Towards Uniform Point Distribution in Feature-preserving Point Cloud Filtering

Shuaijun Chen
Deakin University
Australia

Jinxi Wang
Northwest A&F University
Yangling, China

Wei Pan
South China University of Technology
China

Shang Gao
Deakin University
Australia

Meili Wang
Northwest A&F University
Yangling, China

Xuequan Lu* (corresponding author)
Deakin University, Australia
xuequan.lu@deakin.edu.au

Abstract

While a popular representation of 3D data, point clouds may contain noise and need filtering before use. Existing point cloud filtering methods either cannot preserve sharp features or result in uneven point distributions in the filtered output. To address this problem, this paper introduces a point cloud filtering method that considers both point distribution and feature preservation during filtering. The key idea is to incorporate a repulsion term with a data term in energy minimization. The repulsion term is responsible for the point distribution, while the data term aims to approximate the noisy surfaces while preserving geometric features. This method is capable of handling models with fine-scale features and sharp features. Extensive experiments show that our method quickly yields good results with relatively uniform point distribution.

Key words: point cloud filtering; point distribution; feature preservation

1. Introduction

Researchers have made remarkable progress in point cloud filtering in recent years. Recent methods typically aim to maintain sharp features in the original point cloud while projecting the noisy points to the underlying surfaces. The filtered point cloud data can then be used for upsampling [12], surface reconstruction [13, 26], skeleton learning [21, 22], computer animation [24, 27], etc.

Existing point cloud filtering methods can be divided into traditional and deep learning techniques. In the traditional class, position-based methods [17, 11, 28] obtain good smoothing results, while normal-based methods [26, 25] better maintain sharp edges of models (e.g. CAD models). Some of these methods incorporate repulsion terms to prevent points from aggregating but still leave gaps near the edges of geometric features, which affects reconstruction quality. Deep learning-based approaches [30, 31, 36] require a number of noisy point clouds with ground-truth models for training and often achieve good denoising results through a proper number of iterations. These methods are usually based on local information, and lead to uneven point distribution in the filtered results even in the presence of a repulsion loss term. It is difficult for such methods to handle unevenly distributed and sparsely sampled point clouds since it is difficult to automatically adjust the patch size. Different patch sizes in the point cloud also pose a significant challenge to the learning procedure.

The above analysis motivates us to produce filtered point clouds which preserve sharp features yet have a relatively uniform point distribution. Given a noisy point cloud with normals as input, we first smooth the input normals using bilateral filtering [12]. Principal component analysis (PCA) [10] is used for the initial estimation of normals. Secondly, we update the point positions in a local manner by reformulating an objective function consisting of an edge-aware data term and a repulsion term inspired by [23, 25]. The two aim to preserve geometric features and provide uniform point distribution, respectively. Output with these properties is obtained after a few iterations. We have conducted extensive experiments to compare our approach to various other approaches, including position- and normal-based approaches, both learning and traditional. The results demonstrate that our method outperforms state-of-the-art methods in most cases, both visually and quantitatively.

2. Related Work

In this paper, we only review the most relevant work to our research, including traditional point cloud filtering and deep learning-based point cloud filtering.

2.1. Traditional Point Cloud Filtering

2.1.1 Position-based methods

LOP was first proposed in [17]. It is a parameterization-free method and does not rely on normal estimation. Besides fitting the original model, a density repulsion term was added

to evenly control the point cloud distribution. WLOP [11] provided a novel repulsion term to solve the problem that the original repulsion function in LOP drops too quickly as the support radius increases. The filtered points are distributed more evenly by WLOP. EAR [12] added an anisotropic weighting function to WLOP to smooth the model while preserving sharp features. CLOP [28] is another LOP-based approach; it redefines the data term as a continuous representation of a set of input points.

Although only based on point positions, these approaches achieve reasonable smoothing results. However, as they disregard normal information, these approaches tend to smear sharp features such as sharp edges and corners.

2.1.2 Normal-based methods

FLOP [16] adds normal information to the novel feature-preserving projection operator and preserves features well. Meanwhile, a new kernel density estimate (KDE)-based random sampling method was proposed for accelerating FLOP. MLS-based approaches [14, 15] have also been applied to point cloud filtering; they rely upon the assumption that the given set of points implicitly define a surface. In [1], the authors presented an algorithm that allocated an MLS local reference domain for each point that most suited its adjacent points and further projected the points to the underlying plane. This approach uses the eigenvectors of a weighted covariance matrix to obtain normals when the input point cloud has no normal information. APSS [7], RMLS [32], and RIMLS [26] use this idea. RIMLS is based on robust local kernel regression and gives better results when noise is high. GPF [25] incorporates normal information in a Gaussian mixture model (GMM) with two terms, and preserves sharp features well. A robust normal estimation method was proposed in [23] for both point clouds and meshes using a low-rank matrix approximation algorithm, where an application of point cloud filtering was demonstrated. To keep the geometric features, [19] first filters the normals using discrete operators defined on point clouds, and uses a bi-tensor voting scheme for the feature detection step.

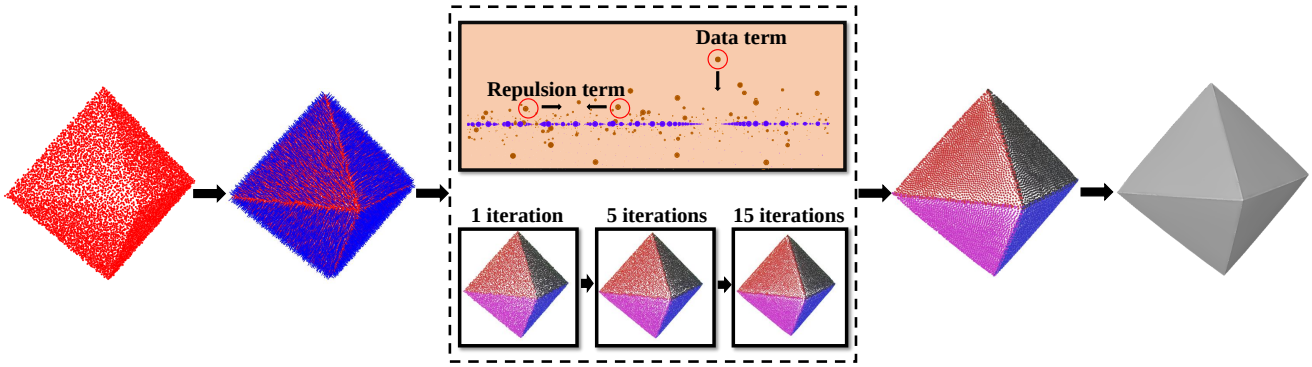
Inspired by image denoising, researchers have also investigated use of non-local data in point cloud denoising. Non-local-based point cloud filtering methods [3, 4, 35, 2] often incorporate normal information and use various similarity definitions to update point positions in a non-local manner. Thus, [3] proposed a similarity descriptor for point cloud patches based on MLS surfaces. [4] designed a height vector field to describe the difference between the neighborhood of the point with neighborhoods of other points on the surface. Inspired by the low-dimensional manifold model, [35] extends it from image patches to point cloud surface patches, which serves as a similarity descriptor for non-local patches. [2] presented a new multi-patch collaborative method that regards denoising as a low-rank matrix recovery problem. They define the given patch as a rotation-invariant height-map patch and denoise the points by imposing a graph constraint.

Filtering methods that rely on normal information usually yield good results, especially for point clouds with sharp features. However, these methods strongly depend on the quality of the input normals, and poor normal estimates may lead to poor filtering results.

Our proposed approach falls in the normal-based category. Inspired by GPF, we estimate normals of the input point cloud using bilateral filtering [12] to get high-quality normal information. Note that if the input point cloud only contains positional information, PCA is used to compute the initial normals. The point positions are then updated in a local manner using the bilaterally filtered normals [23]. We also add a repulsion term [23] to ensure a more uniform distribution of the filtered points.

