Point Cloud Denoising

Alfred Krister Ulvog

***Abstract*—3d data has become more prominent as accessibility to depth sensors and the demand for such has increased with technology such as autonomous cars are on the rise. 2d image models have been heavily studied and while researchers are looking into 3d data, we still lack knowledge in well-fit models relative to that on 2d images. In this project, I investigated into one of the newest methods in point cloud denoising and also implemented my denoising algorithm to gain better understanding in 3d data models, specifically, point clouds.**

I. Introduction

For this project, I investigated the feature graph learning method for point cloud denoising [3] and my own point cloud denoising algorithm. Here we assume a point cloud is corrupted by i.i.d zero-mean Gaussian noise in every dimension (x, y, and z-axes). The assumption of i.i.d. noise in all direction may not be the best assumption to make in most settings since 3d point measurement comes with different accuracy along different directions due to the nature of depth acquisition system. This assumption may be helpful in settings where depth measurements from many incident angles are stitched together and we have no prior knowledge of the angle of the incident for each point measurement.

However, the main motivation here is to develop a model for point cloud and assess it. Omnidirectional i.i.d. Gaussian usually gives us simple formulation in denoising algorithms and perhaps a good starting point for developing models for point clouds.

II. Feature Graph Learning

Here, we follow the work from [3]. The denoising is done by solving for the following objective.

$$\min_{\mathbf{X},\mathbf{M}} ||\mathbf{Y}-\mathbf{X}||_F^2 + \gamma(\mathbf{S}\mathbf{X}-\mathbf{C})^T\mathbf{L}(\mathbf{M})(\mathbf{S}\mathbf{X}-\mathbf{C}) \tag{1}$$

$\mathbf{Y}, \mathbf{X}$ are the noisy point cloud and denoised point cloud, respectively. $\mathbf{S}$ and $\mathbf{C}$ are the sampling matrix and patch center matrix which will be more explained in Section II.A. The objective includes the data-fidelity term and regularization term. The regularization that are being imposed here is signal depencent Laplacian regularization term. General form of Laplacian regularization is

$$\mathbf{z}^T\mathbf{L}\mathbf{z} \tag{2}$$

The matrix $\mathbf{L}$ is called the Laplacian matrix that represents the connectivity of the graph. The matrix $\mathbf{L}$ has the following structure.

$$\mathbf{L} = \begin{bmatrix} \sum_{j\neq 0} w_{1j} & -w_{12} & \dots & -w_{1N} \\ -w_{21} & \sum_{j\neq 1} w_{2j} & \dots & -w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ -w_{N1} & -w_{N2} & \dots & \sum_{j\neq N} w_{Nj} \end{bmatrix} \tag{3}$$

$\mathbf{L}$ can be thought of as a precision matrix in Gaussian Markov Random Field model, and similarly, the regularization term Eq. 2 is intending to minimize the high-frequency energy. Knowing the structure of $\mathbf{L}$, the regularization term can also be written as

$$\mathbf{z}^T\mathbf{L}\mathbf{z} = \sum_{\{i,j\}} w_{i,j}(z_i - z_j)^2 \tag{4}$$

$w_{i,j}$ are the weights that represents the signal correlation. These weights can be determined by

$$w_{ij} = \exp\left((\mathbf{f}_i - \mathbf{f}_j)^T\mathbf{M}(\mathbf{f}_i - \mathbf{f}_j)\right) \tag{5}$$

where $\mathbf{f}_i, \mathbf{f}_j$ are the feature vectors at index i and j. $\mathbf{M}$ is called the metric matrix. Although this metric learning method is not applied in 2d images in previous works, this could potentially have benefits for 3d point cloud denoising since point clouds have higher dimensional data and potentially be correlation in direction that could be exploited on.

The algorithm can be broken down into 4 parts: patch construction, normal estimation, metric learning, and denoising.

