Imperial College London

Department of Mathematics

Outlier Removal for 3D Point Clouds

via ℓ_1 optimisation and enriched second-order methods

Author:

Yue Wu

Supervisor:

Dr. Estefania Loayza Romero

MSc Project

September 9, 2024

Contents

DECLARATION

I declare that the work contained in this thesis is my own work unless otherwise stated.

Acknowledgements

I would like to thank to my supervisor Dr. Estefania Loayza Romero for her guidance. I am proud to be her first student, first is always different (maybe).

Also, thanks to The Stanford 3D Scanning Repository and Romanengo et al., [1] for sharing point cloud data.

Abstract

Point clouds are crucial in fields such as 3D modeling, computer vision, and robotics. However, outliers in point clouds pose significant challenges to tasks like surface reconstruction and object detection. In this study, we propose an efficient outlier removal method based on the ℓ_1 optimisation, combined with orthantwise enriched second order method(OESOM). The outlier removal method requires minimal prior information about the point cloud, demonstrates effective outlier removal, fast convergence, and suitability for real-time applications.

1 Introduction

1.1 Point cloud and meshing

Point clouds, which are sets of spatial data points representing the geometry of objects, play a crucial role in various fields, such as computer vision, robotics, and 3D modeling [2]. Point clouds capture the surface details of objects and are frequently used for surface reconstruction by techniques such as polygon mesh [3] or spline fitting [4]. 3D modeling is essential for applications that require detailed and accurate representations of objects, including virtual reality, digital archiving, and precision engineering.

Point clouds can be obtained using active techniques that involve sensors or light-based methods, such as laser scanning. A laser scanner emits laser beams toward an object and measures the time it takes for the light to bounce back to the sensor. By calculating these time intervals, the scanner can determine the distance to different points on the object's surface, creating a dense and accurate point cloud that reflects the object's geometry [5].

An alternative method for generating point clouds is through photogrammetry, which involves capturing multiple photographs of an object from different angles. These images are then processed to estimate the 3D coordinates of points on the object's surface. Photogrammetry is popular because it is cost-effective and relatively easy to implement, especially when compared to more sophisticated hardware-based techniques like laser scanning. However, point clouds generated through photogrammetry often come with drawbacks, such as noise and outliers. This noise can arise from various factors, including inaccuracies in image matching, lighting conditions, and camera limitations [6, 7].

The presence of noise and outliers in point clouds can significantly impact the effectiveness of surface reconstruction techniques. When meshing algorithms, which are designed to create surfaces from point clouds, are applied to noisy data, they may produce inaccurate or unrealistic results. The algorithms might struggle to differentiate between genuine surface points and noise, leading to either over-smoothing (which results in loss of detail) or the creation of artifacts that do not represent the true geometry of the object. This is particularly problematic in fields where precision is paramount, as even small inaccuracies can lead to substantial errors in the final model [7, 8].

(a) one input image **(b)** point cloud **(c)** meshing of *(b)* **(d)** our filtered point cloud **(e)** meshing of *(d)*

Figure 1: A set of images showing why outlier removal is important for surface reconstruction. (c) is suffering from outliers and produces meaningless surfaces on the top, where (d) is a much more favorable result. These photos are copied from [7].

Therefore, it is essential to preprocess point clouds before applying meshing algorithms. Outlier removal helps to clean the data, ensuring that the points fed into the meshing process accurately represent the object's surface. By eliminating these erroneous points, the algorithms can more effectively create detailed and accurate mesh models. (See Figure 1).

1.2 An overview of filtering methods for point clouds

Point cloud denoising is a widely studied problem, and various approaches have been employed in past research. [9] conclude that the primary approaches of point cloud denoising can be divided into the following categories:

1.2.1 Statistical approach

Many techniques leverage the adaptability of statistical concepts. Kernel-based clustering methods calculate local likelihoods for each point, creating a probability model for noisy point clouds. Points conforming to maximum likelihood are then identified. [10] PCA, a classic information processing method, has been adapted for point cloud denoising in [11, 12]. [12] uses weighted PCA, where weights are inversely proportional to the sum of distances from each point to the mean, enhancing its effectiveness in noise reduction.

1.2.2 Neighborhood filtering approach

Neighborhood filtering approach involves using information from the point's neighbors, determining the position of a point by employing similarity measures between the point and its neighborhood. The simplest example is the constrained moving average for smoothing images and its various variants, such as Gaussian filtering, which is based on the Gaussian function. In Gaussian filtering, each point is averaged with its neighbors, with the weights determined by the distance between points and the standard deviation of the Gaussian function [9].

1.2.3 PDE approach

PDEs-based techniques for filtering point cloud can be considered as an extension of that to the triangular meshes [13]. The approach is developed from image denosing processing. The idea of anisotropic diffusion equation [14] and total variation minimization [15] used in image denosing, was then extended to point cloud denoising in [16].

1.2.4 Projection-based approaches

The core idea of projection methods is to define a projection operator that projects points onto a point cloud. The earliest research in this area originated from Moving-least-squares (MLS) approaches [17]. MLS works by fitting local surfaces to neighborhoods around each point in the cloud using weighted least squares. The original points are then projected onto these locally fitted surfaces, resulting in a smoother, noise-reduced point set. Another project approach method is local optimal projection operator(LOP) that works by projecting points onto locally optimal surfaces [18]. For each point in the cloud, it first determines a local neighborhood, typically using k-nearest neighbors or fixed-radius search. Within this defined neighborhood, it estimates a best-fitting local surface, which can be a plane, quadratic surface, or higher-order surface. The original point is then orthogonally projected onto this estimated local surface to obtain a new point position.

1.3 ℓ_p norms and sparse solution

One statistical approach is using sparsity of outliers. Outliers are data points that deviate significantly from the majority data distribution. By definition, outliers are sparse. a non sparse cluster of data point would be considered part of the data distribution. In optimisation problems, adding penalties are techniques used to encourage certain behaviors in the solution space by adding penalty terms. ℓ_p norms are commonly used for penalty terms, defined as

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

where $\mathbf{x} = (x_1, x_2, x_3, \cdots, x_n)$. The ℓ_0 norm counts the number of non-zero elements in a vector, so adding an $\|\mathbf{x}\|_0$ penalty term will lead to sparse solutions, as we aim to maximize the number of zero entries in our solution. However, the ℓ_0 norm is non-differentiable and non-convex, posing challenges for optimisation algorithm design. [19] state ℓ_1 norm is often considered a convex relaxation of the ℓ_1 norm and also imposes sparse solutions (See Fig. 3).

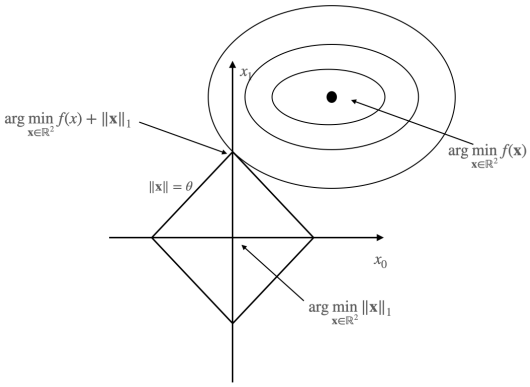


Figure 2: Geometric interpretation of ℓ_1 regularization: level curves of $f(x)$ intersecting the ℓ_1 norm constraint $\|x\|_1 = \theta$, showing the regularized solution at the diamond vertex, promoting sparsity.

A two-dimensional iso-surfaces plot can intuitive present why ℓ_1 norm encourages sparse solution. Suppose we aim to optimise $f(\mathbf{x}) + \|\mathbf{x}\|_1 | x \in \mathbb{R}^2$. As shown in Figure 2., due to the shape of norm-1 circle, the global minimiser is most likely to sit on the

full x_1 or x_0, leaving another entry to be 0.

Many researchers have conducted methods for outlier detection in point clouds using sparsity and ℓ_1 norm. [20] proposed a fast graph-based local algorithm to address the point cloud denoising problem efficiently. [21] introduced a new model that combines ℓ_1-median filtering with sparse ℓ_1 regularization to denoise normal vectors and preserve sharp features in point clouds. This approach aims to reconstruct and smooth point clouds effectively by utilizing ℓ_1 regularization techniques which have been shown to be effective in denoising point clouds and preserving sharp features.

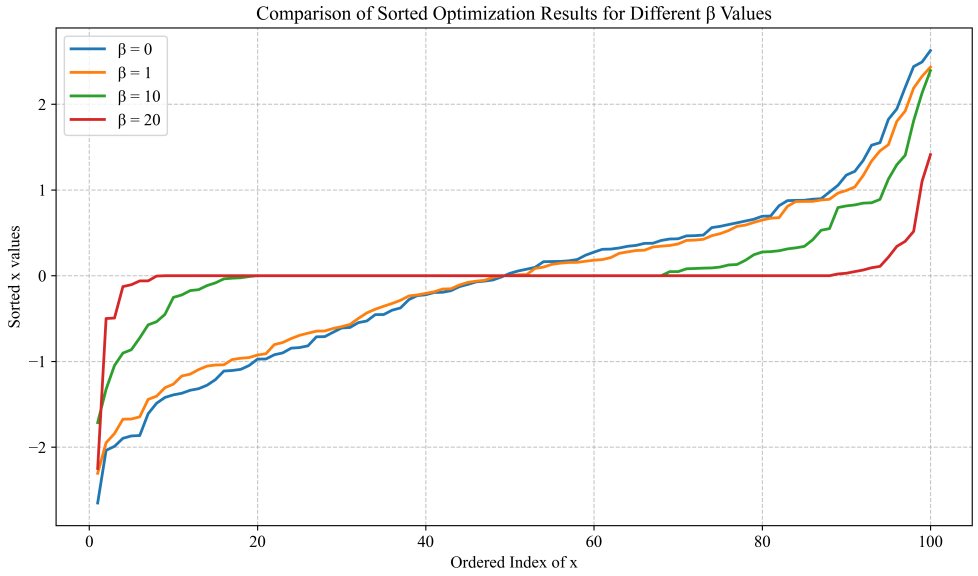


Figure 3: Ordered result of $\arg\min_x \exp(\cos(x-y))^2 + \beta\|x\|_1$ where $x, y \in \mathbb{R}^100$ and y is a fixed random vector. As β increases, the sparsity of solution increases (more 0 entries).

In this study, we will use the ℓ_1 norm to promote sparse solutions for outliers, formulate an optimisation problem, and solve it using the orthantwise enriched second-order method (OESOM) proposed by Dr. Estefania Loayza Romero [22], which will be introduced in next chapter.