2.2. Deep Learning Point Cloud Filtering

A variety of deep learning-based methods have emerged for dealing with noisy point clouds [18, 6, 36, 5, 33, 30, 20]. For point cloud filtering, PointProNets [31] introduced a novel generative neural network architecture that encodes geometric features in a local way and efficiently obtains an underlying surface. However, the generated underlying surface suffers from holes due to the input shapes. NPD [5] redesigned the framework on the basis of PointNet [29] to estimate normals from noisy shapes and then projected the noisy points to the predicted reference planes. Another PointNet-inspired method is called Pointfilter [36]. It starts from points and learns the displacement between the predicted points and the raw input points. Moreover, this approach requires normals only in the training phase. In the testing phase, only point positions are taken as input to obtain filtered shapes with feature-preserving effects. EC-NET [33] presented an edge-aware network (similar to PU-NET [34]) for connecting edges of the original points. This method retains sharp edges of 3D shapes well, but the training stage requires manual labeling of edges. Inspired by PCPNet [8], PointCleanNet [30] developed a data-driven method for both classifying outliers and reducing noise in raw point clouds. A novel feature-preserving normal estimation method was designed in [20] for point cloud filtering while preserving geometric features. Deep learning-based filtering methods usually yield good results more automatically, for point clouds with high density. However, low-density input may lead to poor filtering outcomes. Also, such methods require sufficient suitable samples for training.



(a) Noisy input (b) Normal filtering (c) Position update (d) Result (e) Reconstruction

Figure 1. Approach. (a) Noisy input. Red points are corrupted with noise. (b) Filtered normals. Blue lines are filtered results of the initial normals. (c) Position update method, using a data term for feature preservation and a repulsion term for uniform distribution. Multiple iterations are performed to achieve a better filtered result. (d) Filtered point cloud. (e) Mesh reconstructed from (d).

3. Method

3.1. Overview

Our approach has two phases. In phase one, we smooth the initial normals using bilateral filtering, following [12], to ensure the quality of normals. In phase two, we update point positions with the smoothed normals to obtain a uniformly distributed point cloud preserving geometric features. Figure 1 overviews the proposed approach. We now explain the second phase in detail.

3.2. Position Update

We define the noisy input with M points to be $P = \{\mathbf{p}_i\}_{i=1}^{M}$, $\mathbf{p}_i \in R^3$, with corresponding filtered normals $N = \{\mathbf{n}_i\}_{i=1}^{M}$, $\mathbf{n}_i \in R^3$. To obtain local information from a given point $\mathbf{p}_i$, we define a local structure $\mathbf{s}_i$ for each point in the point cloud, consisting of the k nearest points to the current point. We employ an edge-aware recovery algorithm [23] to obtain filtered points by minimizing

$$\mathcal{D}(P,N) = \sum_i \sum_{j \in \mathbf{s}_i} |\,(\mathbf{p}_i - \mathbf{p}_j)\,\mathbf{n}_j^T|^2 + |\,(\mathbf{p}_i - \mathbf{p}_j)\,\mathbf{n}_i^T|^2, \tag{1}$$

where $\mathbf{p}_i$ is the point to be updated and $\mathbf{p}_j$ is some neighboring point in the corresponding set $\mathbf{s}_i$. Eq. (1) essentially adjusts the angles between the tangent vector formed by $\mathbf{p}_i$ and $\mathbf{p}_j$ and the corresponding normal vectors $\mathbf{n}_i$, $\mathbf{n}_j$.

Figure 2 demonstrates how the points are updated on an assumed local plane by this edge-aware technique. It can be seen that the quality of the filtered points depends heavily on the quality of the estimated normals. Our normals are generated by bilaterally filtering the original input normals, given the simplicity and effectiveness of this approach.

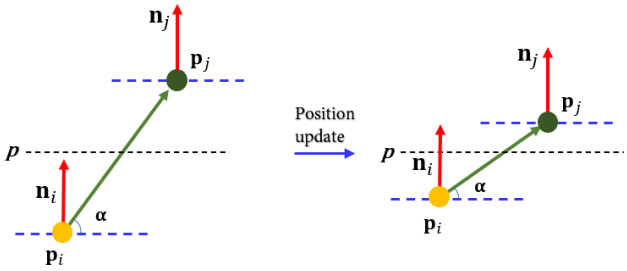


Figure 2. Left: original points. Right: updated points. $\mathbf{p}_i$ and $\mathbf{p}_j$ are the current point and a neighboring point. $\mathbf{n}_i$, and $\mathbf{n}_j$ are the normals of $\mathbf{p}_i$ and $\mathbf{p}_j$, respectively. A local plane surface is assumed.

3.3. Repulsive Force

From Figure 3, it can be seen that points move towards sharp edges during the position update step, resulting in gaps near sharp edges. It is demonstrated in [23] that minimizing $\mathcal{D}(P,N)$ inevitably yields gaps near sharp edges; the gaps in the filtered points can greatly impact downstream applications such as upsampling and surface reconstruction. Thus, we introduce a repulsion force $\mathcal{R}(P,N)$ based on both point coordinates and normals [25] to better control the distribution of points:

$$\mathcal{R}(P,N) = \sum_i \lambda_i \sum_{j \in s_i}^{M} \eta\,(r_{ij})\,\theta\,(r_{ij})\,, \tag{2}$$

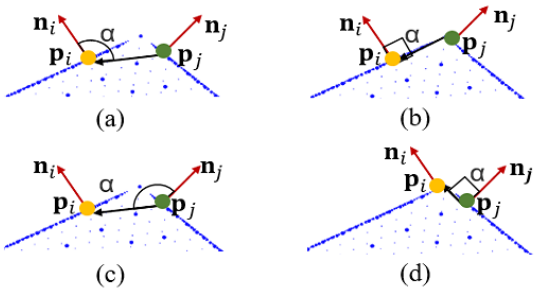


Figure 3. Movement of the filtered points around sharp edges. Blue points: underlying surface. Yellow, green points: two neighboring points that need to be moved. (a, b) movement of $\mathbf{p}_j$ with fixed $\mathbf{p}_i$. (c, d) movement of $\mathbf{p}_i$ with fixed $\mathbf{p}_j$. Note how point move towards the sharp edges and concentrate there, leading to gaps around the sharp edges.

where $r_{ij} = \left\|(\mathbf{p}_i - \mathbf{p}_j) - (\mathbf{p}_i - \mathbf{p}_j)\,\mathbf{n}_j^T\mathbf{n}_j\right\|$, $\eta(r) = -r$, and $\theta(r) = \exp\left(-\mathbf{r}^2/(h/2)^2\right)$ is a smoothly decaying weight function.

3.4. Minimization

Combining Eq. (1) and Eq. (2), our final position update optimization problem is to find:

$$A = \underset{P}{\operatorname{argmin}}\, \mathcal{D}(P, N) + \mathcal{R}(P, N) \tag{3}$$

We employ the gradient descent method to do so and obtain each updated point $\mathbf{p}_i'$. The partial derivative of Eq. (3) with respect to $\mathbf{p}_i$ is:

$$\frac{\partial A}{\partial \mathbf{p}_i} = \sum_{j \in \mathbf{s}_i} \frac{\left(\mathbf{n}_j\mathbf{p}_i^T - \mathbf{n}_j\mathbf{p}_j^T\right)\left(\mathbf{p}_i\mathbf{n}_j^T - \mathbf{p}_j{\mathbf{n}_j}^T\right)}{\partial \mathbf{p}_i} + \frac{\lambda_i\beta_{ij}\left(\mathbf{p}_i - \mathbf{p}_j\right)\left(\mathbf{I} - \mathbf{n}_j^T\mathbf{n}_j\right)}{\partial \mathbf{p}_i}, \tag{4}$$

where $\mathbf{I}$ is a 3×3 identity matrix, and

$$\beta_{ij} = \frac{\theta\left(r_{ij}\right)}{r_{ij}}\left|\frac{\partial \eta\left(r_{ij}\right)}{\partial r}\right|. \tag{5}$$

The updated point $\mathbf{p}_i'$ can thus be calculated by:

$$\mathbf{p}_i' = \mathbf{p}_i + \gamma_i \sum_{j \in s_i}\left(\mathbf{p}_j - \mathbf{p}_i\right)\left(\mathbf{n}_j^T\mathbf{n}_j + \mathbf{n}_i^T\mathbf{n}_i\right) + \mu\frac{\sum_{j \in s_i} w_j\beta_{ij}\left(\mathbf{p}_i - \mathbf{p}_j\right)\left(\mathbf{I} - \mathbf{n}_j^T\mathbf{n}_j\right)}{\sum_{j \in s_i} w_j\beta_{ij}}, \tag{6}$$

where γ_i is set to $1/(3\,|s_i|)$ following [23], $w_j = 1 + \sum_{j \in s_i}\theta\left(\|\mathbf{p}_i - \mathbf{p}_j\|\right)$, and μ is a parameter which controls the relative magnitude of the repulsive force.

