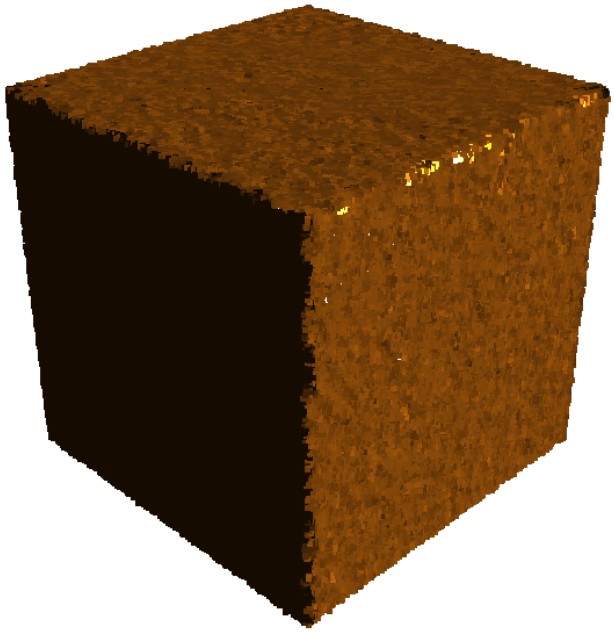
A point-cloud normal surface estimation methods comparison



Bachelor Thesis
26.03.2023

by Alessio Brazerol, 18-924-084

Supervisors:
Prof. Dr. Renato Pajarola
Luciano A. Romero Calla (assistant)

Visualization and MultiMedia Lab
Department of Informatics
University of Zürich

University of Zurich[UZH]

VISUALIZATIONANDMULTIMEDIALAB

Abstract

Point clouds are used in various algorithms in related fields, such as Visual Computing. Some of these algorithms require high-quality normals as input. Computing these surface normals is often the job of a different algorithm. Researchers have suggested vastly different algorithms over the years. Recent work focuses on using deep learning to estimate surface normals. It is important that these normal estimation algorithms perform well, as the other algorithms rely on their quality. This means the normal estimation algorithms need to be robust against various defects, including noise, outliers and differences in sampling density. Researches have proposed different measures to evaluate these normal estimation algorithms as well as how to produce synthetic test data to test against. Synthetic test data is especially useful as it provides ground truth normals to test against and can be constructed to contain a single or multiple defects with various strengths. Because of this, synthetic models are a valuable addition to real-world test data. In this paper, we evaluate four different normal estimation algorithms. This includes three regression based and one deep learning based method. We develop a tool to execute and compare the selected normal estimation algorithms. Our goal is to use good measures and cover multiple defects with our test data. In order to do this, we test the methods against various levels of noise and outliers. We use five different synthetic point cloud models and a real-world point cloud to test against. This allows us to identify the strengths and weaknesses of the tested methods.

Contents

1 Introduction

Estimating surface normals of a point cloud is a much-researched area. Many key algorithms depend on surface normals of high quality for input. These are, among others, surface reconstruction [HDD+92], geometric primitive extraction [PKG03], anisotropic smoothing [LP05][CCZ+18]. The main problem is that in the presence of noise, outliers, or a difference in sampling density, the estimation of the normals of a surface remains a highly complex problem. This is especially true for sharp regions, such as the edges and corners of a point cloud.

Furthermore, these algorithms often estimate point-cloud surface normals in the magnitude of millions or even larger. This means that they do not only need to produce high-quality normal estimations but also need to be efficient. Thus, performance versus quality is something algorithms sometimes have to consider.

Most commonly used methods by normal estimation algorithms can be loosely categorized into regression-based, Voronoi-based, and deep-learning methods [ZPLZ19; BM12]. According to Boulch and colleagues [BM12], regression-based methods try to estimate a tangent plane or shapes, like spheres or jets, to a set of neighboring points. They state that these methods are resilient to noise but sensitive to outliers. Recent research has focused on improving this shortfall, as seen in the paper by Mura et al [MWP18]. They point out that regression-based techniques can cause the smoothing of sharp features, resulting in inaccurate normals being estimated around edges and corners. As they highlight, the estimate's accuracy is determined by the size of the neighborhood; a larger neighborhood creates a more noise-resistant result, but it also has the potential to make sharp features appear even smoother [BM12].