2 Optimisation

optimisation is a key concept in mathematics and computer science that involves identifying the best solution from a range of possible options. Mathematically, this can be represented as:

$$\min_{x \in \mathscr{D}} f(x)$$

where $f(x)$ is the objective function to be minimized, x represents the decision variable, and $\mathscr{D}$ is the feasible set or domain of the problem. The aim of optimisation is to determine the values of x that minimize (or maximize) the objective function within the feasible set.

Unconstrained optimisation problem is a kind of optimisation problem where the feasible set $\mathscr{D}$ is the entire space. Many powerful algorithms has been developed for solve unconstrained optimisation. All algorithms for unconstrained minimization necessitate that the user provides an initial starting point, typically represented as x_0. Starting from x_0, optimisation algorithms produce a series of updates, denoted as $\{x_k\}$ from $k = 0$ to infinity, that continue until no further improvements can be achieved or it appears that a solution has been closely approximated. The decision to progress from one point x_k to the next involves utilizing data about the function f evaluated at x_k. This information helps to identify the next point x_{k+1}, which ideally has a lower value of the function f than x_k [23]. Newton's method (2nd order) and gradient descent (1st order) and their variants are the most commonly used unconstrained optimisation algorithms, with Newton's method converging faster but being more computationally difficult than gradient descent.

Most unconstrained algorithms rely on derivatives to guide the next step, however, the ℓ_1 norm is non-differentiable, making conventional methods inapplicable to ℓ_1 optimisation. However, using the fact that directional derivative of ℓ_1 always exists, it is feasible to implement ℓ_1 optimisation algorithms. In the following chapters, I will introduce from scratch a specialized method for optimising with the ℓ_1 norm. It employs a second order approach and confines the solution to a specific region to avoid issues with the non-differentiability of the ℓ_1 norm, then integrates additional techniques to achieve optimisation.

2.1 Second order unconstrained optimisation

2.1.1 Basic idea

Newton's method and it's variant is a technique that leverages the second-order derivative information of the objective function to accelerate the optimisation process. The core idea is to replace the original function with a quadratic approximation near the current point, and then find the optimal solution within this approximated model.

2.1.2 Quadratic approximation

For objective function $f(\mathbf{x})$, consider its second-order Taylor expansion at a certain point $\mathbf{x}_k$. The function can be approximated as:

$$f(\mathbf{x}) \approx f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top (\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^\top \mathbf{H}_k (\mathbf{x} - \mathbf{x}_k)$$

where:

- $\nabla f(\mathbf{x}_k)$ is the gradient of $f(\mathbf{x})$ at point $\mathbf{x}_k$.
- $\mathbf{H}_k$ is the Hessian matrix of $f(\mathbf{x})$ at point $\mathbf{x}_k$.

The goal of Newton's method and it's variant is to find the point that minimizes this quadratic approximation.

2.1.3 Search direction

To find the point $\mathbf{x}$ that minimizes quadratic approximation state above, take the derivative of the above expression and set it to zero:

$$\nabla f(\mathbf{x}_k) + \mathbf{H}_k (\mathbf{x} - \mathbf{x}_k) = 0$$

By rearranging, we get:

$$\mathbf{H}_k (\mathbf{x} - \mathbf{x}_k) = -\nabla f(\mathbf{x}_k)$$

Letting $\mathbf{d}_k = \mathbf{x} - \mathbf{x}_k$, we obtain the linear system used in Newton's method:

$$\mathbf{H}_k \mathbf{d}_k = -\nabla f(\mathbf{x}_k)$$

Solving this linear system gives us the search direction $\mathbf{d}_k$. This search direction is the steepest direction of quadratic approximation, providing a solid base for fast convergence. If we strictly move by $\mathbf{d}_k$, i.e., step size = 1, we move to the minimal of quadratic approximation. However, when the quadratic model is not an accurate representation of the function, we need to consider step size more carefully to avoid problems such as overshoot.

2.1.4 Step size

Step size decides how far to move in $\mathbf{d}_k$. The update formula for each iteration is

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$$

The goal of line search is to find a suitable step size α_k such that the objective function $f(\mathbf{x})$ at the new point $\mathbf{x}_{k+1}$ decreases significantly, while ensuring the stability and convergence of the algorithm. Ideally, we want the step size α_k to meet the following two objectives:

- Function value decrease: The step size should ensure that the value of the objective function $f(\mathbf{x})$ decreases significantly, ensuring progress toward the minimum of the objective function.
- Step size should not be too small or too large: If step size is too small, we get small updates, requiring more iterations. A too-large step size may cause the search to overshoot the "effective region" of the objective function, potentially increasing the function value or even causing the algorithm to diverge.

The most trivial way is to find the global minimizer along the search direction, so called exact line search.

Exact line search

The goal of exact line search is to find the step size α_k that minimizes the objective function along the direction $\mathbf{d}_k$. Mathematically, this can be expressed as:

$$\alpha_k = \arg\min_{\alpha \geq 0} f(\mathbf{x}_k + \alpha \mathbf{d}_k)$$

However, exact line search is not common in most practical applications because it requires solving a separate optimisation problem, which is often very complex and computationally expensive (See [24] chapter 3).

Inexact line search

Due to the high computational complexity of exact line search, inexact line search is used more commonly in practice. Inexact line search identify a step length that achieves sufficient reductions rather than fully minimizing it. One commonly used inexact line search search finding a step size which achieves Armijo condition, also known as sufficient decrease condition(or Wolfe condition). The Armijo condition requires that the step size α_k ensures sufficient decrease in the objective function, specifically:

$$f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) \leq f(\mathbf{x}_k) + c_1 \alpha_k \nabla f(\mathbf{x}_k)^\top \mathbf{d}_k$$

where c_1 is a small positive constant e.g. 10^{-4}. To find α_k that satisfies Armijo condition, procedure called back-tracking is often used. Backtrack starts with an initial estimate for the step size, and checks if the Armijo condition is satisfied. If the condition is not met, the candidate α is reduced by multiplying it with a reduction factor such as 0.9. This process is repeated iteratively until the Armijo condition is met. In Newton's method, 1 is often used as initial guess of α as it leads to the minimum of the quadratic approximation.

Finally, update $\mathbf{x}_k$ by $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$ and terminate the algorithm when termination condition meets, such as $\nabla f(\mathbf{x}_k) < t$ where t stands for tolerant, a hyper-parameter controls precision and running time of algorithm.

2.2 ℓ_1 optimisation

To optimisation function with ℓ_1 penalty

$$\min_x f(x) + \beta \|x\|_1,$$

Although the ℓ_1 norm is non-differentiable, when restricted by the set of points that never change sign, it becomes differentiable. On those sets of points, the derivative of the ℓ_1 part is constant, and hence its second derivative is zero. This means that the second-order behavior is controlled only by the $f(x)$ part [25]. Therefore, we can find the set of points that never change sign, i.e., an orthant, and optimise $f(x) + \beta \|x\|_1$ along the orthant.

Thus, when optimising $f(x) + \beta \|x\|_1$, the following strategy can be used:

1. Construct a quadratic approximation: Find an orthant that contains the current point, meaning a region in space where the ℓ_1 regularization term remains linear. Since the ℓ_1 regularization term is linear within the orthant, it does not affect the second-order derivative of the objective function.

2. Ensure that the quadratic approximation is valid within a specific orthant that includes the current point.

3. Then search in the direction of the minimum of the quadratic.

4. Make sure that the search is always restricted to the currently selected orthant. This is because the quadratic approximation is only valid within this orthant. If the search crosses into another orthant, the behavior of the ℓ_1 regularization term will change, causing the original quadratic approximation to become inaccurate.

2.3 Orthantwise Enriched Second-order Method (OESOM)

OESOM is an algorithm designed specific for optimising problem with form of $\min_x g(x)$ where $g(x) = f(x) + \beta \|x\|_1$. OESOM is proposed by Dr. Estefania Loayza Romero in [22].

Although $g(x)$ is non-differentiable, it's first order information can still be used by defining a subgradient. A vector $\mathbf{d}$ is a subgradient of a convex function f at $\mathbf{w}$ if

$$f(\mathbf{v}) \geq f(\mathbf{w}) + \mathbf{d}^\top (\mathbf{v} - \mathbf{w}), \quad \forall \mathbf{v}$$

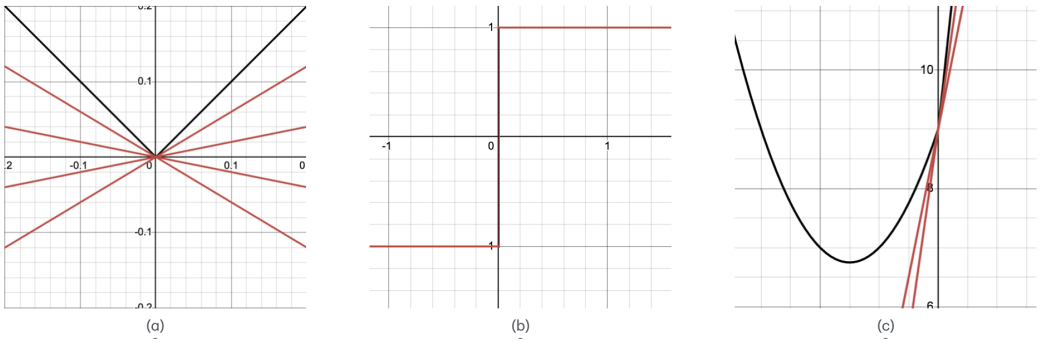


Figure 4: Set of images for illustrating of sub-gradients where (a) presents $y = \|x\|_1$ (black curve) and it's possible sub-gradients at x = 0 (red lines), (b) shows sub-gradients of $y = \|x\|_1$ and (c) shows $y = (x+3)^2 + 3\|x\|_1$ (black curve) and it's possible sub-gradients at x = 0 (red lines).