Algorithm 1 Point cloud filtering algorithm

Input: Noisy point set P, with corresponding filtered normals N, neighborhood size k, number of iterations t, repulsion strength μ
Output: Uniformly distributed set of filtered points P'
for t iterations **do**
  for each point $\mathbf{p}_i$ **do**
    construct a local patch $\mathbf{s}_i$ from k nearest neighbors
    update point position via Eq. (6)
  end for
end for

Table 1. Parameter settings for various models.

Parameter	k	μ	t
Figure 4	30	0.3	5
Figure 5	30	0.3	5
Figure 6	30	0.3	5
Figure 7	30	0.3	3
Figure 8	30	0.3	5
Figure 9	30	0.3	5
Figure 10	30	0.3	10
Figure 11	30	0.3	5
Figure 12	30	0.3	5

3.5. Algorithm

The proposed method is described in Algorithm 1. We first filter the normals using bilateral filtering. By feeding the filtered normals and raw point positions into Algorithm 1, we obtain the updated point positions. Depending on the number of points in the model and the noise level, we accordingly choose k used to generate the local patches and the number of iterations to perform. Table 1 lists the parameter settings used for various models in the experiments.

4. Experiments

4.1. Settings

The proposed method was implemented in Visual Studio 2017 and executed on a PC with an Intel i9-9750h CPU and NVidia RTX2070 GPU.

4.2. Parameter Settings

The parameters to be chosen are the local neighborhood size k, the coefficient of repulsion force μ, and the number of iterations t. As the number of points affects the range of neighbors significantly, in order to find the appropriate k for different models, we determined the value of k in the range [15, 45] (k = 30 by default) according to the number of points in each model. To make the distribution of points more even while preserving features, we use the parameter μ to balance the magnitude of the repulsive force between points and t to control the number of iterations. Usually, for

models with relatively smooth surfaces, we set a relatively larger μ and a lower t, and for models with sharp surfaces, we set a relatively smaller μ with a higher number of iterations to obtain the filtered points. Table 1 gives the parameters used for each model considered in the experiments.

4.3. Methods Compared and Approach

The proposed method was compared to state-of-the-art techniques including a non-deep learning position-based method CLOP [28], non-deep learning normal-based methods GPF [25] and RIMLS [26], and deep learning-based methods TotalDenoising (TD) [9], PointCleanNet (PCN) [30] and Pointfilter (PF) [36]. We employ the following approach to ensure a fair comparison. (a) We first normalized and centralized the noisy input. (b) As GPF and RIMLS require high-quality normals, we used the same bilateral filter [12] in each case to provide the same input normals for each model. (c) We tuned the main parameters of each method as well as we could, to produce the best final visual results. (d) For the deep learning-based methods, we used the results of the 6^{th} iteration for TD and iterate three times for both PCN and PF. (e) For visual comparisons, we used EAR [12] for upsampling to provide a similar number of points for each method when visualizing a given model. For surface reconstruction, we adopted the same parameters for each given model.

4.4. Evaluation Metrics

We used two common evaluation metrics to quantitatively analyze the results. Let the ground-truth point cloud and the filtered point cloud be respectively defined as: $S_1 = \{\mathbf{x}_i\}_{i=1}^{|S_1|}$, $S_2 = \{\mathbf{y}_i\}_{i=1}^{|S_2|}$. Note that the numbers of ground-truth points $|S_1|$ and filtered points $|S_2|$ may differ slightly. The metrics are: *chamfer distance*:

$$e_{\text{CD}}\left(S_1, S_2\right) = \frac{1}{|S_1|}\sum_{x\in S_1}\min_{y\in S_2}\|x-y\|_2^2 + \frac{1}{|S_2|}\sum_{y\in S_2}\min_{x\in S_1}\|y-x\|_2^2, \tag{7}$$

and *mean square error*:

$$e_{\text{MSE}}(S_1, S_2) = \frac{1}{|S_1||\text{NN}(y)|}\sum_{x\in S_1}\sum_{y\in \text{NN}(y)}\|x-y\|_2^2, \tag{8}$$

where $\text{NN}(y)$ denotes the nearest neighbors in S_1 to point y in S_2. Here we set $|\text{NN}(y)| = 10$ following [36], i.e. we search for 10 nearest neighbors for each point y in the predicted point set S_2.

4.5. Visual Comparisons

4.5.1 Point clouds with synthetic noise

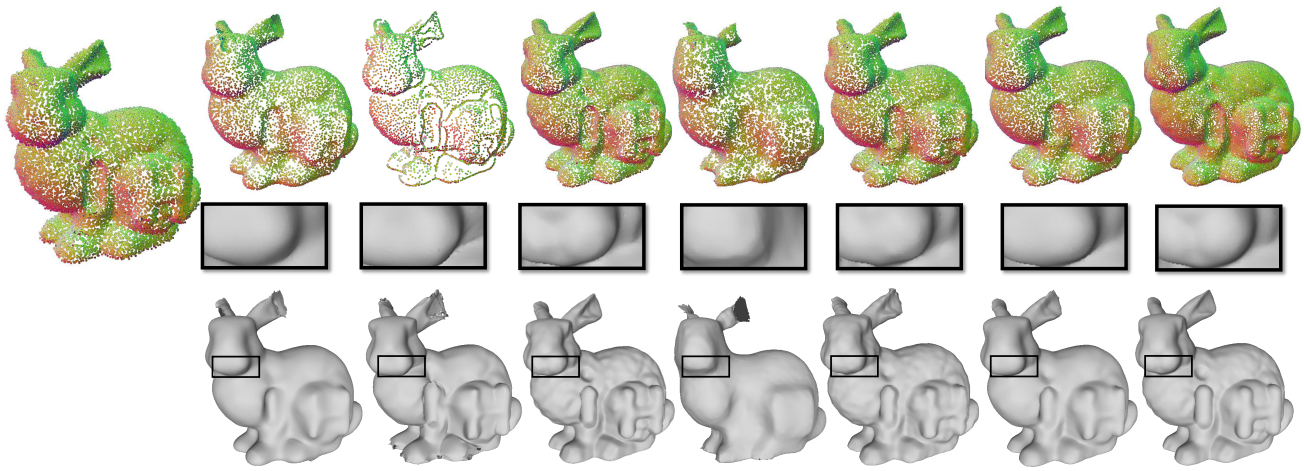
To show the denoising effects of our method, we conducted experiments on models with synthetic Gaussian noise at levels of 0.5% and 1.0%. Compared to other state-of-the-art methods, our visual results outperform them both in terms of smoothing and feature preservation. The results benefit from the fact that the position update considers normal information and distributes the filtered points more evenly.

We may also observe the traits of other methods in the experiments. CLOP always obtains good results in terms of smoothing. However, since it is a position-based method, it may blur sharp features. While GPF adds a gap-filling step after projecting the points onto the underlying surface, it still finds it difficult to maintain a uniform distribution, especially for points near sharp edges. This method may also sharpen less sharp features. RIMLS yields promising results in both noise removal and feature preservation. However, its filtered points are often unevenly distributed, which affects the performance of downstream applications such as upsampling and surface reconstruction.

The learning-based method TD also yields good smoothing results, but it does not seem to maintain the fine features of the model well. PCN typically produces less sharp features, while PCN does not provide good smoothing given a relatively high level of noise. PF does not need normal information at run time, and achieves good feature-preserving effects while denoising. However, when the noisy points are sparse, this method cannot extract needed information from the sparse point cloud, leading to distortion of the filtered points.

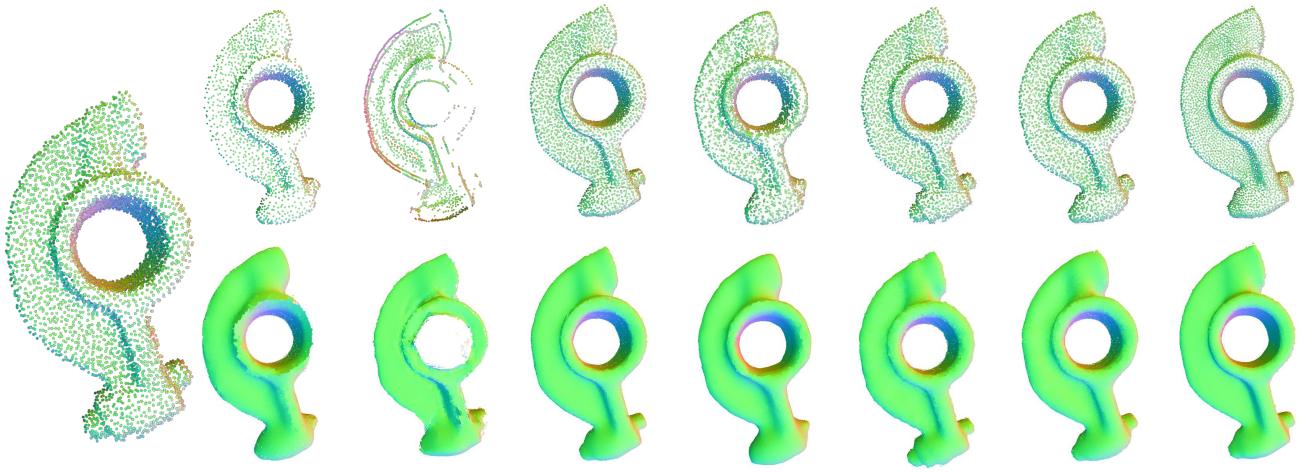
Since normal information is taken into account in our method, it can keep sharp features well. Importantly, the greater uniformity of its point distribution makes it stand out in point cloud filtering and downstream applications like upsampling and surface reconstruction.