Voronoi-based methods use a Voronoi diagram and the furthest vertex of the Voronoi cell to approximate the normal [YWG+19]. The methods preserve sharp features and are reliable with differences in sampling density [YWG+19; ZPLZ19]. Dey and Goswami [DG06] extended this method to make it more robust to noisy point clouds. Still, Voronoi-based methods are often incapable of handling heavy noise [YWG+19; ZPLZ19]. Lui et al. [LZC+15] proposed a method that produces reliable results that withstand noise and a difference in sampling density. Still, it is much more computationally demanding than other available methods [YWG+19].

In recent years, methods that use deep learning techniques to estimate surface normals of unstructured point clouds have emerged. However, implementing deep learning techniques remains difficult, especially when applied to problems involving unstructured point clouds [BG20]. As with other fields using deep learning techniques, these approaches need a big set of training samples with ground truth data to compare against [ZPLZ19]. If no ground truth normals are available, training these methods can be challenging [ZPLZ19].

2 Related Work

Klasing and colleagues compare in their paper [KAWB09] five different regression-based methods for range sensing applications. They focus their analysis on the quality and performance of the different methods while varying neighborhood size and noise strength. To estimate the quality, they use the mean normalized inner product of the estimated normal and its ground truth equivalent. As the central point of their paper is to check if, for a given algorithm, the increased computation time justifies the higher performance, they define a performance index that depends on the quality, computation time, and the optimal time they wish the algorithm to have [KAWB09]. In contrast, we compare the execution time of the different methods, as we do not require that we can use the algorithms in real-time scenarios. Klasing and colleagues prepare two test sets, one comprising synthetic box and cylinder data. The second is a scanned point cloud scanned by a laser scanner [KAWB09].

In contrast, Dey et al. [DLS05] focus on comparing Voronoi-based methods. They expand on the evaluation of just testing point clouds with a varying degree of noise by testing point clouds, which have some difference in sampling density. This means they were generated by sampling a surface unevenly. Additionally, they tested point clouds, which are generated by sampling a surface with high curvature in some areas. To compare the quality of the computed results, they use the mean angular error, and the standard deviation [DLS05].

We use a different evaluator to compare the quality and the mean angular error between the estimated normal and its ground truth equivalent. Using the angular error is easier to interpret than the inner product. Still, both are equally meaningful, as our measure is just the arccosine of a normalized inner product. However, in contrast to Klasing et al. [KAWB09] and Dey et al. [DLS05], we also use the median angular error as a measure, as it is much more robust to outliers. Both of them do not test their chosen methods against outliers. But as they are an important defect in point clouds, we decided to test their robustness against them.

A popular error measure is Root Mean Square (RMS), which was extended by Boulch and Renaud [BM12] to Root Mean Square with threshold RMS_τ as according to them, RMS favors smooth reconstruction everywhere, including at edges. This means that a smooth edge has a lower error than having some normals with values from the other side of the edge [BM12]. According to Boulch and Renaud, RMS_τ considers any angular error above the threshold τ as terrible by setting it to $\frac{\pi}{2}$. They take $\tau = 10^\circ$ in their examples [BM12]. Numerous papers, such as Zhao et al. [ZPLZ19] or Cao et al. [CCZ+18], use RMS and RMS_τ as it is widely regarded as a good measure. In contrast, we use the percentage of points with an angular error of less than 10°. This measure is very similar as it is equivalent to calculating the percentage of points inside the threshold of RMS_τ, which is the percentage of points not penalized because of unwanted smoothing.