This definition comes from the intuition that differentiable convex functions are always above their tangent lines (or planes in higher dimensions). Although ℓ_1 is non-differentiable where $x^i = 0$, it has a sub-gradient where

$$\frac{d\|\mathbf{x}\|_1}{dx^i} = \begin{cases} 1 & \text{if } x^i > 0 \\ -1 & \text{if } x^i < 0 \\ [-1,1] & \text{if } x^i = 0 \end{cases}$$

For optimising g(x), we need to choose one sub-gradient to be the steepest descent direction to find search direction, we choose sub-gradient to be

$$\widetilde{\nabla}_i g(x) = \begin{cases} \nabla_i f(x) + \beta \operatorname{sign}(x_i) & \text{if } x_i \neq 0, \\ \nabla_i f(x) + \beta & \text{if } x_i = 0 \text{ and } \nabla_i f(x) < -\beta, \\ \nabla_i f(x) - \beta & \text{if } x_i = 0 \text{ and } \nabla_i f(x) > \beta, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Similarly, define orthant directions to be

$$z_i(x) = \begin{cases} \operatorname{sign}(x_i) & \text{if } x_i \neq 0, \\ 1 & \text{if } x_i = 0 \text{ and } \nabla_i f(x) < -\beta, \\ -1 & \text{if } x_i = 0 \text{ and } \nabla_i f(x) > \beta, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The orthant defined by z(x) is

$$\Omega := \{d : \text{sign}(d) = \text{sign}(z(x))\},$$

and corresponding orthogonal projection

$$P(y)_i = \begin{cases} y_i & \text{if sign}(y_i) = \text{sign}(z_i(x)), \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

This particular orthant was chosen because it is steepest descent direction and the element of the gradient of the ℓ_1 norm with smallest norm [26]. And, it measures how much information the smooth part contributing to the descent direction and only if it is large enough takes a step in that direction.

With orthant, orthogonal projection and sub-gradient defined, together with strategy above, a Newton like method for optimising g(x) can be constructed. However, as we are interested in large number of 0 entries, we can use use Huber regularisation to extract some second order information on the ℓ_1 norm to promote sparse solution. Huber regularisation, also known as smooth ℓ_1 is a relaxation of ℓ_1 norm by smoothing non-differential part, defined as $\|x\|_{1,\gamma} := \sum_{i=1}^{m} h_\gamma(x_i)$, where

$$\gamma(x_i) = \begin{cases} \gamma \frac{x_i^2}{2} & \text{if } |x_i| \leq \frac{1}{\gamma}, \\ |x_i| - \frac{1}{2\gamma} & \text{if } |x_i| > \frac{1}{\gamma}. \end{cases} \tag{4}$$

The Hessian of Huber regularisation is a diagnoal matrix and can be compute as follows

$$\Gamma_{ii} = \begin{cases} \gamma & \text{if } \gamma|x_i| \leq 1, \\ 0 & \text{elsewhere.} \end{cases} \tag{5}$$

Combining orthant information and Huber regularisation, search direction d^k can be found by

$$\left(B^k + \beta\Gamma^k\right) d^k = -\tilde{\nabla} g\left(x^k\right). \tag{6}$$

Where B is second order information of f(x), such as Hessian or quasi- Newton approximation(BFGS for example). Step size s^k can be find using Armijo condition on orthant direction, i.e.,

$$g\left[P\left(x^k + s_k d^k\right)\right] \leq g\left(x^k\right) + \tilde{\nabla} g\left(x^k\right)^T \left[P\left(x^k + s_k d^k\right) - x^k\right] \tag{7}$$

Algorithm 1 Orthantwise Enriched Second Order Method (OESOM)

1: Initialize x^0 and B^0.

2: **repeat**

3: Compute the matrix Γ^k.

4: Compute the pseudo-gradient $\widetilde{\nabla} g(x^k)$.

5: Compute search direction d^k by solving $\left(B^k + \beta\Gamma^k\right) d^k = -\tilde{\nabla} g\left(x^k\right)$.

6: Compute search step s^k by backtracking

$$g\left[P\left(x^k + s_k d^k\right)\right] \leq g\left(x^k\right) + \tilde{\nabla} g\left(x^k\right)^T \left[P\left(x^k + s_k d^k\right) - x^k\right]$$

7: Compute next position

$$x^{k+1} = P(x^k + s_k d^k),$$

8: Update the matrix B^k.

9: $k \leftarrow k+1$.

10: **until** stopping criteria is satisfied

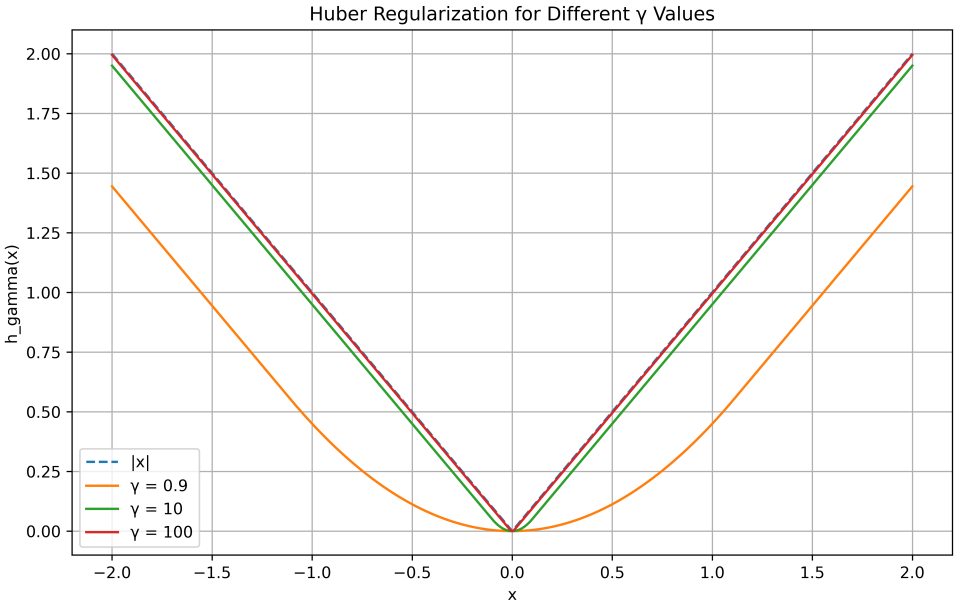


Figure 5: Plot of Huber Regularization under different γ and $\|x\|_1$.

3 Point cloud outlier removal via ℓ_1 optimisation

3.1 Problem formulation

As outlined in the introduction, a point cloud is a collection of 3D coordinates where the information for each point is encoded based on its position or the normal vector of its neighboring surface. In this section, I will discuss how to leverage the sparsity of outliers, in conjunction with the information encoded for each point, to effectively perform outlier removal.

Consider a noisy point cloud Y consisting of n points, where $Y = [y_1, y_2, y_3]$ and $y_i \in \mathbb{R}^n$. Each entry in y_i represents the value of the i-th dimension of a data point. We set Y to be composition of "real" data $X = [x_1, x_2, x_3]$ and outlier $O = [o_1, o_2, o_3]$ is degree of perturbation of X, i.e, $Y = X + O$.

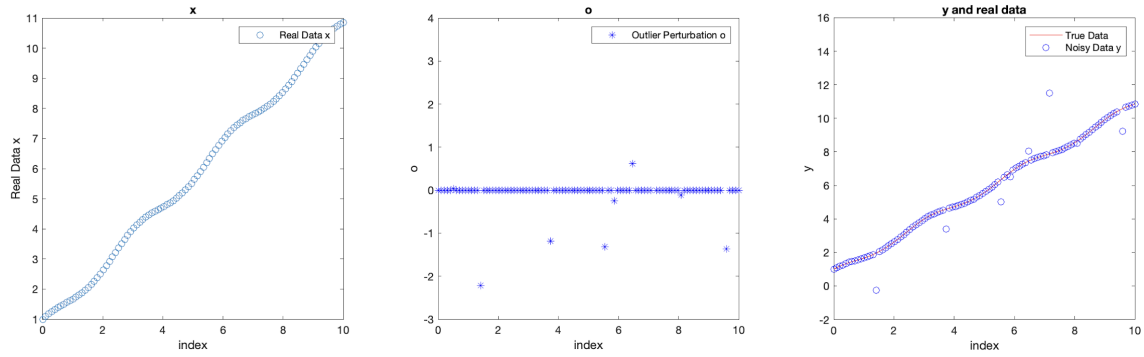


Figure 6: A one dimensional data for illustrating problem formulation, where observed data $Y = X + O$, if a data is not outlier, then it's corresponding O entry should be 0.

To recover Y using X and O, we aim to use the sparsity of the outliers by encouraging a sparse solution for O. This can be achieved by solving the following optimisation problem:

$$\min_{X,O} \|X + O - Y\|_2^2 + \beta \|O\|_1 \tag{8}$$

Here, the term $\|X + O - Y\|_2^2$ is referred to as the fidelity term, which ensures that the solution closely resembles the original data. The parameter β controls the sparsity of the outlier O. A larger value of β promotes greater sparsity, meaning that fewer

outliers are identified. However, solving (8) will returns $Y = X$ and $O = \mathbf{0}$, the all zero vector, as $\mathbf{0}$ is the most sparse solution and we are achieving global minimal here. To avoid this trivial solution, we can introduce a penalty term for X, enforcing a specific structure on X, In this study, we achieve this by constraining the distances within its neighborhood. In order to constraining the distances, first find k-nearest neighbourhood for all data points and construct k $n \times n$ matrix L^l by the following:

$$L^l(i,j) = \begin{cases} 1 & \text{if } i = j, \\ -1 & \text{if } j \text{ is the } l\text{-th nearest neighbor of } i, \\ 0 & \text{otherwise.} \end{cases}$$

$L^l x$ will return difference between current point and l-th nearest neighborhood. Combining data fidelity term, sparsity term and smoothness term together, we obtained the following optimisation problem:

$$\min_{X,O} \|X + O - Y\|_2^2 + \alpha \sum_{l=1}^{k} \|(L^l X)\|_2^2 + \beta \|O\|_1, \tag{9}$$

where X, Y, and O are matrices in $\mathbb{R}^{n\times 3}$, representing the original data, observed data, and the noise or outliers.

To make the objective function more specific and to avoid any potential misunderstanding that might arise from the use of matrix norms, the problem formulation is refined as follows:

Consider the dataset with outliers $Y = [y_1, \ldots, y_d]$ where $Y \in \mathbb{R}^{n\times d}$, each data point has d-dimensions, and n is the number of data points. The objective is to identify and remove outliers by solving the optimisation problem

$$\min_{\mathbf{X},\mathbf{O}} \Phi(\mathbf{X}, \mathbf{O}) \tag{\textbf{P}}$$

$$\Phi(\mathbf{X}, \mathbf{O}) = \sum_{i=1}^{n} [\|x_i + o_i - y_i\|_2^2 + \alpha \sum_{l=1}^{k} \|(L^l x_i)\|_2^2 + \beta \|o_i\|_1], \tag{10}$$

where $X = [x_1, \ldots, x_d]$ and $O = [o_1, \ldots, o_d]$, $X, O \in \mathbb{R}^{n\times d}$ are matrices representing the estimated true data points and the outliers respectively. $\alpha, \beta \in \mathbb{R}_{\geq 0}$ are parameters controlling smoothness and sparsity of solution respectively. Each vector x_i and $o_i \in \mathbb{R}^n$ corresponds to the d-th dimension data in $\mathbf{X}$ and $\mathbf{O}$. In this study, our primary focus is on the removal of outliers within 3D point clouds, hence $d = 3$ unless otherwise specified.