The top rows of Figures 4–6, readily show that our method provides the most uniform point distribution. Figures 5, 6 and 8 give upsampling results for three different models after filtering. As shown below in Figures 5 and 6, sharp edges are maintained well during denoising. The enlargement in Figure 8 also shows the effect of our filtering method, where the shapes of the kitten's ears are well maintained . Results of surface reconstruction are presented in Figures 4 and 7. As the enlargement shows, the bunny's mouth and nose in Figure 4 are well maintained. Figure 7 also shows filtered results for our method on a simple geometric model with sharp edges. Our method is best in terms of maintaining details and sharp edges.



(a) Noisy input (b) CLOP (c) GPF (d) RIMLS (e) TD (f) PCN (g) PF (h) Ours

Figure 4. Above: results on the Bunnyhi model corrupted with 0.5% synthetic noise. Below: surface reconstruction results.



(a) Noisy input (b) CLOP (c) GPF (d) RIMLS (e) TD (f) PCN (g) PF (h) Ours

Figure 5. Above: results on Rockerarm corrupted with 0.5% synthetic noise. Below: corresponding upsampling results.

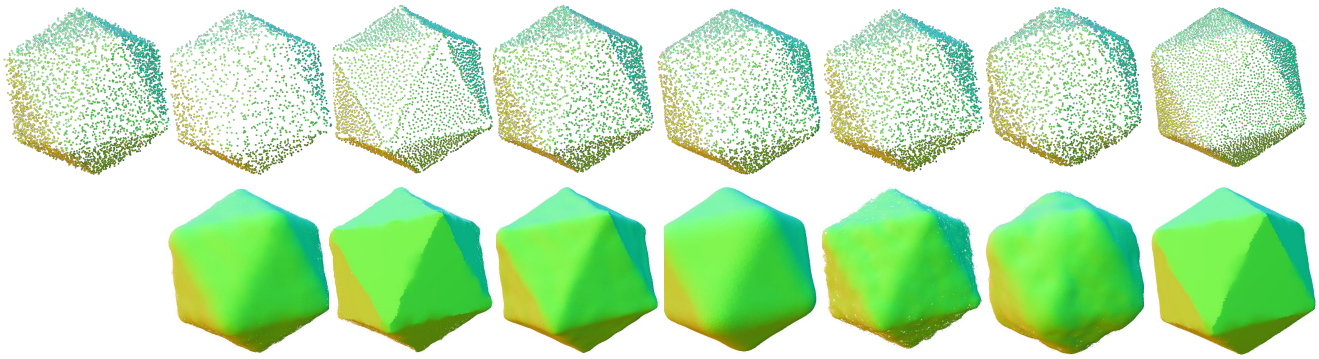
4.5.2 Point clouds with raw scanner noise

In addition to using synthetic noise, we also performed experiments on raw scanned point clouds. Results for our method and existing methods are given in Figures 9–12. The filtered results in Figure 9 show that our method performs best in terms of smoothing and preserving details. As can be seen from the enlargement, most methods blur the mouth of the model or even lose it after denoising. Note that although the model we use here has the same shape as one used in [36], the filtered results may differ since our sample points were sparser than theirs.

Figure 10 shows the filtered results on the BuddhaStele raw scanned model. Results after upsampling are shown above; below are results after surface reconstruction using the screened Poisson method [13]. From details such as the steps in the model, it can be seen that our method again outperforms the other methods. In Figure 11, our method again maintains sharp edges well. As the enlargement shows, other state-of-the-art methods either distort sharp edges or smooth them. Figure 12 shows filtered results on the David raw scanned model. Our method preserves features better during filtering, and as the enlargement shows, our approach maintains facial features better than other methods.

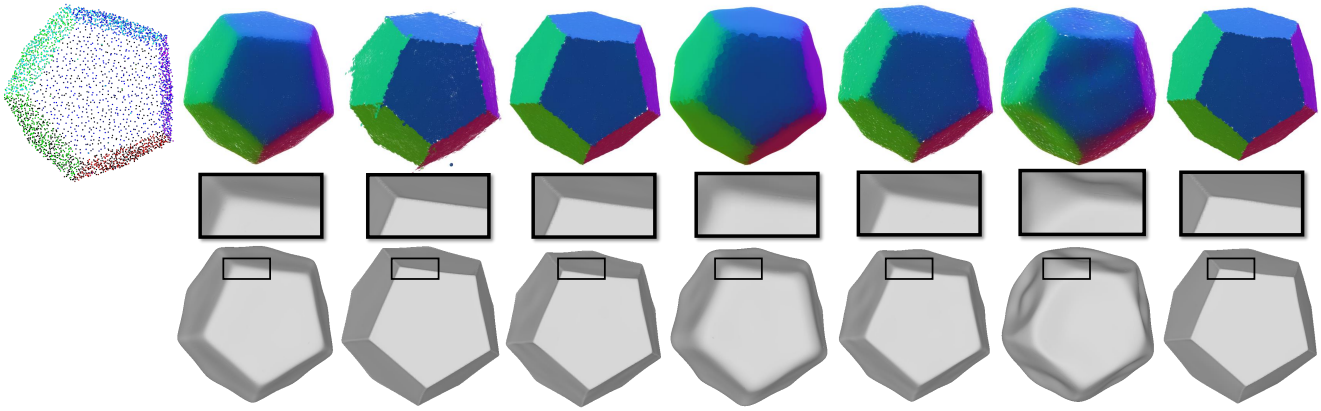
4.6. Quantitative Comparisons

We also make a quantitative comparison using the two evaluation metrics given earlier. As there is no corresponding ground-truth model for the raw scanned point clouds, we used the models with synthetic noise for evaluation. Re-



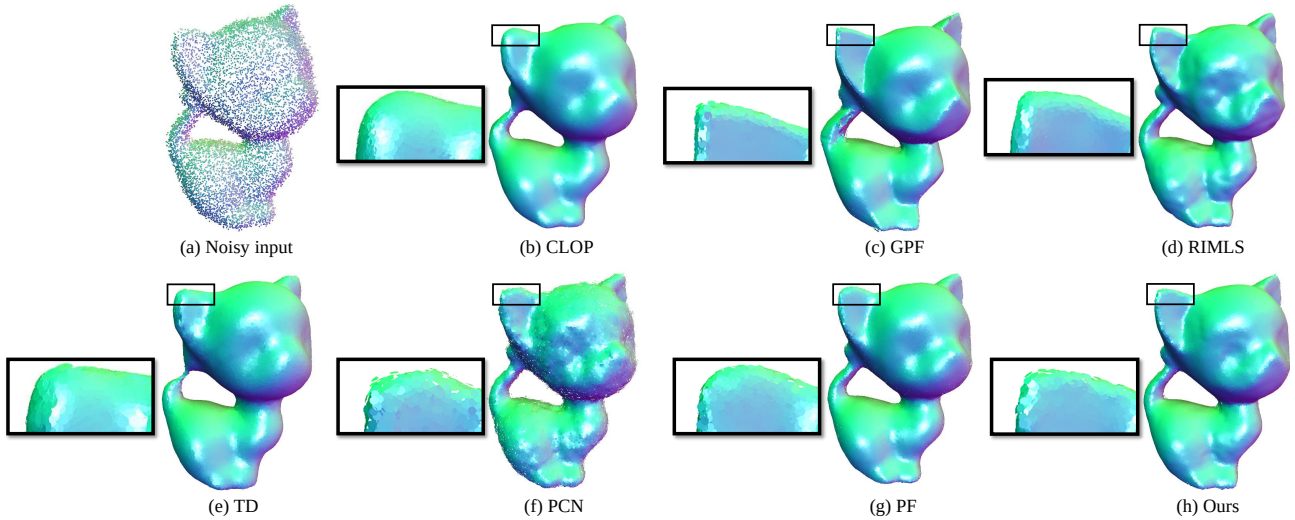
(a) Noisy input (b) CLOP (c) GPF (d) RIMLS (e) TD (f) PCN (g) PF (h) Ours

Figure 6. Above: results on Icosahedron corrupted with 1.0% synthetic noise. Below: corresponding upsampling results.