Our work strongly relates with Mura et al.'s [MWP18] because we implemented their method using C++ and compared it with a range of other methods. We adopted their evaluation method and the shape of their synthetic models to compare our results with theirs. Besides the models proposed by Mura et al., we introduce a more complex synthetic model with higher surface detail and structural complexity. The synthetic models they used are simple shapes with no surface detail [MWP18]. Furthermore, we build upon their evaluation by testing the method against a state-of-the-art deep learning-based method.

3 Technical Solution

In our choice of algorithms to test robust normal estimation against, we looked at popular C++ libraries that work with point clouds. We used the Point Cloud Library (PCL) [RC11] and the Computational Geometry Algorithms Library (CGAL) [The23a].

According to Rusu and Coisins, PCL is "a fully templated, modern C++ library for 3D point cloud processin" [RC11, p. 1]. Their normal estimation algorithm is based on Principal Component Analysis (PCA) [11].

CGAL is described as, "a software project that provides easy access to efficient and reliable geometric algorithms in the form of a C++ library" [The23b, para. 1]. According to their documentation, the library provides three different algorithms for estimating surface normals [AGJ+23]. They describe the different methods as `jet_estimate_normals()`, `pca_estimate_normals()` and `vcm_estimate_normals()`.

The method `jet_estimate_normals()` estimates a normal by fitting a jet surface over its nearest neighbors [AGJ+23]. The method `pca_estimate_normals()` is based on the same theory as the algorithm used by PCL [AGJ+23]. `vcm_estimate_normals()` is a Voronoi-based method based on the article by Merigot and their colleagues [MOG11] [AGJ+23].

We decided to use `jet_estimate_normals()` to test against. This is becase we already had a PCA-based method, and we could not produce adequate results with the Voronoi-based method while using the same fixed number of the nearest neighbors as in the paper by Mura et al. [MWP18]. To ensure comparability with the results of Mura et al. [MWP18], we used the same fixed number of neighbors as in their paper.

For the deep learning method, we decided to use DeepFit by Ben-Shabat and Gould [BG20], as it is a very recent method and an expansion of a jet-estimation method.

3.1 Principal Component Analysis

We will present a similar explanation as presented in the paper by Shakarji and Srinivasan [SS13]. The normal to a query point $\boldsymbol{p}$ of a surface can be determined by the normal of the tangent plane to the surface at the query point p. Given a finite set of points $N_{\boldsymbol{p}}^k = \{x_i = (x_i, y_i, z_i) : i = 1, 2, ..., k\}$ form the neighborhood of the query point, the tangent plane can be estimated by the plane minimizing the total squared distance to the points from the neighborhood. Such a plane is called a least-square fitted plane to the set $N_{\boldsymbol{p}}^k$.

To determine the least-square fitting plane to a set of points $N_{\boldsymbol{x}}^k$, let $w_1, w_2, ..., w_N$ be the corresponding positive weights, $\boldsymbol{x} = (x, y, z)$ be an arbitrary point on the least square fitted plane, and $\boldsymbol{a} = (a, b, c)$ the normal of the plane. Then, Shakarji and Srinivasan [SS13] define the signed orthogonal distance function of a point $\boldsymbol{x}_i$ to the plane as:

$$d_i = \boldsymbol{a} \cdot (\boldsymbol{p}_i - \boldsymbol{x}).$$

Shakarji and Srinivasan [SS13] define the weighted centroid, which lies on the least square-fitted plane constrained to have the normal $\boldsymbol{a}$:

$$\bar{\boldsymbol{x}} = (\bar{x}, \bar{y}, \bar{z}) = \frac{\sum_{i=1}^{k} w_i \boldsymbol{x}_i}{\sum_{i=1}^{k} w_i}.$$