3.2 Optimality condition

First we need to check existence of solution. $\Phi(\mathbf{X},\mathbf{O})$ is lower bonded by 0 as it's sum of non-negative terms. $\Phi(\mathbf{X},\mathbf{O})$ is coercive as for every possible direction, fidelity term will tends to infinity, rest two terms are non-negative. Coerciveness guarantees attainment of global optima. Furthermore, $\Phi(\mathbf{X},\mathbf{O})$ is convex as ℓ_p norm is convex with $p \geq 1$ and sum of convex functions is convex function. Convexity combined with first order optimality condition is sufficient for global optimality.
Using Fermat's condition, the first order optimality condition of (**P**) is as follows:

$$\begin{cases} 2(x_i^j + o_i^j - y_i^j) + 2\alpha \sum_{l=1}^{k} (L^l)^T L^l x_i^j = 0 & \text{for all } i,j \\ 2(x_i^j + o_i^j - y_i^j) + \beta \operatorname{sgn}(o_i^j) = 0 & \text{for all } i,j \mid o_i^j \neq 0 \\ 0 \in 2(x_i^j + o_i^j - y_i^j) + \beta \operatorname{sign}(o_i^j) & \text{for all } i,j \mid o_i^j = 0 \end{cases}$$

where $\operatorname{sgn}(o_i^j)$ is defined as:

$$\operatorname{sgn}(o_i^j) = \begin{cases} 1 & \text{if } o_i^j > 0 \\ -1 & \text{if } o_i^j < 0 \\ [-1,1] & \text{if } o_i^j = 0 \end{cases}$$

Here, when $o_i^j = 0$, as long as $0 \in [2(x_i^j + o_i^j - y_i^j) + \beta, 0 \in 2(x_i^j + o_i^j - y_i^j) + \beta]$, **O** can be considered as optimal solution.

3.3 Numerical solution

The idea of OESOM introduced in section 2.3 is used to optimise objective function. Due to the multi-variable nature of objective function, there are slight modifications in the implementation; however, the core principles remain unchanged. In order to pre-form the method, we first need to find Hessian and gradient (pseudo-gradient). Recall pseudo derivative defined in (1), the partial (pseudo) derivative of each variables are

$$\frac{\partial \phi}{\partial x_i} = 2(x_i + o_i - y_i) + 2\alpha \sum_{l=1}^{k} (L^l)^T L^l x_i,$$

$$\frac{\partial\tilde{\phi}}{\partial o_i}=\begin{cases}2(x_i+o_i-y_i),+\beta\text{sign}(o_i) & \text{if } o_i\neq 0,\\ 2(x_i+o_i-y_i)+\beta & \text{if } o_i=0 \text{ and } 2(x_i+o_i-y_i)<-\beta,\\ 2(x_i+o_i-y_i)-\beta & \text{if } o_i=0 \text{ and } 2(x_i+o_i-y_i)>\beta,\\ 0 & \text{otherwise.}\end{cases} \tag{11}$$

and pseudo gradient is

$$\tilde{\nabla}\phi(\mathbf{x},\mathbf{o})=\begin{bmatrix}2(x_1+o_1-y_1)+2\alpha\sum_{l=1}^{k}(L^l)^TL^lx_1\\ 2(x_2+o_2-y_2)+2\alpha\sum_{l=1}^{k}(L^l)^TL^lx_2\\ 2(x_3+o_3-y_3)+2\alpha\sum_{l=1}^{k}(L^l)^TL^lx_3\\ \frac{\partial\tilde{\phi}}{\partial o_1}\\ \frac{\partial\tilde{\phi}}{\partial o_2}\\ \frac{\partial\tilde{\phi}}{\partial o_3}\end{bmatrix}. \tag{12}$$

The Hessian combined with Huber regularization is

$$\mathbf{H}_x=2\mathbb{I}_n+2\alpha\sum_{l=1}^{K}L^{l\top}L^l$$

$$\tilde{\mathbf{H}_\phi}=\begin{bmatrix}\mathbf{H}_x & 0 & 0 & 2\mathbb{I} & 0 & 0\\ 0 & \mathbf{H}_x & 0 & 0 & 2\mathbb{I} & 0\\ 0 & 0 & \mathbf{H}_x & 0 & 0 & 2\mathbb{I}\\ 2\mathbb{I} & 0 & 0 & 2\mathbb{I}+\Gamma_{o_1} & 0 & 0\\ 0 & 2\mathbb{I} & 0 & 0 & 2\mathbb{I}+\Gamma_{o_2} & 0\\ 0 & 0 & 2\mathbb{I} & 0 & 0 & 2\mathbb{I}+\Gamma_{o_3}\end{bmatrix}. \tag{13}$$

Where Γ_{o_i} is seconded order information of Huber regularization, can be calculated using (5). The Hessian (13) can be further simplified to

$$\mathbf{H}_\phi=\begin{bmatrix}A & 2\mathbb{I}_{3n}\\ 2\mathbb{I}_{3n} & B\end{bmatrix}, \tag{14}$$

where $A=diag\{\mathbf{H}_x,\mathbf{H}_x,\mathbf{H}_x\}$ and $B=diag\{2\mathbb{I}+\Gamma_{o_1},2\mathbb{I}+\Gamma_{o_2},2\mathbb{I}+\Gamma_{o_3}\}$. Search direction d^k can be found by solving the linear system

$$-\mathbf{H}_\phi d^k=\tilde{\nabla}\phi(x,o), \tag{15}$$

where

$$d^k=\begin{bmatrix}d_x^k\\ d_o^k\end{bmatrix},\nabla\phi(x,o)=\begin{bmatrix}\nabla_x\phi(x,o)\\ \tilde{\nabla}_o\phi(x,o)\end{bmatrix}$$

Where d_x^k and $d_o^k \in \mathbb{R}^{\text{ⱶ}\times\text{ⱪ}}$ and first n entries correspond to the search direction for the first coordinate and so forth.

To accelerate the calculation, [27] state that for a 2×2 block matrix

$$M = \begin{bmatrix} P & Q \\ R & S \end{bmatrix},$$

where P, Q, R, and S are non-singular, the inverse of M is given by:

$$M^{-1} = \begin{bmatrix} (M/S)^{-1} & (M/Q)^{-1} \\ (M/R)^{-1} & (M/P)^{-1} \end{bmatrix}. \tag{16}$$

Where $M/S = P - QS^{-1}R, M/Q = R - PQ^{-1}S, M/R = R - SQ^{-1}P, M/P = S - RP^{-1}Q$. For (14), Q and R are obviously non-singular. For A, first consider $\mathbf{H}_x$. $(L^l)^T L^l$ is a Gramian matrix hence it's always positive semi-definite. Hence $\mathbf{H}_x$ is non-singular as positive definite + positive semi-definite results in positive definite hence invertible. Which implies A being invertible as block diagonal matrices is invertible iff each blocks are invertible. B is also non-singular as it's diagonal matrix with all positive entries.

Apply (16) to (14), we obtained the following

$$\mathbf{H}_\phi^{-1} = \begin{bmatrix} (A - 4B^{-1})^{-1} & (2\mathbb{I} - \frac{1}{2}AB)^{-1} \\ (2\mathbb{I} - \frac{1}{2}BA)^{-1} & (B - 4A^{-1})^{-1} \end{bmatrix}, \tag{17}$$

Hence

$$d_x^k = -(A - 4B^{-1})^{-1}\nabla_x\phi(x,o) - (2\mathbb{I} - \frac{1}{2}AB)^{-1}\tilde{\nabla}_o\phi(x,o), \tag{18}$$

$$d_o^k = -(2\mathbb{I} - \frac{1}{2}BA)^{-1}\nabla_x\phi(x,o) - (B - 4A^{-1})^{-1}\tilde{\nabla}_o\phi(x,o). \tag{19}$$

For the step size, we need to restrict o_i in the orthant but not x. s_{oi}^k can be found by backtracking

$$\phi\left(\mathbf{X}, \mathbf{O}\setminus o_i, P\left(o_i^k + s_{oi}^k d_{oi}^k\right)\right) \leq \phi\left(\mathbf{X},\mathbf{O}\right) + \nabla\phi\left(\mathbf{X}^k,\mathbf{O}^k\right)^T\left[P\left(o_i^k + s_{oi}^k d_{oi}^k\right) - o_i^k\right] \tag{20}$$

Where $P(.)$ is orthogonal projection define in (3). For $s_{x_i}^k$, backtracking

$$\phi(x_i + s_{xi}^k d_{xi}^k, \mathbf{X}\setminus x_i, \mathbf{O}) \leq \phi\left(\mathbf{X},\mathbf{O}\right) + \nabla\phi\left(\mathbf{X}^k,\mathbf{O}^k\right)^T s_{x_i}^k d_{xi}^k \tag{21}$$

Algorithm 2 Point cloud outlier removal via OESOM

1: Initialize x_i^0, o_i^0.

2: Build L^l matrix based on KNN search

3: **repeat**

4: Compute gradient using (12)

5: Compute search direction d^k using (18), (19)

6: Compute search step s^k by backtracking (20), (21)

7: Compute next position with

$$o_i^{k+1} = P(o_i^k + s_{oi}^k d_{oi}^k),$$

$$x_i^{k+1} = x_i^k + s_{xi}^k d_{xi}^k,$$

8: $k \leftarrow k+1$.

9: **until** stopping criteria is satisfied

4 Result and discussion

The experiment introduces outliers to the true data by randomly selecting a proportion of data points. For each selected point, random noise is generated using a normal distribution with zero mean and increased variance. This noise is then added to the original coordinates of the selected points, creating outliers. The process ensures that the number and position of outliers are controlled, the proportion of outliers added in this study is 1% if not specified, all result can be duplicate with random seed = 42.

The Alpha Shape algorithm is employed to generate a 3D surface model from point cloud data. This technique creates a surface by connecting points in the cloud to form triangles, effectively capturing the object's shape and features. The level of detail in the resulting model can be adjusted by modifying the alpha parameter [28]. In MATLAB, the built-in "alphaShape" function is utilized to execute this process, with the alpha value being automatically determined by default.

All experiments were conducted on a MacBook Air equipped with an Apple M1 chip and 16 GB of RAM. The code was implemented in MATLAB 23.2.0.2515942 (R2023b) Update 7. The complete source code are provided in Appendix B.