(a) Noisy input (b) CLOP (c) GPF (d) RIMLS (e) TD (f) PCN (g) PF (h) Ours

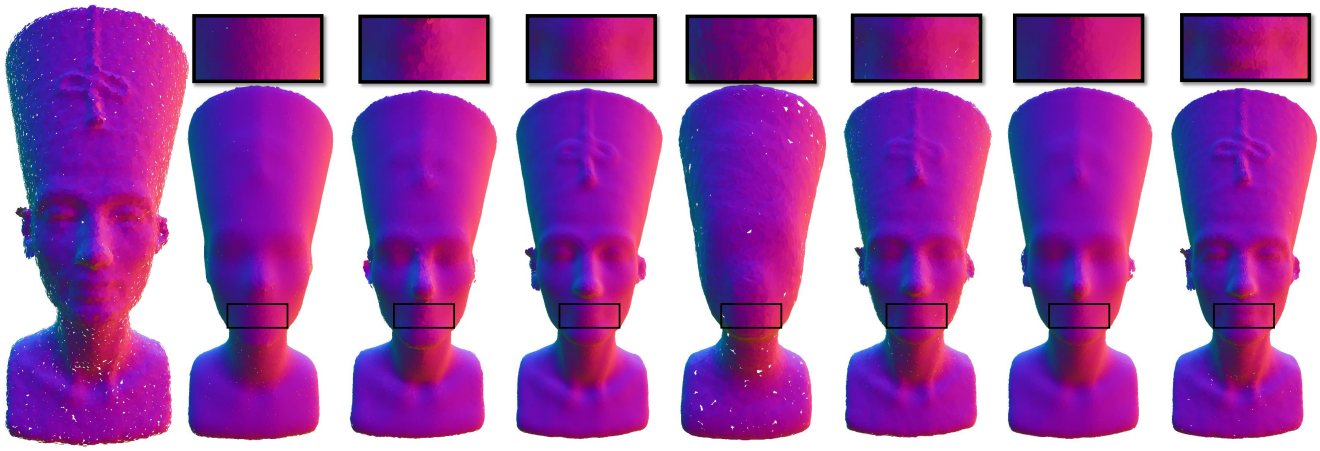
Figure 7. Results on Dodecahedron corrupted with 0.5% synthetic noise. Above: corresponding upsampling results. Below: reconstructed meshes.



(a) Noisy input (b) CLOP (c) GPF (d) RIMLS

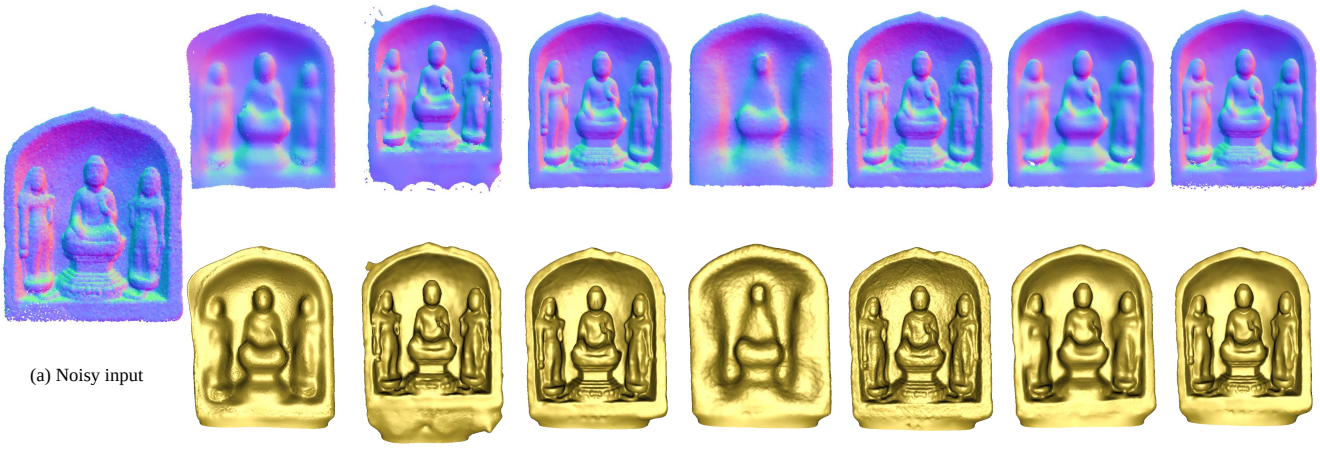
(e) TD (f) PCN (g) PF (h) Ours

Figure 8. Upsampling results on kitten corrupted with 1.0% synthetic noise.



(a) Noisy input (b) CLOP (c) GPF (d) RIMLS (e) TD (f) PCN (g) PF (h) Ours

Figure 9. Upsampling results on the Nefertiti raw scanned model.



(a) Noisy input (b) CLOP (c) GPF (d) RIMLS (e) TD (f) PCN (g) PF (h) Ours

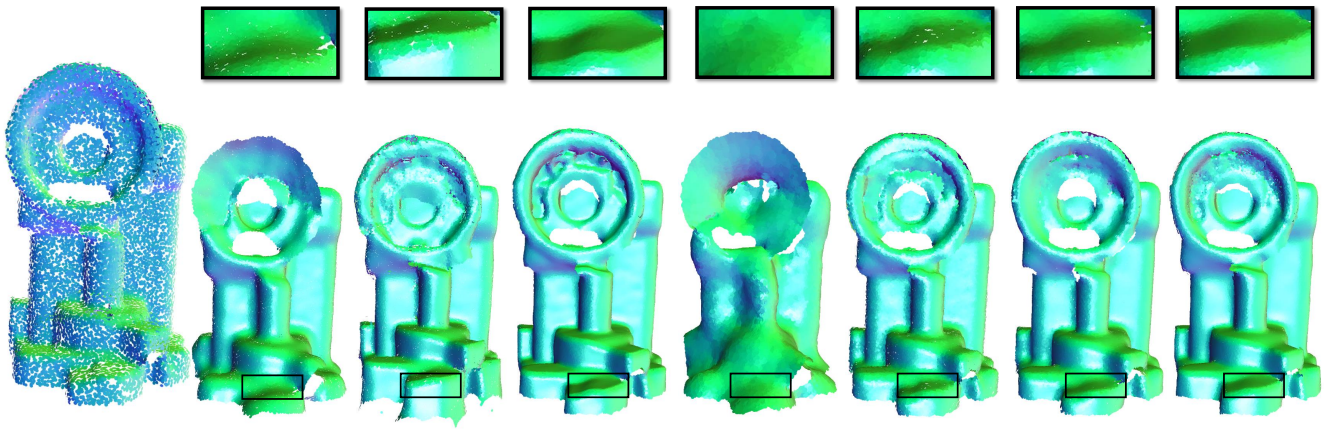
Figure 10. Results on BuddhaStele raw scanned model. Above: corresponding upsampling results. Below: reconstructed meshes.

sults using the Chamfer Distance metric are given in Table 2. Despite the fact that deep learning-based methods are trained on a large number of point clouds, our method still outperforms all deep learning-based methods and indeed achieves the best quantitative result in most cases. In terms of the other evaluation metric MSE, our method still outperforms most deep learning methods and again provides the lowest quantitative error in most cases, as shown in Table 3.

These quantitative results are consistent with the visual results, demonstrating that our method generally outperforms existing methods both visually and quantitatively. We believe this is because our method provides a more uniform distribution of the filtered points and can handle both sparsely and densely sampled point clouds. In the case of sparse sampling, some deep learning-based methods are unable to obtain meaningful local geometric information from the sparse local neighboring points. It is also worth noting that although RIMLS achieves comparable visual results to our method in some cases, its numerical errors are greater than those for our method in most cases due to its uneven point cloud distribution.