A. Patch Selection

Patch centers are selected by downsampling the original point cloud. The downsampling method used here is farthest point sampling [2]. In farthest point sampling method, a sample is drawn from the original set and added to the downsampled set at a time. If the original set is R, the downsampled set is S, a sample from $R\backslash S$ that is to be added to S is an argument that maximizes the following equation.

$$\max_{s_j \notin R\backslash S} \min_{s_i \in S} d(s_i, s_j) \tag{6}$$

$d(s_i, s_j)$ is a distance metric of s_i and s_j. In this paper, the distance metric used is simply an Euclidean distance. This sampling procedure attempts to choose points that are spread out from each other as much as possible.

The idea of patch similarity has been successful in 2d image models such as Non-Local Means. Fig. 1 shows an example of patch selection. As in the figure, the patches are chosen so they are overlapping to its neighbors. Same figure shows the example of graph connection. For each point in a patch, and for each neighboring patch, connectivity is established with

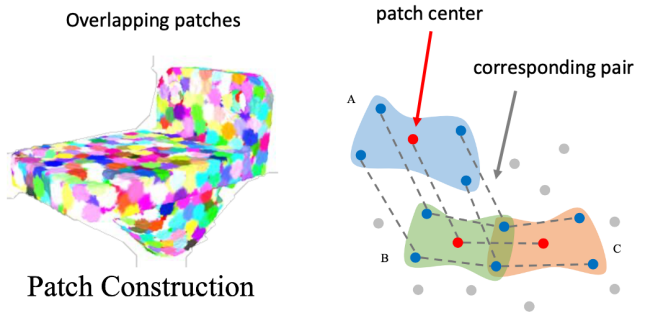

Fig. 1. Example of patch selection and graph connectivity. Patches are constructed so that they are overlapping with each other's neighbors. Graph connectivity between two points are established by minimizing the relative coordinate to respective patch centers.

one of the points in the neighboring patch. If $\mathbf{V}_m$ and $\mathbf{V}_n$ are the patches with patch center $\mathbf{c}_m$ and $\mathbf{c}_n$, respectively, and satisfy $\mathbf{c}_n \in \mathcal{N}(\mathbf{c}_m)$, the connectivity is established between $\mathbf{x}_i \in \mathbf{V}_m$ and $\mathbf{x}_j \in \mathbf{V}_n$ where the minimizes the coordinate relative to its own patch centers.

$$\min_{\mathbf{x}_j \in \mathbf{V}_n} d(\mathbf{x}_i - \mathbf{c}_m, \mathbf{x}_j - \mathbf{c}_n) \tag{7}$$

For computation reasons, we limit the number of neighbors that we search for. The resulting sparse graph will have the number of edges equal to (# patches) $\times$ (# neighboring patches) $\times$ (# points in a patch).

B. Normal Estimation

Normal of a point is computed by the eigen decomposition of the sample covariance matrix of the point and its k-nearest neighbors. The eigenvector with the smallest eigenvalue is the estimated normal for that point.

C. Metric Learning

In order to learn the metric $\mathbf{M}$, the optimization is split into two parts: optimization of diagonal components and optimization of off-diagonal components. Optimization is alternated between these two until convergence.

When learning $\mathbf{M}$, the goal is to minimize the following.

$$\begin{aligned} &\min_{\mathbf{M}} \sum_{\{i,j\}} \exp\left\{-(\mathbf{f}_i - \mathbf{f}_j)^T \mathbf{M} (\mathbf{f}_i - \mathbf{f}_j)\right\} \\ &\text{s.t. } \mathbf{M} \succ 0, \quad \text{tr}(\mathbf{M}) \leq C \end{aligned} \tag{8}$$

The constraint C on the trace is make sure the objective is not minimized by making the weights zero.