To show that the centroid lies on the least square-fitted plane, they use the definition of the plane $\boldsymbol{a} \cdot \boldsymbol{x} - d = 0$, where d is the signed distance from the plane to the origin. They define the distance of a point $\boldsymbol{x}$ to the plane as $d_{plane}(x) = \boldsymbol{a} \cdot \boldsymbol{x} - d$. With this, Shakarji and Srinivasan [SS13] define the weighted sum of squares objective function for a plane with a normal vector $\boldsymbol{a}$ as:

$$F(d) = \sum_{i=1}^{k} w_i(\boldsymbol{a} \cdot \boldsymbol{x}_i - d)^2$$

Shakarji and Srinivasan [SS13] find the first derivative of $F'(d)$ to be defined by $F'(d) = -2\sum_{i=1}^{N} w_i(\boldsymbol{a}\cdot\boldsymbol{x}_i - d)$ and the second derivative to be defined by by $F''(d) = 2\sum_{i=1}^{N} w_i > 0$.

The second derivative of $F(d)$ being strictly positive implies that if there exists a point at which the first derivative is equal to zero, we find a global minimum at this point [SS13]. They then solve for d by distributing the summation:

$$d = \frac{\sum_{i=1}^{k} w_i(\boldsymbol{a} \cdot \boldsymbol{x}_i)}{\sum_{i=1}^{k} w_i} = \boldsymbol{a} \cdot \left(\frac{\sum_{i=1}^{k} w_i \boldsymbol{x}_i}{\sum_{i=1}^{k} w_i} \right) = \boldsymbol{a} \cdot \bar{\boldsymbol{x}}.$$

This shows that at the least square solution, the weighted centroid must lie on the plane as $\boldsymbol{a} \cdot \bar{\boldsymbol{x}} - d = 0$ [SS13].

As the centroid of the point lies on the solution of the least square-fitted plane, they use the centroid as the guaranteed point on the plane and get the new objective function:

$$F_*(d) = \sum_{i=1}^{k} w_i(\boldsymbol{a} \cdot \boldsymbol{x}_i - d)^2.$$

This problem can be solved using Lagrange multipliers. We want to minimize $F_*(x)$ subject to the constraint $G(a) = 0 = |\boldsymbol{a}|^2 - 1$ [SS13]. They find that the gradients of F and G are defined as:

$$\nabla F_*(d) = \begin{bmatrix} \frac{\partial F}{\partial a} \\ \frac{\partial F}{\partial b} \\ \frac{\partial F}{\partial c} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} w_i[\boldsymbol{a} \cdot (\boldsymbol{x}_i - \bar{\boldsymbol{x}})](\boldsymbol{x}_i - \bar{\boldsymbol{x}}) \\ \sum_{i=1}^{N} w_i[\boldsymbol{a} \cdot (\boldsymbol{x}_i - \bar{\boldsymbol{x}})](\boldsymbol{y}_i - \bar{\boldsymbol{y}}) \\ \sum_{i=1}^{N} w_i[\boldsymbol{a} \cdot (\boldsymbol{x}_i - \bar{\boldsymbol{x}})](\boldsymbol{z}_i - \bar{\boldsymbol{z}}) \end{bmatrix}$$

$$\nabla G = 2\boldsymbol{a}.$$

According to Shakarji and Srinivasan [SS13], the minimum occurs at the point where $\nabla F_* = \lambda \nabla G, \lambda \in \mathbb{R}$ and we can write $\nabla F_* = 2\mathbf{L}\boldsymbol{a} = 2(\mathbf{M}^\top \mathbf{M})\boldsymbol{a}$, where $\mathbf{M}$ is defined as the $k \times 3$ matrix given by:

$$\mathbf{M} = \begin{bmatrix} \sqrt{w_1}(\boldsymbol{x}_1 - \bar{\boldsymbol{x}}) & \sqrt{w_1}(\boldsymbol{y}_1 - \bar{\boldsymbol{y}}) & \sqrt{w_1}(\boldsymbol{z}_1 - \bar{\boldsymbol{z}}) \\ \sqrt{w_2}(\boldsymbol{x}_2 - \bar{\boldsymbol{x}}) & \sqrt{w_2}(\boldsymbol{y}_2 - \bar{\boldsymbol{y}}) & \sqrt{w_2}(\boldsymbol{z}_2 - \bar{\boldsymbol{z}}) \\ \vdots & \vdots & \vdots \\ \sqrt{w_N}(\boldsymbol{x}_N - \bar{\boldsymbol{x}}) & \sqrt{w_N}(\boldsymbol{y}_N - \bar{\boldsymbol{y}}) & \sqrt{w_N}(\boldsymbol{z}_N - \bar{\boldsymbol{z}}) \end{bmatrix}$$