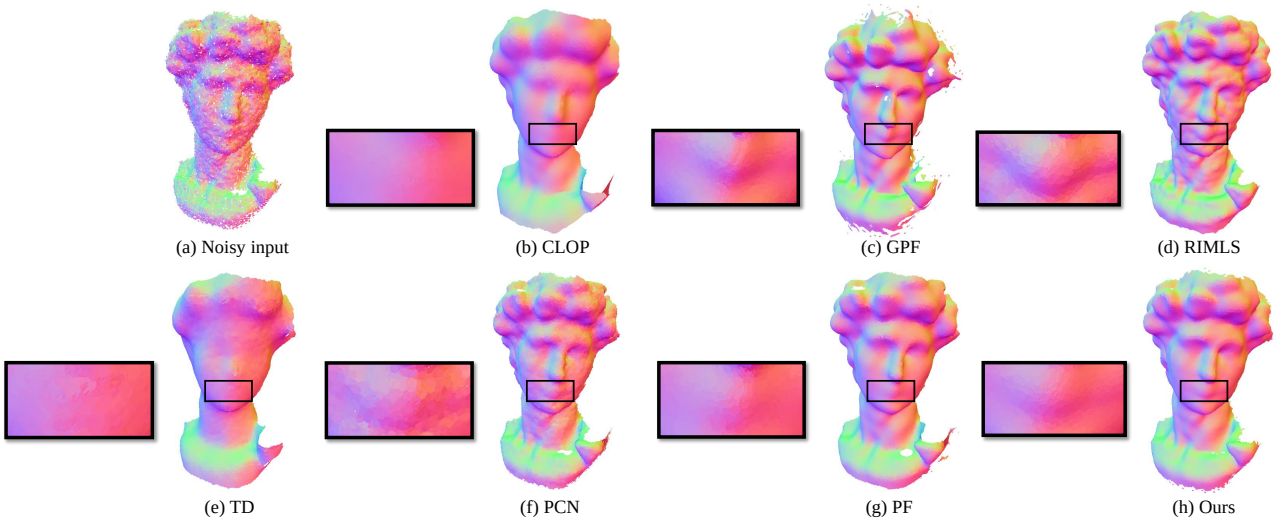
4.7. Parameter Values

We now consider settings for the various parameters. We performed experiments on a point cloud containing 7682 points using different values of k. Figure 13 shows the best value of k is 30, which we use as the default value for this parameter. It is clear that the best k depends strongly on the density of the point cloud. Using a fixed value of k, when the model has sparse points, the locality determined by k is larger, which may lead to an excessive range that should not be treated as local information, resulting in poor outcome.



(a) Noisy input (b) CLOP (c) GPF (d) RIMLS (e) TD (f) PCN (g) PF (h) Ours

Figure 11. Upsampling results on the Realscan raw scanned model.



(a) Noisy input (b) CLOP (c) GPF (d) RIMLS

(e) TD (f) PCN (g) PF (h) Ours

Figure 12. Upsampling results on the David raw scanned model.

For point clouds with denser distribution, the size of the neighborhood for the same k becomes smaller, meaning that the k neighborhood contains only a smaller amount of very local information, leading to an uneven distribution of the point cloud. Generally, we use a larger k for denser point clouds to ensure an appropriate number of local neighbors.

As μ is related to the number of iterations t, we give filtered results for different values of μ for various numbers of iterations. Figure 14 demonstrates the filtered point clouds obtained for different μ values when $t = 30$ and $k = 30$. We can see from this figure that as μ increases, the distribution of the point cloud becomes more uniform, but making μ too large makes the model chaotic again. Figure 14(b) shows the filtered results using a low value of μ when $i = 30$ and $k = 30$. As we can see, a smaller μ better maintains the edges of the model.

We also conducted experiments using different numbers of iterations. Figure 15 shows that with increasing iterations, the distribution of the filtered point cloud becomes more uniform. However, Figure 15(d) shows that if too many iterations are used, the boundary of the model becomes unclear again.

4.8. Repulsion term

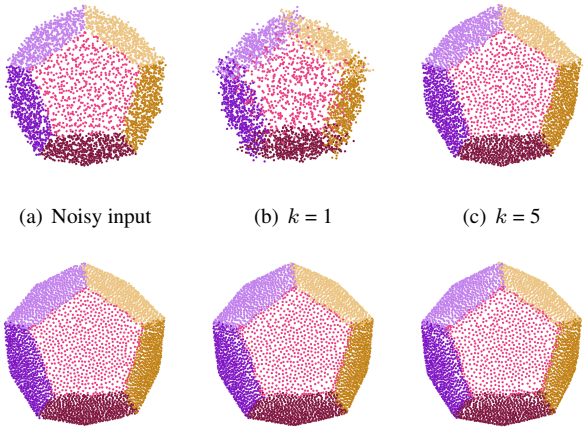
Local-based filtering approaches tend to converge in certain places when updating the positions. Obviously, this will make downstream applications such as surface reconstruction very difficult. Our method adopts the repulsion term mentioned in Section 3 to evenly distribute the points while filtering, thus improving the quality of the filtered point cloud. As Figure 16(a) shows, without the repulsion term, it is clear that some points are concentrated at edges,

Table 2. Quantitative evaluation of methods on the synthetic point clouds in Figures 4–8. Deep learning methods are indicated by *. Metric used is chamfer distance ($\times 10^{-5}$), The best method for each model is highlighted in bold.

Method	Figure 4	Figure 5	Figure 6	Figure 7	Figure 8	Avg.
CLOP [28]	7.84	25.35	26.46	23.83	6.73	16.70
GPF [25]	16.19	31.85	21.52	18.35	15.54	17.58
RIMLS [26]	3.72	**5.22**	15.70	10.98	4.16	7.12
TD* [9]	23.88	13.20	24.43	19.22	11.86	16.15
PCN* [30]	4.76	6.38	29.87	14.96	6.24	11.18
PF* [36]	4.01	6.63	33.38	30.60	3.68	14.93
Ours	**3.11**	5.48	**12.26**	**8.14**	**3.18**	**5.80**

Table 3. Quantitative evaluation of methods on the synthetic point clouds in Figures 4–8. Deep learning methods are indicated by *. Metric used is mean square error ($\times 10^{-3}$). The best method for each model is highlighted in bold.

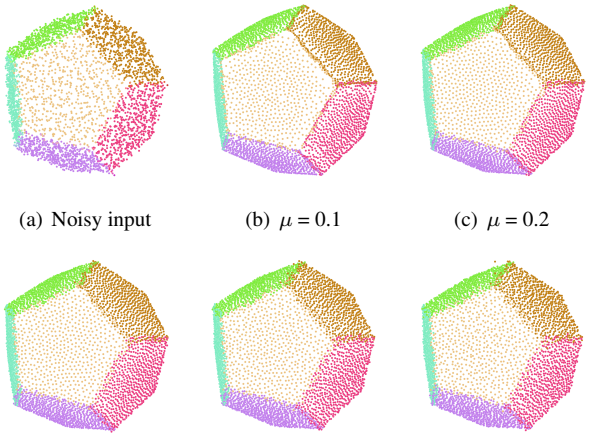
Method	Figure 4	Figure 5	Figure 6	Figure 7	Figure 8	Avg.
CLOP [28]	10.32	**13.91**	21.86	23.31	9.88	15.86
GPF [25]	11.64	17.07	22.43	23.88	11.97	17.40
RIMLS [26]	10.05	14.02	21.68	23.30	10.02	15.81
TD* [9]	13.22	14.78	22.44	23.36	11.45	17.05
PCN* [30]	10.30	14.28	23.68	23.79	10.59	16.53
PF* [36]	10.02	14.17	23.71	25.28	**9.82**	16.60
Ours	**9.92**	14.01	**21.46**	**23.15**	9.92	**15.69**



(a) Noisy input (b) $k = 1$ (c) $k = 5$

(d) $k = 15$ (e) $k = 30$ (f) $k = 45$

Figure 13. Filtered results for different k. Noise level: 1.0%. $t = 30$, $\mu = 0.3$.



(a) Noisy input (b) $\mu = 0.1$ (c) $\mu = 0.2$

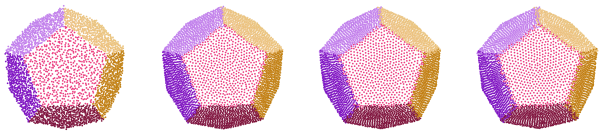
(d) $\mu = 0.3$ (e) $\mu = 0.4$ (f) $\mu = 0.5$

Figure 14. Filtered results for different μ. Noise level: 1.0%, $t = 30$, $k = 30$.

whereas the distribution in Figure 16(b) is more even.