In this section, for better readability, some expression are replaced by new symbols.

$$\mathbf{g}_{i,j} = \mathbf{f}_i - \mathbf{f}_j \tag{9}$$

$$d'_{i,j} = \exp\left\{-\mathbf{g}_{i,j}^T \mathbf{M}_{2,2} \mathbf{g}_{i,j}(2:)\right\} d_{i,j} \tag{10}$$

$$d''_{i,j} = \exp\left\{-\mathbf{g}_{i,j}(1)^2 m_{1,1} - \mathbf{g}_{i,j}(2)^T \mathbf{M}_{2,2} \mathbf{g}_{i,j}(2:)\right\} d_{i,j} \tag{11}$$

The feature vector $\mathbf{f}_i$ is a concatenation of a coordinate vector $\mathbf{x}_i$ and a normal vector $\mathbf{n}_i$, but this could potentially be replaced some other features.

1) Learning Diagonal Component: The objective for the diagonal components can be simplified from Eq. 8 to

$$\begin{aligned} &\min_{\mathbf{m}} \sum_{i,j} \exp\left\{-\sum_k m_{k,k} \mathbf{g}_{i,j}(k)^2\right\} d'_{i,j} + I_S(\mathbf{m}) \\ &\text{s.t. } m_{i,i} - \sum_{j \neq i} |m_{i,j}| > 0 \ \forall i, \quad \sum_i m_{i,i} \leq C \end{aligned} \tag{12}$$

Eq. 12 can be solved by proximal gradient descent by transforming the constraint to an indicator function. Proximal Gradient descent should converge since the objective is convex although the indicator function is non-differentiable.

2) Learning Off-diagonal Component: Optimization of off-diagonal components are done by block coordinate descent and one column of off-diagonal components is optimized at a time.

$$\mathbf{M} = \begin{bmatrix} m_{1,1} & \mathbf{M}_{1,2} \\ \mathbf{M}_{2,1} & \mathbf{M}_{2,2} \end{bmatrix} \tag{13}$$

Note that $\mathbf{M}_{1,2}^T = \mathbf{M}_{2,1}$ since M is supposed to be symmetrical. When the metric matrix is broken into blocks as such and we fix $\mathbf{M}_{2,2}$, we can optimize $\mathbf{M}_{2,1}$.

$$\begin{aligned} &\min_{\mathbf{M}_{2,1}} \sum_{\{i,j\}} \exp\left\{-2\mathbf{g}_{i,j}(1) \mathbf{M}_{2,1}^T \mathbf{g}(2:)\right\} d''_{i,j} + I_S(\mathbf{M}_{2,1}) \\ &\text{s.t. } m_{i,i} - \mathbf{M}_{2,1}^T \mathbf{M}_{2,2}^{-1} \mathbf{M}_{2,1} > 0 \\ &\quad m_{1,1} \leq C - \text{tr}(\mathbf{M}_{2,2}) \end{aligned} \tag{14}$$

Similar to the optimization of Eq. 12, we can optimize Eq. 14 can be solved by Proximal Gradient Descent by transforming the constraint to an indicator function.

D. Denoising

After the metric learning from the previous section, here we attempt to denoising $\mathbf{Y}$ by constructing the Laplacian matrix from $\mathbf{M}$. We construct $\mathbf{L}$ from $\mathbf{M}$ and we freeze the Laplacian matrix to only solve for $\mathbf{X}$. By taking the derivative and decomposing $\mathbf{X}, \mathbf{Y}, \mathbf{C}$ matrices into x,y,z components, we can solve for each direction independently.

$$\begin{aligned} (\gamma \mathbf{S}^T \mathbf{L} \mathbf{S} + \mathbf{I}) \mathbf{X}_x &= \mathbf{Y}_x + \gamma \mathbf{S}^T \mathbf{L} \mathbf{C}_x \\ (\gamma \mathbf{S}^T \mathbf{L} \mathbf{S} + \mathbf{I}) \mathbf{X}_y &= \mathbf{Y}_y + \gamma \mathbf{S}^T \mathbf{L} \mathbf{C}_y \\ (\gamma \mathbf{S}^T \mathbf{L} \mathbf{S} + \mathbf{I}) \mathbf{X}_z &= \mathbf{Y}_z + \gamma \mathbf{S}^T \mathbf{L} \mathbf{C}_z \end{aligned} \tag{15}$$

E. Full Algorithm

Now that we discussed the four main parts of the algorithm, we now discuss the flow the entire algorithm. As shown in Algorithm 1, the algorithm repeats: 1. normal estimation, 2. patch selection, 3. metric learning, 4. denoising. The process is iterated for about 5 to 7 times.