With this, they define the 3×3 eigenvector problem:

$$\mathbf{L} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \lambda \begin{bmatrix} a \\ b \\ c \end{bmatrix}.$$

Solving this 3×3 eigenvector problem gives us the direction of the fitted plane, which can be done this using well-known methods, such as Jacobi iterations [Sha98]. Shakarji and Srinivasan [SS13] further explains that as the eigenvectors of $\mathbf{M}^\top \mathbf{M}$ are also the singular vectors of $\mathbf{M}$, we could also apply singular value decomposition to gain better numerical results.

The question remains of how to select the correct eigenvector. For this, they write the normal equations as follows:

$$\sum_{i=1}^{k} w_i(\boldsymbol{x}_i - \bar{\boldsymbol{x}})[\boldsymbol{a} \cdot (\boldsymbol{x}_i - \bar{\boldsymbol{x}})] = \lambda a$$

$$\sum_{i=1}^{k} w_i(\boldsymbol{y}_i - \bar{\boldsymbol{y}})[\boldsymbol{a} \cdot (\boldsymbol{x}_i - \bar{\boldsymbol{x}})] = \lambda b$$

$$\sum_{i=1}^{k} w_i(\boldsymbol{z}_i - \bar{\boldsymbol{z}})[\boldsymbol{a} \cdot (\boldsymbol{x}_i - \bar{\boldsymbol{x}})] = \lambda c$$

Multiplying these equations by a, b and c, respectively, gives [SS13]:

$$\sum_{i=1}^{N} w_i[\boldsymbol{a} \cdot (\boldsymbol{x}_i - \bar{\boldsymbol{x}})]^2 = \lambda |\boldsymbol{a}|^2 = \lambda.$$

This means $F_*(x) = \lambda$ and, because we try to minimize F_*, the correct solution corresponds to the smallest eigenvalue value [SS13].

In their paper [HDD+92] by Hoppe et al., they demonstrate that the normal vector $\boldsymbol{a}$ to the least square fitted plane at the query point $\boldsymbol{p}$ together with the centroid $\bar{\boldsymbol{x}}$ can be determined using principal component analysis. For this, they find the symmetric 3×3 covariance matrix defined by:

$$C = \sum_{\boldsymbol{x}_i \in \mathcal{N}_p^k} (\boldsymbol{x}_i - \bar{\boldsymbol{x}}) \otimes (\boldsymbol{x}_i - \bar{\boldsymbol{x}})$$

For the vectors $\boldsymbol{a} = \{a_1, a_2, ..., a_n\}$ and $\boldsymbol{b} = \{b_1, b_2, ..., b_m\}$, the outer product $a \otimes b$ is the $n \times m$ Matrix such that $a_{ij} = a_i \cdot b_j$ [HDD+92]. They then compute the eigenvalues and corresponding eigenvectors for the covariance matrix $\mathbf{C}$. The correct eigenvector that determines $\boldsymbol{a}$ is the eigenvector corresponding to the smallest eigenvalue, much like when using Lagrange multipliers [HDD+92].

3.2 Robust Normal Estimation

We can categorize Robust Normal Estimation by Mura et al. [MWP18] as a regression-based method, as PCA is a big part of its normal estimation procedure.