4.1 Effectiveness of point cloud outlier removal via ℓ_1 optimisation

Fig 7. shows an result of 7.5k point cleaning with 0.1% outliers. In this example, without parameter tuning, the algorithm effectively cleaned the data and gives a reasonable result on surface reconstruction. The algorithm not only effectively cleans the data but also accurately identifies the positions of outliers (See Fig 8.).

Fig 9. demonstrates the algorithm's performance on a more complex shape. If the initial outlier detection is not satisfactory, the denoising process can be performed iteratively. While repeated application of the algorithm can aid in noise reduction, it may also result in some loss of detail and a sparser point cloud.

As size of point cloud expand, the algorithm exhibits reduced parameter sensitivity. This phenomenon is primarily attributed to the distance-based nature of penalty calculations, which allows for more flexible parameter tuning in spatially extensive point cloud datasets.

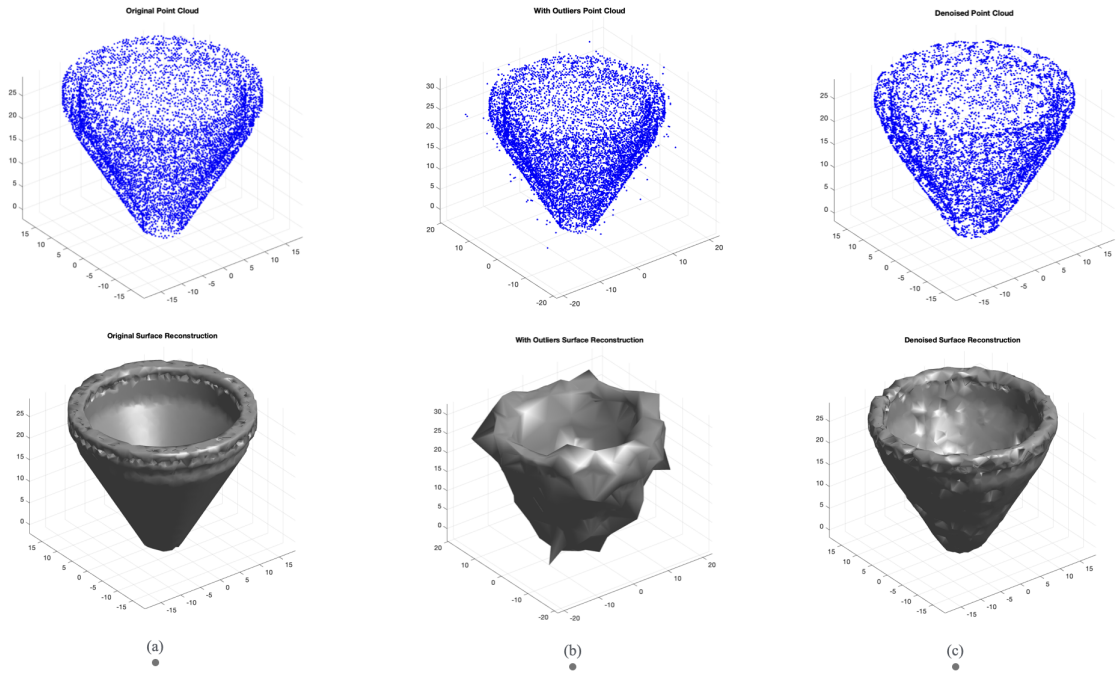


Figure 7: Point cloud denoising and surface reconstruction comparison. (a) Original point cloud and its surface reconstruction(7.5k points). (b) Point cloud with added outliers and resulting distorted surface reconstruction. (c) Denoised point cloud using the proposed algorithm and its improved surface reconstruction, demonstrating effective noise removal and shape preservation. The setting is $\alpha = 0.1$, $\beta = 0.001$, K = 5

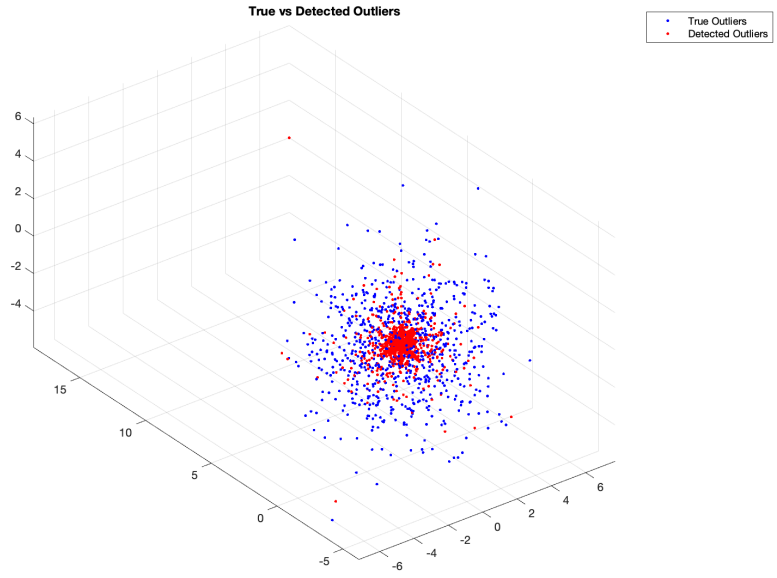


Figure 8: Comparison of detected outliers(Red) and artificially added outliers(Blue).

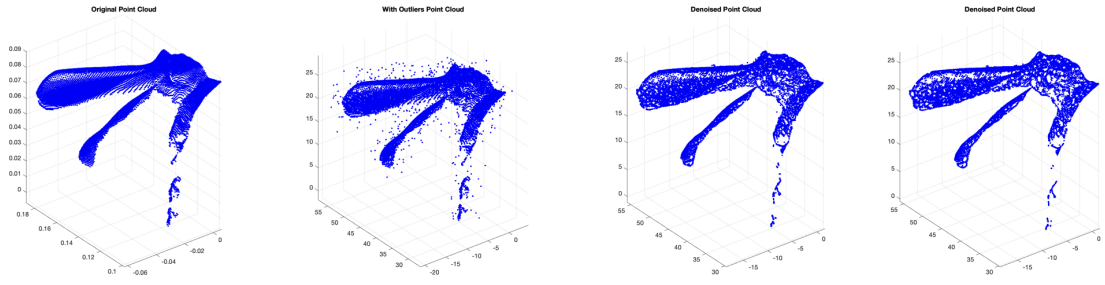


Figure 9: Point cloud denoising in more complex shape (ear of rabbit). (a) Original data(8.2k points) (b) Input noisy data (c) Result of algorithm in first iteration (d) Result of algorithm in second iteration. The setting for first iteration is $\alpha = 0.1$, $\beta = 0.001$, K = 5, the setting for second iteration is $\alpha = 0.01$, $\beta = 0.01$, K = 2

In the case of processing extensive point cloud datasets, computational challenges may arise due to the algorithm's reliance on Hessian matrix calculations and its inverse. These operations can lead to reduced efficiency in both processing speed and memory utilization. A solution to this problem involves spatially partitioning the point cloud into smaller, manageable batches, which are processed independently and subsequently merged. This approach is illustrated in Fig. 10, which demonstrates the denoising outcomes for a substantial dataset of 78,056 points (scan of Happy Buddha). While

attempting to apply the algorithm directly to the entire dataset resulted in technical difficulties, primarily memory overflow issues (28 GB memory requested), adopting a strategy of dividing the space into eight distinct sections for separate processing yielded impressive results. This approach not only help for rapid convergence but also proved highly effective in eliminating the majority of noise present in the point cloud, showcasing the algorithm's adaptability and efficacy when dealing with large-scale data.

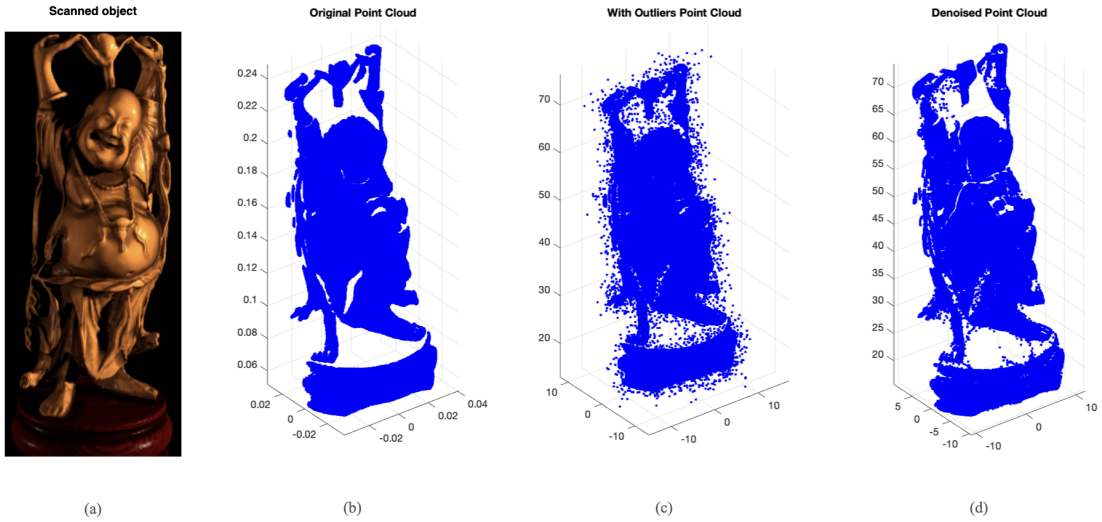


Figure 10: Point cloud denoising process for a 3D scanned object with 78k points. The space is uniformly divided into 2 x 2 x 2 = 8 subspace for noise reduction. (a) Original scanned object. (b) Original point cloud. (c) Point cloud with outliers. (d) Denoised point cloud. The setting for is $\alpha = 0.5$, $\beta = 0.01$, K = 5.

4.2 Parameters and scaling

The impact of scaling (of point cloud) effects on the algorithm's parameters overshadows its responsiveness to parameter adjustments. When the point cloud's distribution is confined to a narrow range, the smoothing component, denoted by $\sum_{l=1}^{k} L^l x$, can become the dominant factor in the optimisation process. This dominance may lead to image shrinkage and loss of shape (as illustrated in Fig 11), with extreme cases resulting in the trivial solution $x = \mathbf{0}, o = y$. While the algorithm exhibits significant sensitivity to parameter changes, the application of appropriate scaling methods can effectively reduce this sensitivity, thereby improving the algorithm's resilience to parameter fluctuations. Z-score normalization with a mean of 0 and standard deviation of 10 has demonstrated

consistent and reliable performance in this context.