Algorithm 1 Point Cloud Denoising via Feature Graph Learning

Input: noisy point cloud $\mathbf{Y}$, # patches, # points in a patch, # neighboring patches, γ
Output: denoising point cloud $\mathbf{X}$ and metric matrix $\mathbf{M}$

```
while not converged do
    Compute normals
    Construct S, C
    while not converged do
        Optimize off-diagonal M
        Optimize diagonal M
    Solve for X with learned M
```

III. My Algorithm

In this method of denoising, we estimate the denoised point based on the idea that the point should exist in the proximity of surface planes of the point's neighbors. Example is illustrated in Fig. 2.

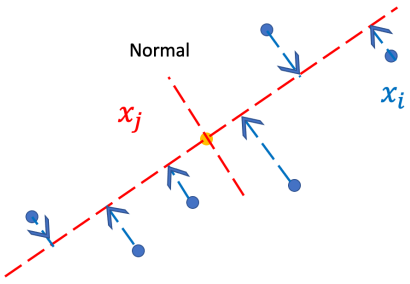

Fig. 2. Caption

For each point $\mathbf{x}_i$, we compute a descending direction $\mathbf{u}_{ij}$ which is difference vector of $\mathbf{x}_i$ and $\mathbf{x}_i$ projected onto the plane perpendicular to $\mathbf{x}_j$'s normal vector $\mathbf{n}_j$. We compute $\mathbf{u}_{ij}$ for every j such that $\mathbf{x}_i \in \mathcal{N}(\mathbf{x}_j)$. The mean descending direction $\mathbf{u}_i$ is a weighted average of $\mathbf{u}_{ij}$ where the weights are determined by a chosen distance metric between x_i and x_j.

Finally, the points are updated by weighted average of original noisy point cloud and the

$$\mathbf{x}_i \leftarrow \frac{1}{1+\lambda}\left(\frac{1}{K}\sum_{j:\ y_j \in \mathcal{N}(x_i)} \mathbf{y}_j + \lambda(\mathbf{x}_i - \mathbf{u}_i)\right) \tag{16}$$

This operation is to ensure that the denoised point cloud does not drift away too much from the original point cloud.

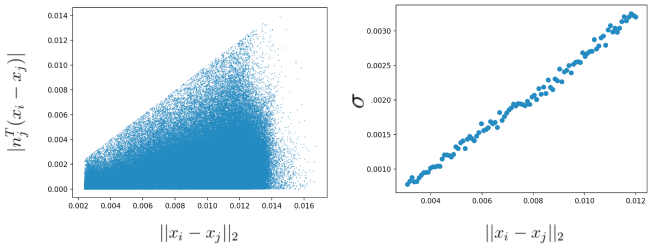

Fig. 3. The figure on the left shows the scatter plot of points with one axis for Euclidean distance of neighboring points and the other axis for Euclidan distance to the plane perpendicular to the normal of the reference point. The figure on the right shows the standard deviation of the distance to the plane at a given small range of Euclidean distance. The data was collected from the Stanford bunny model.

In addition to prevent oversmoothing, λ is chosen to be exponentially decaying value with iteration.