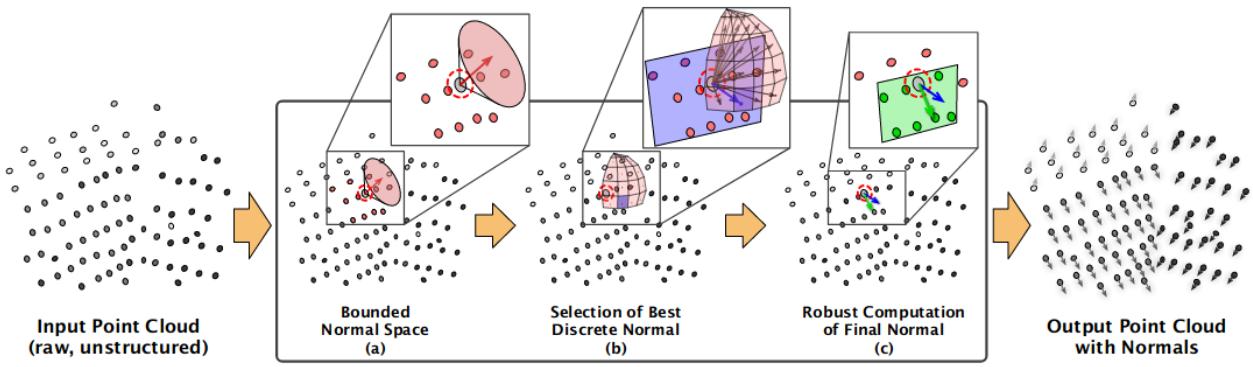


Figure 3.1: Visual overview of the robust normal estimation pipeline. First, an initial estimated normal is computed using PCA and its local neighborhood. A sub-region around the initial normal is defined where the correct normal lies with a certain percentage (a). Next, the sub-region is discretized, and for each patch, a candidate normal is defined that starts at the query point and ends in the center of the patch. The best candidate normal is selected based on the smallest median distance of the points in the neighborhood to the tangent plane defined by the candidate normal and the query point (b). Lastly, the final normal is computed by iteratively applying PCA to a set of inliers (c) [MWP18].

Given a query point $\boldsymbol{p}$, its neighborhood $\mathbf{N}_x^k$ and the complete normal space where the normal of point $\boldsymbol{p}$ could lie, Mura et al. [MWP18] want to define a sub-region of the normal space where the correct normal lies with a high probability. To do this, they divide the computed sub-region into discrete patches; each patch represents a

plane with a certain orientation defined by a normal vector. From this set of patches, they select the patch that, as a tangent plane of point $\boldsymbol{p}$, has the least median distance to the point in the neighborhood. Given this best tangent plane, Murea et al. [MWP18] robustly select the inlier points and use them to compute a final estimated normal using PCA iteratively. An overview of the pipeline can be seen in Figure 3.1.

3.2.1 Limit sample space

Mura et al. [MWP18] start with an initial normal $\tilde{\boldsymbol{n}}$ generated by the PCA of the k nearest neighbors $\mathcal{N}_{\boldsymbol{p}}^k$ for a query point $\boldsymbol{p}$. Mura et al. [MWP18] then make use of Mitra and Nguyen's [MN03] findings, in which they propose an upper bound for the maximum angular error α of the initial normal $\tilde{\boldsymbol{n}}$ and the correct normal $\boldsymbol{n}_*$ [MWP18]:

$$\alpha \leq c_1 \kappa r + c_2 \frac{\sigma_n}{\sqrt{\epsilon \rho} r^2} + c_3 \frac{\sigma_n^2}{r^2}. \tag{3.1}$$

The bound depends on the curvature κ, the search radius r, the noise scale σ_n, and the sampling density ρ. It further depends on the constants c_1, c_2, and c_3 and the error tolerance ϵ. In contrast to Mitra and Nguyen [MN03], which use the bound to compute a varying search radius, Mura et al. [MWP18] use the bound to limit the normal space where the correct normal could lie to a solid angle around the initial normal $\tilde{\boldsymbol{n}}$.

Mura et al. [MWP18] then further improve the estimates for the curvature and sampling density specified by Mitra and Nguyen [MNG04] in an extension of their work. They did this because, in the initial definition, the curvature does not depend on the local noise level, which makes it unstable in noisy, planer neighborhoods. Because the curvature uses the mean in its formula, it is not robust to outliers. In the initial definition, the sampling density only depends on the query point and its k-th neighbor. Noise or outliers can make this estimate unreliable if one point is affected.