The model's behavior is mainly controlled by a the following parameters: α, β, and . The α manages the degree of smoothness between data points and their closest neighbors. K defines the quantity of points from which information is extracted. β regulates the occurrence of outliers. As β increases, the number of outliers decreases. Table 1 offers a complete set of parameters and their recommended values for optimisation.

Parameter	**Value**	**Description**
α	1 - 0.1	Regularization parameter for smoothness
β	0.1 - 0.0001	Regularization parameter for outlier influence
g	10 - 1e6	Huber regulation, affects speed of convergence
tol	0.6 - 1	Termination value, algorithm terminates when $\|\nabla\phi\|_\infty < tol$
K	2 - 5	Number of nearest neighbors

Table 1: Recommenced Model Parameters

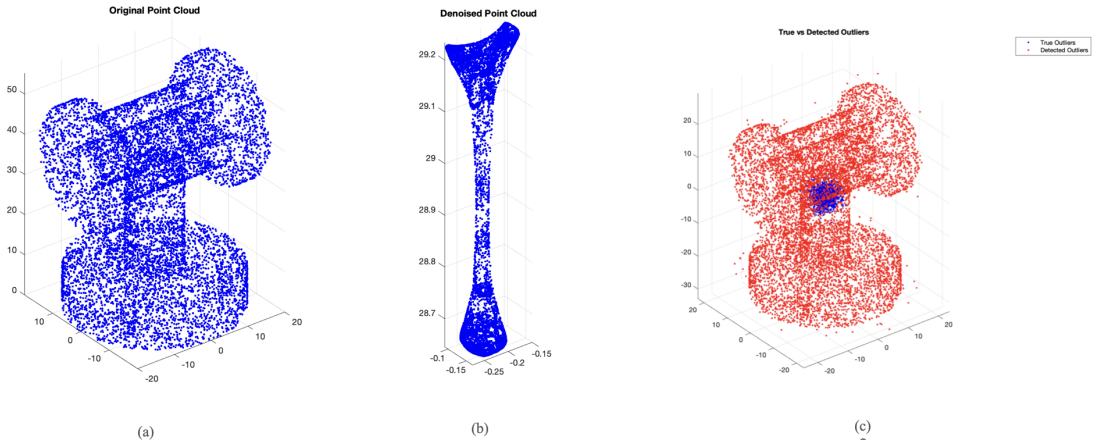


Figure 11: A case where α and K is set too high(1 and 10 respectively). The result shrink in x can be observed in the denoised point cloud (b). The original point cloud (a) is shown alongside the true and detected outliers (c), where the smoothing effect has resulted in a substantial alteration of the original structure.

4.3 Convergence and runtime

The algorithm converges fast in most scenario, often within 5 to 30 iterations. Table 2 presents several experimental models, their convergence rates, and processing times. The corresponding point cloud can be found on Appendix A.

It's worth noting that extremely fast or slow convergence can be a red flag. Convergence that is too rapid, such as within two iterations, or excessively slow, may suggest that the parameters are not optimally set. This can lead to unexpected or anomalous results in the output's shape or structure.

It's noteworthy that despite generally short execution times, the algorithm's runtime increases significantly as the data size grows. This is primarily due to the matrix inverse operation, which has a basic complexity of $O(n^3)$. However, spatial partitioning can effectively resolve this issue. For instance, the "pc 13" dataset contains 1.33 times more points than "pc 1", yet its processing time is approximately three times longer. More impressively, the "happy buddha" dataset, which is nearly eight times larger than "pc 13", only requires about eight times the runtime of "pc 13". This efficiency is achieved through spatial partitioning. Furthermore, the processing time for larger datasets like "happy buddha" could be reduced even further by implementing multi-threading techniques.

Model	Points	Num. Iterations	Time (seconds)
Ball 1	1000	35	1.163329
Ball 3	3000	31	6.762517
PC 1	7500	7	15.449940
PC 13	10000	9	43.569599
Happy Buddha	78056	110	273.829259

Table 2: Convergence and Time Statistics for Different Models

4.4 Limitation

When the α value is too high, or when algorithms are used iteratively for point cloud denoising, the returned results may become sparse, loosing some feature leading to unreliable reconstructed models. One solution can be use Point cloud upsampling algorithms. Point cloud upsampling is the process of increasing the density of a sparse point cloud by adding new points to the existing data. Edge-aware point set resampling (EAR) is one of the most commonly used methods. In EAR method, based on normal vectors, new points are inserted and projected on to the unknown underlying surfaced defined by the point set [29] [8]. Another limitation is that the algorithm often struggles when there are clustered outliers that are very close to the point cloud. The algorithm performs outlier removal on point clouds from a global perspective without considering local features. A possible solution is to incorporate normal vector information into the algorithm design. I will introduce this in detail in the next chapter.

5 Normal vector

Normal vector are vectors that is perpendicular to a surface at a given point, and they play a crucial role in encoding geometric information. While they primarily represent the orientation of the surface (i.e., how it is tilted in 3D space), they also implicitly encode higher-order geometric information, such as curvature and surface variations [30]. [31] observed that smooth surfaces have smoothly varying normals. Therefore, the information about the change of normal vector can be utilised for outlier detection or boundary sensing tasks, such as [32] used change of adjacent normal vector to reconstruct sharp features of surface. [29] used normal to reveal edge locations. However, normal is usually a term for a plane/or line, a point in three-dimensional space does not itself have a corresponding normal. In the next section I'll describe what the normal of a point cloud is and a method for calculating it.

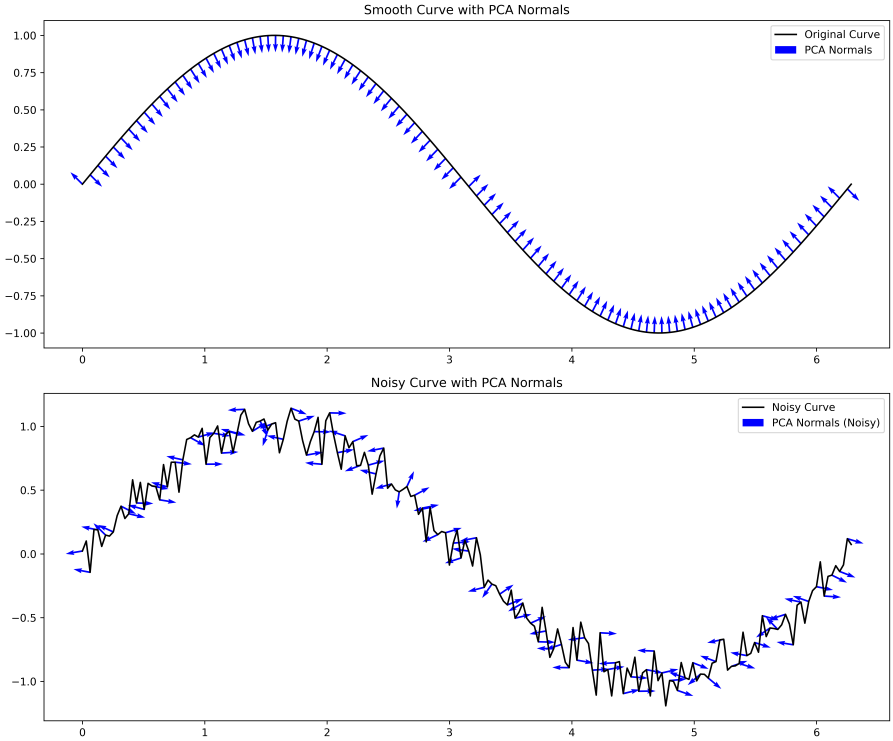


Figure 12: Comparison of normal vectors on smooth and noisy curves. Top: A smooth sinusoidal curve with gradually changing normals. Bottom: The same curve with added noise, showing irregular normal directions. This illustrates that smooth surfaces have smoothly varying normals, while irregular surfaces do not. Normals vectors in the plot are estimated using PCA method.

5.1 Normal vector estimation

In point cloud analysis, normal vector estimation is crucial but challenging due to the absence of a continuous surface. A common approach is to estimate the normal of a point by fitting a local plane. By estimating a plane that is tangent to the local plane, the normal of the point can be approximated. Least squares plane fitting is a mathematical method used to find the plane that best represents a set of points, i.e., the plane that minimizes the sum of the squares of the perpendicular distances,

$$\arg\min_{\hat{n}} \sum_{i=1}^{k} ((p_i - \bar{p}) \cdot \hat{n})^2$$

Where $\bar{p} \in \mathbb{R}^3$ is the current point, $p_i \in \mathbb{R}^3$ is neighbor points considered. The solution for estimating the normal vector of surface is therefore reduced to an analysis of the eigenvectors and eigenvalues (or principal component analysis) of a covariance matrix created from the nearest neighbors of the query point. For each point p_i, we assemble the covariance matrix $\mathbf{C}$ as follows:

$$\mathbf{C} = \frac{1}{k} \sum_{i=1}^{k} (p_i - \bar{p}) \cdot (p_i - \bar{p})^T$$

Where k is the number of point neighbors considered. $\bar{p}$ represents the 3D centroid of the nearest neighbors, which is the current point. The leading eigenvector $\mathbf{v}_1$, represents the direction of maximum data variation, i.e., the direction in which the point cloud is most dispersed. The middle eigenvector $\mathbf{v}_2$ is the second principal direction of variation. $\mathbf{v}_3$ which correspond to smallest eigenvalue λ_3, is perpendicular to the first two directions and represents the direction of minimum variation. This direction is closest to the normal direction of the local surface. Therefore, the eigenvector $\mathbf{v}_3$ of the covariance matrix $\mathbf{C}$ can be used to estimate normal direction. For more details, please see [33] chapter 4.3.

Apart from PCA estimation, there are many other techniques been studied. A review of methods can be found on [33]. If not otherwise indicated, normals in this study are estimated with PCA approach discussed above.

