-scale 3D point clouds. In: IEEE ICCV. (2017) 5678–5687
30. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: PointNet++: deep hierarchical feature learning on point sets in a metric space. In: Advances in Neural Information Processing Systems 30. (2017) 5105–5114
31. Hua, B.S., Tran, M.K., Yeung, S.K.: Point-wise convolutional neural network. (2017)
32. Klokov, R., Lempitsky, V.: Escape from cells: deep Kd-Networks for the recognition of 3D point cloud models. In: IEEE ICCV. (2017) 863–872
33. Landrieu, L., Simonovsky, M.: Large-scale point cloud semantic segmentation with superpoint graphs. arXiv preprint arXiv:1711.09869 (2017)
34. Xu, D., Anguelov, D., Jain, A.: PointFusion: Deep sensor fusion for 3D bounding box estimation. arXiv preprint arXiv:1711.10871 (2017)
35. Wang, W., Yu, R., Huang, Q., Neumann, U.: SGPN: Similarity group proposal network for 3D point cloud instance segmentation. arXiv preprint arXiv:1711.08588 (2017)
36. Yang, Y., Feng, C., Shen, Y., Tian, D.: FoldingNet: Interpretable unsupervised learning on 3D point clouds. In: IEEE CVPR. (2018) 206–215
37. Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J.: Frustum PointNets for 3D object detection from RGB-D data. arXiv preprint arXiv:1711.08488 (2017)
38. Li, Y., Bu, R., Sun, M., Chen, B.: PointCNN. arXiv preprint arXiv:1801.07791 (2018)
39. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph CNN for learning on point clouds. arXiv preprint arXiv:1801.07829 (2018)
40. Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M.H., Kautz, J.: SPLATNet: Sparse lattice networks for point cloud processing. In: IEEE CVPR. (2018) 2530–2539
41. Fan, H., Su, H., Guibas, L.J.: A point set generation network for 3D object reconstruction from a single image. In: IEEE CVPR. (2017) 2463–2471

42. Lin, C.H., Kong, C., Lucey, S.: Learning efficient point cloud generation for dense 3D object reconstruction. In: AAAI. (2018) to appear
43. Chang, A.X., Funkhouser, T., Guibas, L.J., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: ShapeNet: an information-rich 3D model repository. arXiv preprint arXiv:1512.03012 (2015)
44. Eberly, D.: Distance between point and triangle in 3D. https://www.geometrictools.com/Documentation/DistancePoint3Triangle3.pdf (1999)
45. Eberly, D.: Distance between point and line, ray, or line segment. https://www.geometrictools.com/Documentation/DistancePointLine.pdf (1999)
46. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
47. Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., Taubin, G.: The ball-pivoting algorithm for surface reconstruction. IEEE Trans. Vis. & Comp. Graphics **5**(4) (1999) 349–359
48. Kazhdan, M., Hoppe, H.: Screened poisson surface reconstruction. ACM Trans. on Graphics **32**(3) (2013) 29:1–29:13
49. Zheng, Y., Fu, H., Au, O.K.C., Tai, C.L.: Bilateral normal filtering for mesh denoising. IEEE Trans. Vis. & Comp. Graphics **17**(10) (2011) 1521–1530
50. Berger, M., Levine, J.A., Nonato, L.G., et al.: A benchmark for surface reconstruction. ACM Trans. on Graphics **32**(2) (2013) 20
51. Lu, X., Wu, S., Chen, H., Yeung, S.K., Chen, W., Zwicker, M.: GPF: GMM-inspired feature-preserving point set filtering. IEEE Trans. Vis. & Comp. Graphics **24**(8) (2018) 2315–2326