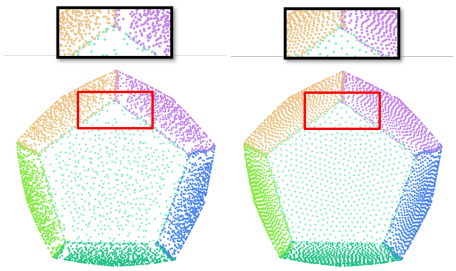
4.9. Point density

Results under different point densities were also tested. Figure 17 shows that our method yields promising results for both sparse and dense point clouds. As our method requires only local information, for point clouds with greater point density, a desired filtered result can be obtained by setting a larger k.



(a) Noisy input (b) 5 iterations (c) 15 iterations (d) 30 iterations

Figure 15. Filtered results for different number of iterations. Noise level: 1.0%, $k = 30$, $\mu = 0.3$.



(a) Without (b) With

Figure 16. Filtered results without and with the repulsion term.

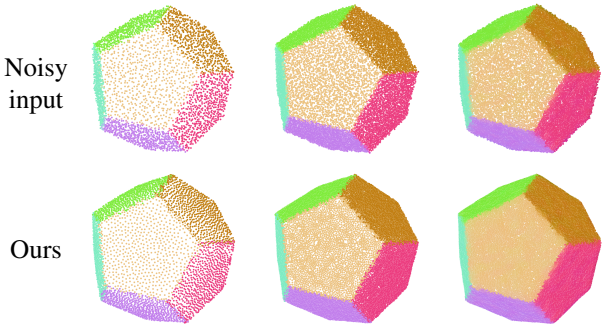


Figure 17. Filtered results for models with different numbers of sample points.

4.10. Noise level

Different noise levels were applied to the same model to verify the robustness of our approach. Figure 18 gives the filtered results by our method under noise levels of 0.5%, 1.0%, 1.5%, 2.0%, 2.5% and 3.0%. It can be seen that our method is capable of handling models with different levels of noise but may work less well given excessively high noise. As our method relies on the quality of normals, it is difficult to accurately keep geometric features if the model has inaccurate normals caused by high noise levels.

4.11. Irregular sampling

We conducted further experiments on models with irregular sampling. Figure 19 provides a visual comparisons of results on an unevenly sampled model for PCN [30], PF [36], and our method. It can be seen that the filtered point cloud from PCN still contains obvious noise while PF blurs the detail features. Our method smooths the model better while preserving features.

4.12. Hole filling

Taking a cube as an example, we experimented on a model with holes. Figure 20 shows the filtered results for different holes. Our method is capable of filling relatively small holes because we consider the distribution of the updated points. However, it is challenging to fill big holes that severely disrupt the surfaces of the model.

Table 4. Runtime (in seconds) on Dodecahedron for different k and t.

Iterations	$t = 5$	$t = 15$	$t = 30$	$t = 60$
$k = 30$	1.41	3.77	7.37	14.69
$k = 60$	2.33	6.37	12.36	24.36

Table 5. Runtime (s) for different methods and models.

Method	CLOP	TD	PCN	PF	Ours
Fig. 4	59.66	16.65	294.00	67.38	**6.27**
Fig. 5	10.82	6.17	78.04	13.86	**1.80**
Fig. 6	3.89	4.91	83.69	38.98	**1.89**
Fig. 7	2.60	5.53	28.24	70.81	**0.91**
Fig. 8	42.85	16.24	186.30	49.99	**4.66**
Fig. 9	60.92	155.76	317.67	81.10	**7.93**
Fig. 10	70.58	102.41	642.01	247.77	**6.37**
Fig. 11	103.91	40.52	241.05	68.23	**28.16**
Fig. 12	50.58	30.01	352.99	74.68	**6.98**

4.13. Comparison to Lowrank

Figure 21 compares results of our approach to those of Lowrank [23]. It demonstrates that our method results in a more uniform point distribution than Lowrank when removing noise.

4.14. Indoor scenes

We performed an experiment on the more challenging indoor scene data shown in Figure 22. The result shows that our method can also deal with point cloud indoor scenes.

4.15. Runtime

Most examples in this paper were completed within 7 s. The most time-demanding was the object in Figure 11, taking 28 s.

Running times for the proposed method were measured for different k and numbers of iterations; Table 4 shows that as k and the number of iterations increase, the runtime increases accordingly.

In each iteration, our method gathers k neighbors for each point. Thus the larger k is, the longer the computation takes. The number of iterations also has a similar effect on the run time. However, since our method is locally based, the overall speed is not slow.

We also compare the runtime for our method and other methods. Table 5 shows that our method is significantly faster than other methods.

5. Limitations

Although our method achieves good results, it still has room for improvement. Like [25], since it is a normal-based

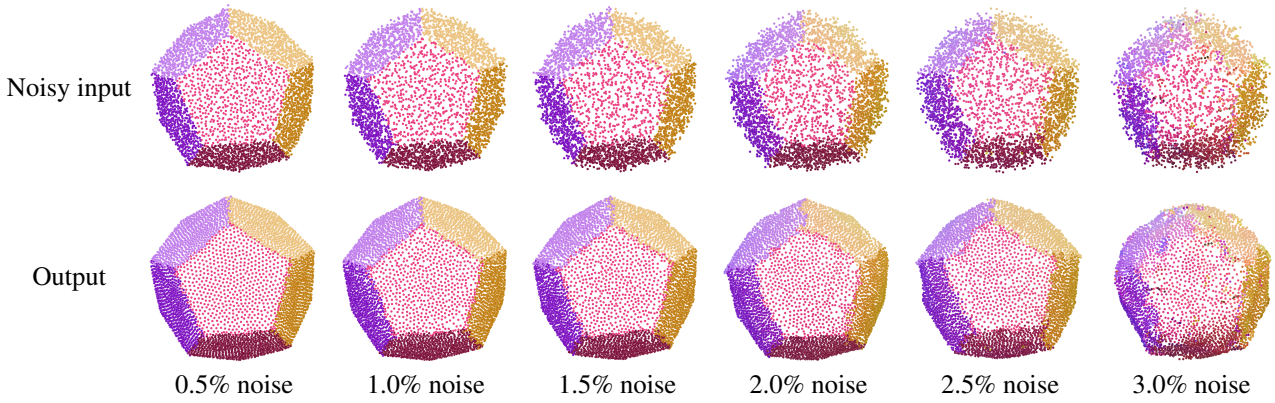


Figure 18. Filtered results for models with different levels of noise.

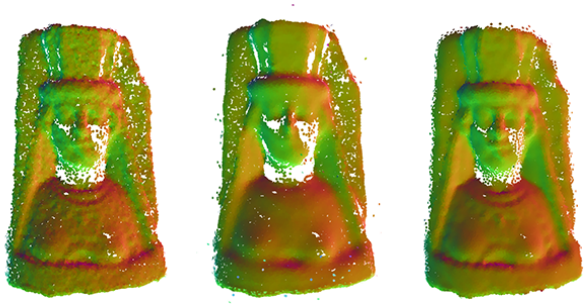


Figure 19. Filtered results for an irregularly sampled point cloud.

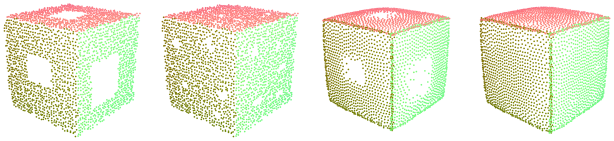


Figure 20. Filtered results with various holes. (a) Input cube with large holes. (b) Input cube with small holes. (c) Output after filtering (a). (d) Output after filtering (b).

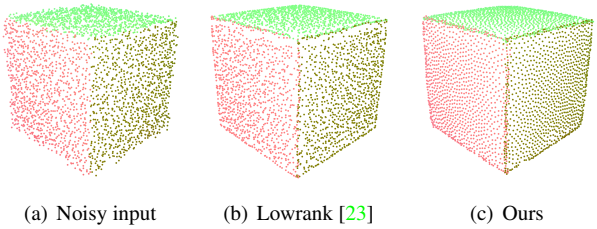


Figure 21. Filtered points of ours and Lowrank [23].

approach, it is inevitably dependent on the quality of the input normals. In each iteration of the position update, each point is estimated with reference to the direction of the normal. Therefore, inaccurate input normals may affect the filtered results. Figure 23 shows an example of such a case.