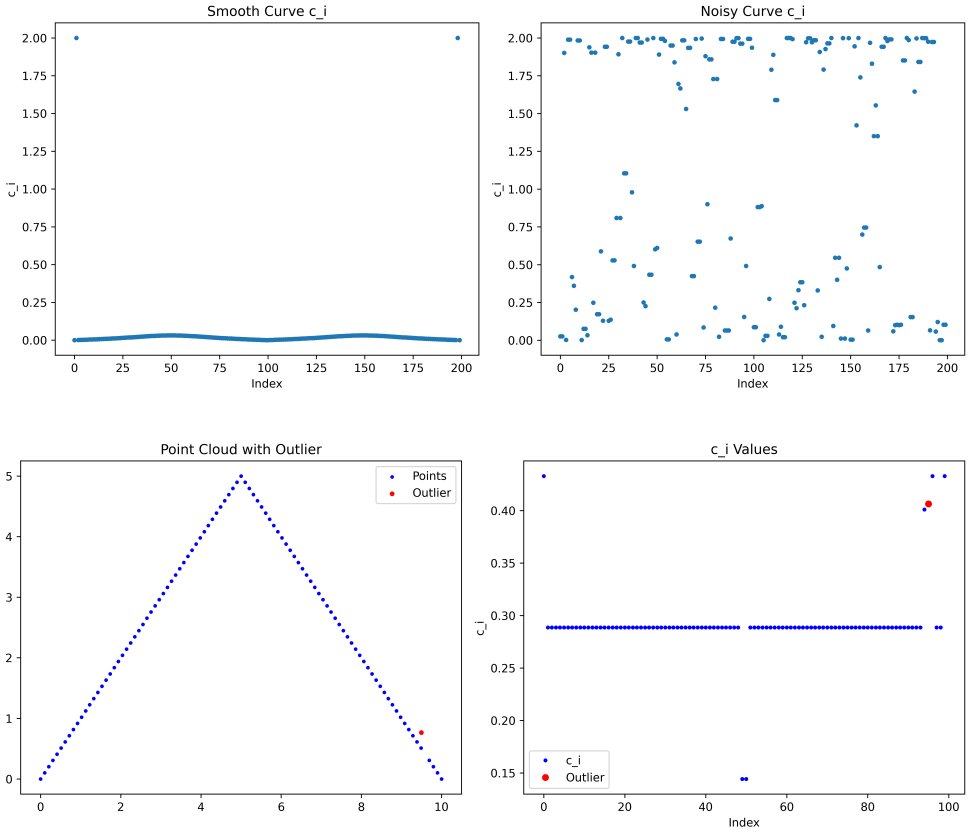


Figure 13: Plots illustrating assumption made in 5.2. The top tow figures shows c_i values in Fig 12. In the smooth curve, c_i values changing smoothly, apart from two initial points which is due by PCA method generate 2 possible opposite direction(pointing in and point out). When outliers are added, c_i values scatter from 0 to 2. The bottom two figures shows perturbed 2-dimensional point cloud with a V-shape and corresponding c_i values. Elevated c_i values occur primarily around the vertex of the V-shape and near the perturbed outlier.

5.2 Proposed solution

Here we take a different approach to define a point cloud. A point cloud P is defined to be set of points $p = \{p_1, p_2, ..., p_n\}$ where $p_i = (x_i, n_i)$. x_i is the coordinate of the i-th point and n_i is the corresponding normal vetcor, with $x_i, n_i \in \mathbb{R}^3$. We assume the underlying geometry of the point cloud is piecewise smooth. Based on the observation that smooth surfaces have smoothly varying normals [31], if two neighboring points p_i and p_j belong to the same smooth surface, then $|n_i - n_j|_2 \simeq 0$. Conversely, if they belong

to different smooth surfaces, i.e., they represent a sharp feature, then $|n_i - n_j|_2 \geq \eta$, where η is a threshold for sharp features [32]. Let $c_i = |n_i - n_j|_2$, where j is the nearest neighbor of i. We can identify outliers by solving the following optimisation problem:

$$\min_{c_x, c_o} \|c_x + c_o - c_y\|_2^2 + \beta \|c_o\|_1, \tag{22}$$

where c_x represents the inlier component of the normal difference, c_o represents the outlier component of the normal difference, c_y is the observed normal difference, β is a regularization parameter that controls the trade-off between the data fidelity term and the sparsity-inducing ℓ_1 norm on the outlier component. Again, to avoid trivial solution $c_x = c_y, c_o = \mathbf{0}$, we can add a term to regularize c_x.

Similar approach can be used. by limiting distance between actual normal difference, the solution will not degenerate.

$$\min_{c_x, c_o} \|c_x + c_o - c_y\|_2^2 + \alpha \|L c_x\|_2^2 + \beta \|c_o\|_1, \tag{23}$$

Where $L \in \mathbb{R}^{n \times n-1}$ and Lx returns $c_x^i - c_x^{i-1}$ limiting sudden variation of c_x. Outlier can be find by checking non-zero entries of c_o. We accept a point to be outlier if $c_o^i < \eta$. This now corresponds to the same optimisation problem described in (**P**), with d=1. The proposed solution enables us to leverage the algorithm's capabilities to not only eliminate outliers but also to determine the characteristics of these outlier entries.

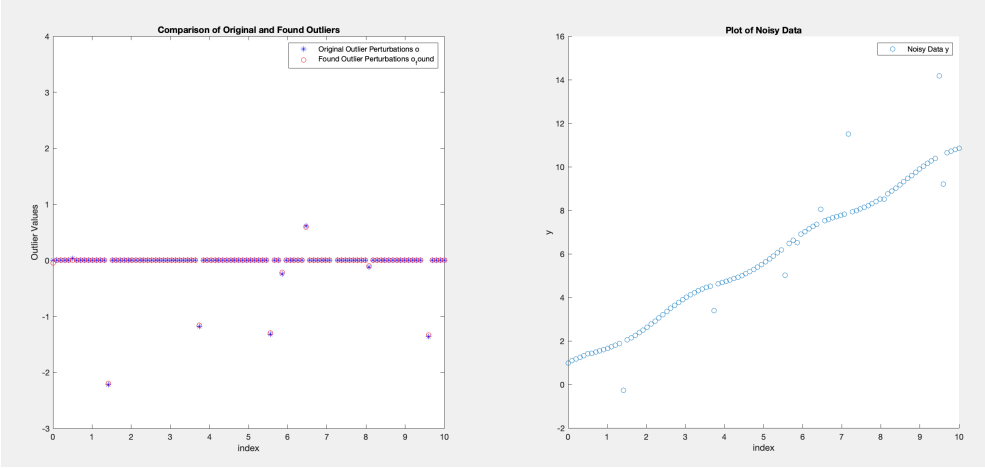


Figure 14: A implementation of algorithm 2 to 1D signal with outlier. From the left figure we can see the algorithm successfully found index and intensity of perturbation. Here alpha = 0.01, beta = 0.001, tol = 1e-3, K = 1.

6 Conclusion

Point clouds are essential in numerous fields, including 3D modeling, computer vision, and robotics. One of the key challenges in working with point clouds is the presence of outliers, which can significantly impact processes like surface reconstruction, object detection, and classification. In this study, we propose an efficient point cloud outlier removal method based on the ℓ_1 optimisation approach, which operates with little to no prior information about the point cloud. This method not only ensures effective outlier removal but also demonstrates rapid convergence, making it suitable for real-time applications. Importantly, the approach is highly flexible and has the potential to be extended to other types of data, making it a versatile tool in data analysis and processing across different domains.

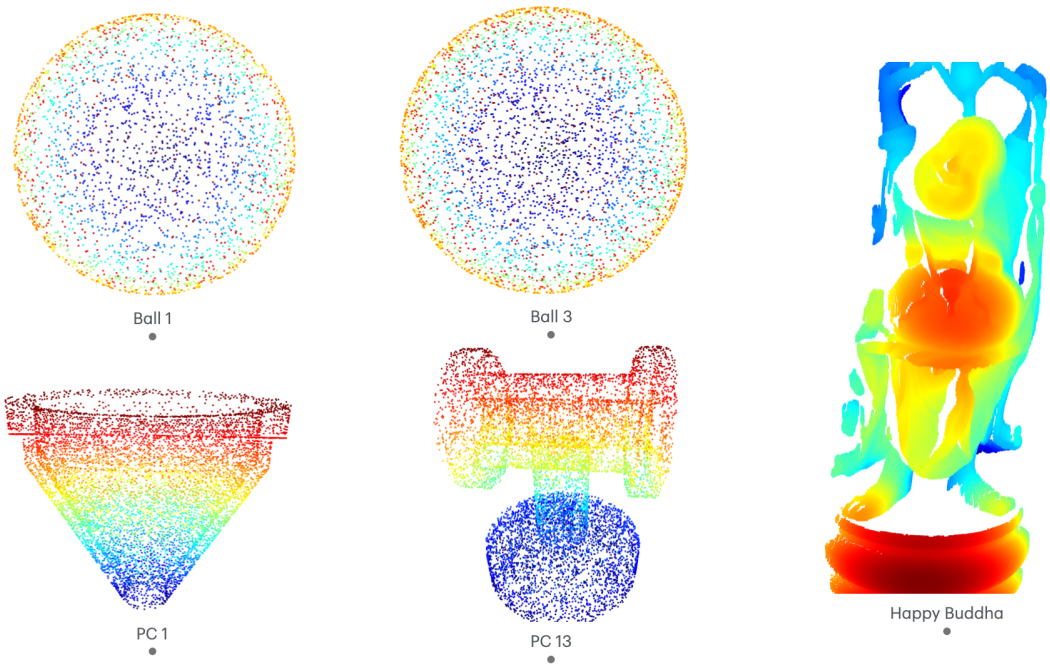
Future research can be conducted in several directions. First, after filtering the point cloud, the algorithm may lead to increased sparsity, creating gaps in the data, this is due to smoothing term, it applies a clustering force to the data. A key area of focus will be how to modify the algorithm to prevent such issues or integrate upsampling methods within the process. Another direction is applying the proposed method to other fields. The outlier removal in this study is focused on point clouds with dimensionality d=3, but we have already experimented with 1D and 2D signals, where the method has also proven to be highly effective. Finally, while the algorithm performs with fast runtime, it requires significant memory usage. optimising the algorithm to handle large-scale datasets efficiently is another promising area for further research.

Different representations of point clouds (normal vectors), have also been briefly studied. A potential outlier removal model was proposed based on the property that normal vectors exhibit smooth changes on smooth surfaces. However, due to time constraints, a comprehensive exploration in this direction was not completed. For the issues encountered and potential solutions, please refer to Appendix C.

In summary, proposed outlier removal method offers a robust and efficient solution for point cloud processing with minimal prior knowledge, showing strong potential for real-time applications and adaptability to other data types. While challenges such as increased sparsity and high memory usage remain, these present valuable opportunities for further research and optimisation. Future work will also explore the potential of

utilizing different point cloud representations, such as normal vectors, to enhance the method's effectiveness across various applications.