Weights w_{ij} applied to descending direction $\mathbf{u}_{ij}$ depends how confident the point should be projected to the plane. In the implementation, Weights are computed by

$$w_{ij} = \frac{1}{\sqrt{2\pi\alpha^2||\mathbf{x}_i - \mathbf{x}_j||_2^2}} \exp\left(-\frac{\left(\mathbf{n}_j^T(\mathbf{x}_i - \mathbf{x}_j)\right)^2}{2\alpha^2||\mathbf{x}_i - \mathbf{x}_j||_2^2}\right) \tag{17}$$

Fig. 3 shows the scatter plot of points with one axis for Euclidean distance of neighboring points and the other axis for Euclidan distance to the plane perpendicular to the normal of the reference point. The data was collected from the Stanford bunny model. Finally, the mean descending direction is computed by

$$\mathbf{u}_i = \frac{1}{\sum_{j\neq i} w_{ij}} \sum_{j\neq i} w_{ij}\mathbf{u}_{ij} \tag{18}$$

Algorithm 2 My Algorithm

Input: Noisy point cloud $\mathbf{Y}$
Output: Denoised point cloud $\mathbf{X}$

```
while not converged do
    Compute normals
    for each point x_j ∈ X do
        for each point x_i ∈ N(x_j) do
            u_ij = n_j^T (x_i − x_j) n_j
            Compute w_ij using Eq. 17
    Update x_i using Eq. 16 and Eq. 18
```

IV. Results

The error of an individual point is calculated by the euclidan distance between the point and the nearest point in the ground truth set.

The result shown in mesh is the result of poisson surface reconstruction [4]. Visually, the surface reconstruction of the denoised point cloud is better, but often times, the mean-square error of the vertices of mesh computed from

the denoised point cloud had little to no improvement or was worse than that of mesh computed from the noisy point cloud.

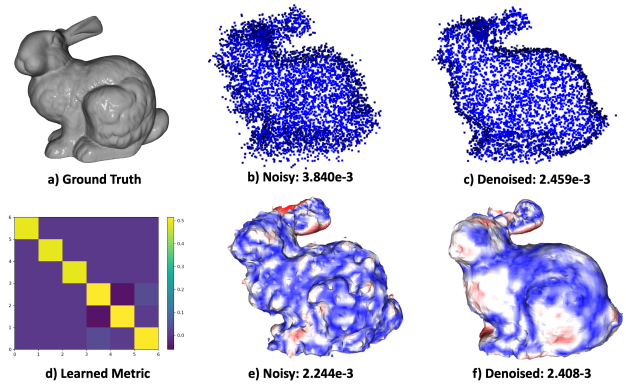

Fig. 4. Denoising of Stanford bunny using Feature Graph Learning method. The numbers is the mean-square error relative to the closet point in the ground truth. The second row shows the Poisson Surface Reconstruction [4] applied to the point cloud above. The mean-square error is calculated with the vertices of the mesh.

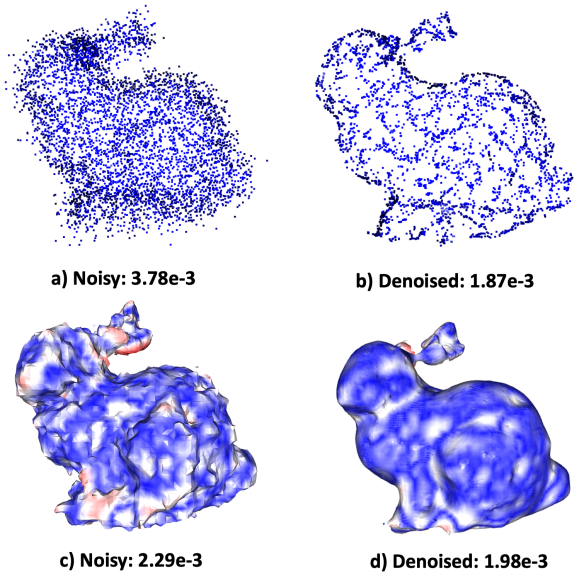

Fig. 5. Denoising of Stanford bunny using my algorithm

Although the smoothing from both algorithms improve the visuals, it comes with strong bias. In other scenarios, the denoising algorithms have no improvement at all. We also see that in the 2nd algorithms, the points move around too much and becomes coarser than the what we originally had. There must be some way to govern the resolution of the point clouds while denoising.