Also, like previous methods, our method may produce

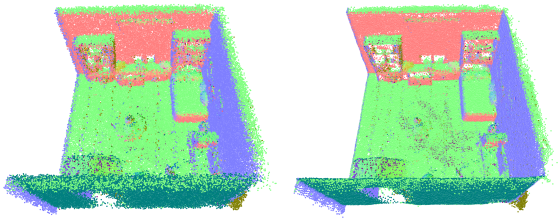


Figure 22. Filtered result on noisy point cloud of an indoor scene.

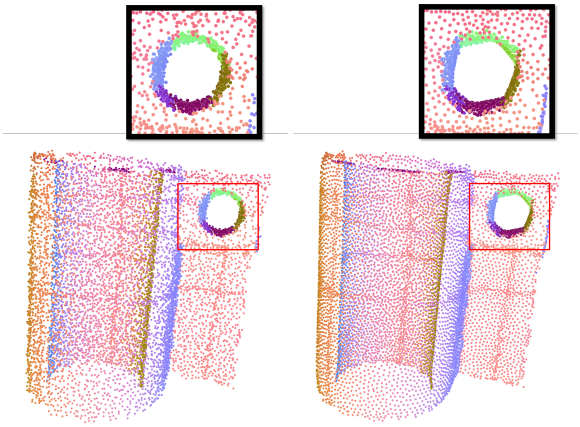


Figure 23. An example showing poor results.

less desirable results when handling a very high level of noise. For instance, Figure 18 indicates 1.5% noise is more challenging than the 0.5% and 1.0% noise.

In future, we hope to develop techniques to handle the above limitations.

6. Conclusion

This paper presents a method to improve point cloud filtering resulting in a more even point distribution in filtered point clouds. Built on top of [23], our method intro-

duces a repulsion term into the objective function. It not only removes noise while preserving sharp features but also ensures a more uniform distribution of the filtered points. Experiments show that our method obtains promising filtered results under different levels of noise and for different densities. Both visual and quantitative comparisons show that it generally outperforms other existing techniques. Our method is also quicker than other compared methods.

References

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003. 2

[2] 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 Transactions on Visualization and Computer Graphics*, 26(11):3255–3270, 2019. 2

[3] J.-E. Deschaud and F. Goulette. Point cloud non local denoising using local surface descriptor similarity. *IAPRS*, 38(3A):109–114, 2010. 2

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

[5] C. Duan, S. Chen, and J. Kovacevic. 3d point cloud denoising via deep neural network based local surface estimation. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8553–8557. IEEE, 2019. 2

[6] P. Erler, P. Guerrero, S. Ohrhallinger, N. J. Mitra, and M. Wimmer. Points2surf learning implicit surfaces from point clouds. In *European Conference on Computer Vision*, pages 108–124. Springer, 2020. 2

[7] G. Guennebaud and M. Gross. Algebraic point set surfaces. In *ACM SIGGRAPH 2007 papers*, page 23, 2007. 2

[8] P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra. PCPNet: Learning local shape properties from raw point clouds. *Computer Graphics Forum*, 37(2):75–85, 2018. 2

[9] P. Hermosilla, T. Ritschel, and T. Ropinski. Total denoising: Unsupervised learning of 3d point cloud cleaning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 5, 10

[10] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th annual conference on computer graphics and interactive techniques*, pages 71–78, 1992. 1

[11] H. Huang, D. Li, H. Zhang, U. Ascher, and D. Cohen-Or. Consolidation of unorganized point clouds for surface reconstruction. *ACM Transactions on Graphics (TOG)*, 28(5):1–7, 2009. 1, 2

[12] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, and H. Zhang. Edge-aware point set resampling. *ACM Transactions on Graphics (TOG)*, 32(1):1–12, 2013. 1, 2, 3, 5

[13] M. Kazhdan and H. Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG)*, 32(3):1–13, 2013. 1, 6

[14] D. Levin. The approximation power of moving least-squares. *Mathematics of Computation*, 67(224):1517–1531, 1998. 2

[15] D. Levin. Mesh-independent surface interpolation. In *Geometric Modeling for Scientific Visualization*, pages 37–49. Springer, 2004. 2

[16] B. Liao, C. Xiao, L. Jin, and H. Fu. Efficient feature-preserving local projection operator for geometry reconstruction. *Computer-Aided Design*, 45(5):861–874, 2013. 2

[17] Y. Lipman, D. Cohen-Or, D. Levin, and H. Tal-Ezer. Parameterization-free projection for geometry reconstruction. *ACM Transactions on Graphics (TOG)*, 26(3):22, 2007. 1

[18] Y. Liu, J. Guo, B. Benes, O. Deussen, X. Zhang, and H. Huang. Treepartnet: neural decomposition of point clouds for 3d tree reconstruction. *ACM Transactions on Graphics (TOG)*, 40(6):1–16, 2021. 2

[19] Z. Liu, X. Xiao, S. Zhong, W. Wang, Y. Li, L. Zhang, and Z. Xie. A feature-preserving framework for point cloud denoising. *Computer Aided Design*, 127:102857, 2020. 2

[20] D. Lu, X. Lu, Y. Sun, and J. Wang. Deep feature-preserving normal estimation for point cloud filtering. *Computer-Aided Design*, 125:102860, 2020. 2

[21] X. Lu, H. Chen, S.-K. Yeung, Z. Deng, and W. Chen. Unsupervised articulated skeleton extraction from point set sequences captured by a single depth camera. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. 1

[22] X. Lu, Z. Deng, J. Luo, W. Chen, S.-K. Yeung, and Y. He. 3d articulated skeleton extraction using a single consumer-grade depth camera. *Computer Vision and Image Understanding*, 188:102792, 2019. 1

[23] X. Lu, S. Schaefer, J. Luo, L. Ma, and Y. He. Low rank matrix approximation for 3d geometry filtering. *IEEE Transactions on Visualization and Computer Graphics*, 2020. 1, 2, 3, 4, 11, 12

[24] X. Lu, Z. Wang, M. Xu, W. Chen, and Z. Deng. A personality model for animating heterogeneous traffic behaviors. *Computer Animation and Virtual Worlds*, 25(3-4):361–371, 2014. 1

[25] X. Lu, S. Wu, H. Chen, S.-K. Yeung, W. Chen, and M. Zwicker. Gpf: Gmm-inspired feature-preserving point set filtering. *IEEE Transactions on Visualization and Computer Graphics*, 24(8):2315–2326, 2017. 1, 2, 3, 5, 10, 11

[26] A. C. Öztireli, G. Guennebaud, and M. Gross. Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum*, 28(2):493–501, 2009. 1, 2, 5, 10

[27] Y. Pei, Z. Huang, W. Yu, M. Wang, and X. Lu. A cascaded approach for keyframes extraction from videos. In F. Tian, X. Yang, D. Thalmann, W. Xu, J. J. Zhang, N. M. Thalmann, and J. Chang, editors, *Computer Animation and Social Agents*, pages 73–81, Cham, 2020. Springer International Publishing. 1

[28] R. Preiner, O. Mattausch, M. Arikan, R. Pajarola, and M. Wimmer. Continuous projection for fast l1 reconstruction. *ACM Transactions on Graphics (TOG)*, 33(4):1–13, 2014. 1, 2, 5, 10

[29] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. 2

[30] M.-J. Rakotosaona, V. La Barbera, P. Guerrero, N. J. Mitra, and M. Ovsjanikov. Pointcleannet: Learning to denoise and remove outliers from dense point clouds. *Computer Graphics Forum*, 39(1):185–203, 2020. 1, 2, 5, 10, 11, 12

[31] R. Roveri, A. C. Öztireli, I. Pandele, and M. Gross. Pointpronets: Consolidation of point clouds with convolutional neural networks. *Computer Graphics Forum*, 37(2):87–99, 2018. 1, 2

[32] R. B. Rusu, N. Blodow, Z. Marton, A. Soos, and M. Beetz. Towards 3d object maps for autonomous household robots. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3191–3198. IEEE, 2007. 2

[33] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng. Ec-net: an edge-aware point set consolidation network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 386–402, 2018. 2

[34] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng. Pu-net: Point cloud upsampling network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2018. 2

[35] J. Zeng, G. Cheung, M. Ng, J. Pang, and C. Yang. 3d point cloud denoising using graph laplacian regularization of a low dimensional manifold model. *IEEE Transactions on Image Processing*, 29:3474–3489, 2019. 2

[36] D. Zhang, X. Lu, H. Qin, and Y. He. Pointfilter: Point cloud filtering via encoder-decoder modeling. *IEEE Transactions on Visualization and Computer Graphics*, 27(3):2015–2027, 2020. 1, 2, 5, 6, 10, 11, 12