A Tested Point Clouds



Point cloud "Happy Buddha" is shared by Stanford 3D Scanning Repository

http://graphics.stanford.edu/data/3Dscanrep/#uses

Point cloud "PC 1" and "PC 13" is shared by Romanengo et al. [1], which can be downloaded from

https://github.com/chiararomanengo/Fit4CAD/tree/main

The code for generate sphere point cloud is listed below

```
function pointCloud = generateSpherePointCloud(num_points)
    rng(0);
    phi = 2 * pi * rand(num_points, 1);
    theta = acos(2 * rand(num_points, 1) - 1);
    r = 30;
    x = r * sin(theta) .* cos(phi);
    y = r * sin(theta) .* sin(phi);
    z = r * cos(theta);
    pointCloud = [x, y, z];
end
```

Listing 1: Code used for sphere point cloud generation

B Source Code

The source code of implementation and can be found and duplicate from

https://github.com/AndYueWu/Simultaneous-denoising-and-outlier-removal

C Problem facing and possible solutions of normal method

While implementing the proposed model, a key issue has arisen. Despite calculating the normal difference on a smooth 3D point cloud without introducing outliers, the values of c_i remain scattered. This outcome contradicts the assumption that $\|n_i - nj\|_2 \simeq 0$ if point i and j lie on the same smooth surface (See Fig. 14). One possible reason is because of the unreliable normal estimate. The reliability of normal prediction has

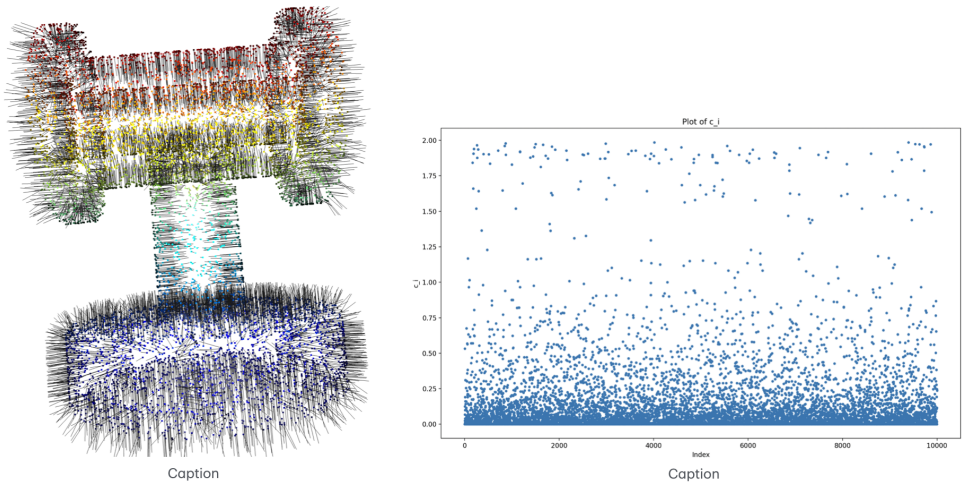


Figure 15: A clean point cloud with it's estimated normal(left) and it's corresponding c_i. The value of c_i is scattered around, contradicted to the assumption.

been a focus of attention in many cases, and scholars have proposed various methods to address it. For example, [29] used iterative bilateral smoothing to pre-process normals, which is a method introduced by [34]. Especially, for a point $p_i = (x_i, n_i)$, the difference

between its assigned normal n_i and the neighborhood normal can be calculated as

$$f(p_i, n_i) = \sum_{s_i' \in N_{s_i}} \|n_i - n_i'\|^2 \theta(\|p_i - p_i'\|) \psi(n_i, n_i'),$$

whereN_{s_i} is the set of points s_i' such that $\|p_i - p_i'\| < \sigma_p$, and σ_p is the given neighborhood size. The spatial and normal weight functions are defined as:

$$\theta(r) = e^{-r^2/\sigma_p^2},$$

$$\psi(n_i, n_i') = e^{-\left(\frac{1-n_i \cdot n_i'}{1-\cos(\sigma_n)}\right)^2}$$

To minimize the normal differences between all points on the surface and their neighbors, iteratively update n_i for all i with:

$$n_i \leftarrow \frac{\sum_{s_i' \in N_{s_i}} \theta(\|p_i - p_i'\|) \psi(n_i, n_i') n_i'}{\sum_{s_i' \in N_{s_i}} \theta(\|p_i - p_i'\|) \psi(n_i, n_i')}.$$

[32] takes a different approach to smooth normal by solving convex optimisation problem

$$\arg\min_{N} \sum_{(p_i, p_j) \in E} w_{ij} \left\|n_i - n_j\right\|_2^2 \quad \text{s.t.} \quad \forall i \left\|n_i - n_i^{\text{in}}\right\|_2 \leq \gamma_n.$$

Where γ_n is expected noise level.

References

[1] C. Romanengo, A. Raffo, Y. Qie, N. Anwer, and B. Falcidieno, "Fit4cad: A point cloud benchmark for fitting simple geometric primitives in cad objects," *Computers & Graphics*, vol. 102, pp. 133–143, 2022.

[2] S. A. Bello, S. Yu, C. Wang, J. M. Adam, and J. Li, "Deep learning on 3d point clouds," *Remote Sensing*, vol. 12, no. 11, p. 1729, 2020.

[3] F. Remondino, "From point cloud to surface: the modeling and visualization problem," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 34, 2003.

[4] M. Marsala, A. Mantzaflaris, and B. Mourrain, "G1 spline functions for point cloud fitting," *Applied Mathematics and Computation*, vol. 460, p. 128279, 2024.

[5] F. o. Blais, "Review of 20 years of range sensor development," *Journal of electronic imaging*, vol. 13, no. 1, pp. 231–243, 2004.

[6] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, vol. 1, pp. 519–528, IEEE, 2006.

[7] K. Wolff, C. Kim, H. Zimmer, C. Schroers, M. Botsch, O. Sorkine-Hornung, and A. Sorkine-Hornung, "Point cloud noise and outlier removal for image-based 3d reconstruction," in *2016 Fourth International Conference on 3D Vision (3DV)*, pp. 118–127, IEEE, 2016.

[8] Y. Sun, S. Schaefer, and W. Wang, "Denoising point sets via l0 minimization," *Computer Aided Geometric Design*, vol. 35, pp. 2–15, 2015.

[9] O. Schall, A. Belyaev, and H.-P. Seidel, "Adaptive feature-preserving non-local denoising of static and time-varying range data," *Computer-Aided Design*, vol. 40, no. 6, pp. 701–707, 2008.

[10] O. Schall, A. Belyaev, and H.-P. Seidel, "Robust filtering of noisy scattered point data," in *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.*, pp. 71–144, IEEE, 2005.

[11] Y. Duan, C. Yang, H. Chen, W. Yan, and H. Li, "Low-complexity point cloud denoising for lidar by pca-based dimension reduction," *Optics Communications*, vol. 482, p. 126567, 2021.

[12] E. A. L. Narváez and N. E. L. Narváez, "Point cloud denoising using robust principal component analysis," in *International Conference on Computer Graphics Theory and Applications*, vol. 2, pp. 51–58, SCITEPRESS, 2006.

[13] X.-F. Han, J. S. Jin, M.-J. Wang, W. Jiang, L. Gao, and L. Xiao, "A review of algorithms for filtering the 3d point cloud," *Signal Processing: Image Communication*, vol. 57, pp. 103–112, 2017.

[14] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 12, no. 7, pp. 629–639, 1990.

[15] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: nonlinear phenomena*, vol. 60, no. 1-4, pp. 259–268, 1992.

[16] C. Lange and K. Polthier, "Anisotropic smoothing of point sets," *Computer Aided Geometric Design*, vol. 22, no. 7, pp. 680–692, 2005.

[17] B. Mederos, L. Velho, and L. H. de Figueiredo, "Robust smoothing of noisy point clouds," in *Proc. SIAM Conference on Geometric Design and Computing*, vol. 2004, p. 2, SIAM Philadelphia, PA, USA, 2003.

[18] A. Tagliasacchi, H. Zhang, and D. Cohen-Or, "Curve skeleton extraction from incomplete point cloud," in *ACM SIGGRAPH 2009 papers*, pp. 1–9, 2009.

[19] T. Zhang, "Multi-stage convex relaxation for learning with sparse regularization," *Advances in neural information processing systems*, vol. 21, 2008.

[20] C. Dinesh, G. Cheung, and I. V. Bajić, "3d point cloud color denoising using convex graph-signal smoothness priors," in *2019 IEEE 21st International Workshop on Multimedia Signal Processing (MMSP)*, pp. 1–6, IEEE, 2019.

[21] E. Leal, G. Sanchez-Torres, and J. W. Branch, "Sparse regularization-based approach for point cloud denoising and sharp features enhancement," *Sensors*, vol. 20, no. 11, p. 3206, 2020.

[22] J. C. De Los Reyes, E. Loayza, and P. Merino, "Second-order orthant-based methods with enriched hessian information for sparse 1-optimization," *Computational Optimization and Applications*, vol. 67, no. 2, pp. 225–258, 2017.

[23] S. J. Wright, "Numerical optimization," 2006.

[24] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 1999.

[25] G. Andrew and J. Gao, "Scalable training of l 1-regularized log-linear models," in *Proceedings of the 24th international conference on Machine learning*, pp. 33–40, 2007.

[26] R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu, "Sample size selection in optimization methods for machine learning," *Mathematical programming*, vol. 134, no. 1, pp. 127–155, 2012.

[27] F. Zhang, *The Schur complement and its applications*, vol. 4. Springer Science & Business Media, 2006.

[28] N. Akkiraju, H. Edelsbrunner, M. Facello, P. Fu, E. Mucke, and C. Varela, "Alpha shapes: definition and software," in *Proceedings of the 1st international computational geometry software workshop*, vol. 63, 1995.

[29] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, and H. Zhang, "Edge-aware point set resampling," *ACM transactions on graphics (TOG)*, vol. 32, no. 1, pp. 1–12, 2013.

[30] I. Guskov, W. Sweldens, and P. Schröder, "Multiresolution signal processing for meshes," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 325–334, 1999.

[31] T. Tasdizen, R. Whitaker, P. Burchard, and S. Osher, "Geometric surface processing via normal maps," *ACM Transactions on Graphics (TOG)*, vol. 22, no. 4, pp. 1012–1033, 2003.

[32] H. Avron, A. Sharf, C. Greif, and D. Cohen-Or, "1-sparse reconstruction of sharp point set surfaces," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 5, pp. 1–12, 2010.

[33] K. Klasing, D. Althoff, D. Wollherr, and M. Buss, "Comparison of surface normal estimation methods for range sensing applications," in *2009 IEEE international conference on robotics and automation*, pp. 3206–3211, Ieee, 2009.

[34] A. C. Öztireli, G. Guennebaud, and M. Gross, "Feature preserving point set surfaces based on non-linear kernel regression," in *Computer graphics forum*, vol. 28, pp. 493–501, Wiley Online Library, 2009.