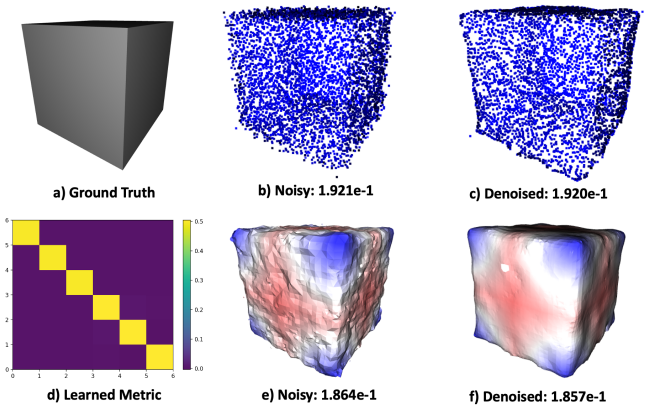

Fig. 6. Denoising of cube using Feature Graph Learning method.

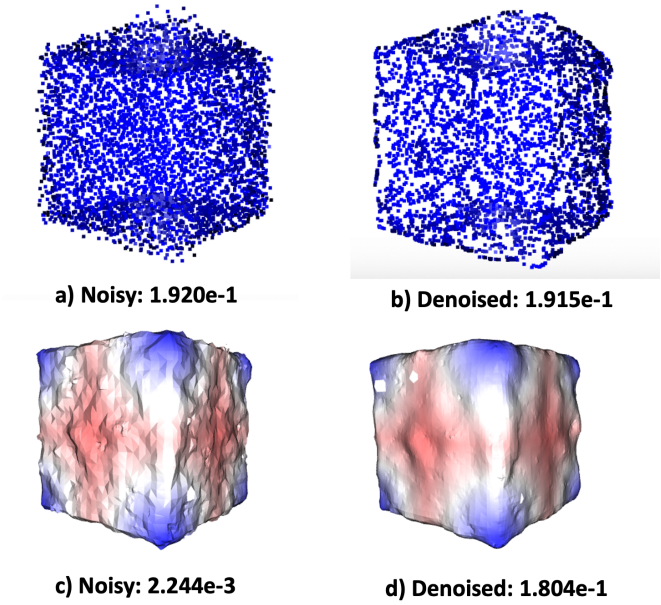

Fig. 7. Denoising of cube using my algorithm.

V. Future Improvements

More thoughts into graph connectivity construction could be important. Use registration algorithm such as Iterative Closest Point [1] to fit the rotation of neighboring patches may improve the decision on similarity of points. Better justification for metric learning is more helpful, or using higher dimensional features instead. Governing the density of point cloud during denoising process is definately helpful to prevent the resolution of the point cloud from decreasing. Lastly, joint-optimization with surface reconstruction, though more complex formulation is required, is a more reasonable approach for denoising and general enhancement of 3d data.

References

[1] P.J. Besl and Neil D. McKay. “A method for registration of 3-D shapes”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 239–256. DOI: 10.1109/34.121791.

[2] Y. Eldar et al. “The farthest point strategy for progressive image sampling”. In: *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 2 - Conference B: Computer Vision Image Processing. (Cat. No.94CH3440-5)*. 1994, 93–97 vol.3. DOI: 10.1109/ICPR.1994.577129.

[3] Wei Hu et al. “Feature Graph Learning for 3D Point Cloud Denoising”. In: *IEEE Transactions on Signal Processing* 68 (2020), pp. 2841–2856. DOI: 10.1109/TSP.2020.2978617.

[4] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. “Poisson Surface Reconstruction”. In: *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. SGP ’06. Cagliari, Sardinia, Italy: Eurographics Association, 2006, pp. 61–70. ISBN: 3905673363.