Mura et al. [MWP18] define the improved estimator for curvature as the ratio between the smallest eigenvalue λ_1 and the sum of all three eigenvalues:

$$\kappa = max(\frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3} - \sigma_n, 0).$$

By subtracting the estimated noise scale σ_n, they compensate for noise encoded in the eigenvalues and make the estimator more robust to noise because of the inherent low-pass filtering [MWP18].

Mura et al. [MWP18] estimate the noise scale σ_n as the median distance of the points in the neighborhood $\mathcal{N}_{\boldsymbol{p}}^k$ to the least square fitted plane [MWP18]. The improved estimator for the sampling density is very similar, but they use d_{med}, the median distance from the query point to its k neighbors $\mathcal{N}_{\boldsymbol{p}}^k$ instead of the mean distance:

$$\rho = 2 \cdot \frac{k}{\pi d_{med}^2}.$$

Using this equality, they define a sub-region of the normal space around the initial normal with a solid angle α, in which the correct normal lies with a probability of $1 - \epsilon$. Mura et al. [MWP18] set the error tolerance ϵ to 0.005. This means the correct normal lies in the sub-region with a probability of 99.5%.

3.2.2 Selection of Best Discrete Normal

Given this sub-region of the normal space around the initial normal, Mura et al. [MWP18] want to further discretize this sub-region. They get the area on the surface of the discrete unit sphere, spanned by α around the point where the first normal is facing. They then use a well-established accumulator design introduced by Borrmann et al. [BELN11]. The accumulator is designed to first divide the discrete unit sphere into N_s horizontal slices. Each slice is bounded by two latitudes and covers an angle of $\frac{\pi}{N_s+1}$ [MWP18]. To ensure that each patch has roughly the same area, the number of subdivisions of each slice must be varied depending on the latitude. This is because the longitude length depends on its latitude, and the longitude at the equator is the largest [BELN11]. Mura et al. [MWP18] then set, like Borrmann et al. [BELN11], the number of subdivisions at the equator to $2 \cdot N_s$

[MWP18]. In contrast to Borrmann et al. [BELN11], Mura et al. [MWP18] define two additional patches of an angle $\frac{1}{2} \cdot \frac{\pi}{N_s+1}$ at each of the two poles.

Given this discretization of the discrete unit sphere, we then need to extract the set patches that correspond to the solid angle α around the initial normal $\tilde{\boldsymbol{n}}$ [MWP18]. The method used by Mura et al. [MWP18] to extract these patches was not further specified, which is why we developed a method ourselves. To do this, we first compare the polar angle of each patch corner to the polar angle of the initial normal $\tilde{\boldsymbol{n}}$. This gives us at least one patch for every possible initial normal. We then further check if a corner of any other patch lies inside the sub-region by comparing the angle between the corners of a patch and the initial normal $\tilde{\boldsymbol{n}}$. We consider the patch if the angle is smaller than α for any corner. This provides us with most patches that overlap with the surface area except for a small subset of extreme cases. This is if the initial normal does not point to a point in the patch, and the initial normal lies between the patch corners when looking at just one of the polar axis. Furthermore, the patch corner lies outside the surface segment. Then the corner points in the other polar axis lie between the polar angle of the initial normal in this axis $\pm\alpha$ we would not consider the patch, even though the patch would intersect the surface segment. We obtained slightly worse results if we did not check for partial overlap.

Mura et al. [MWP18] then get from each selected patch a candidate normal $\tilde{\boldsymbol{n}}_j$, which is the vector that starts at the center of the unit sphere and ends at the center of each patch. For each candidate normal, They compute the median distance of the neighbors to the plane described by the candidate normal and the query point. The best (discrete) candidate normal is then selected by the minimal median distance [MWP18]:

$$\hat{\boldsymbol{n}} = \arg\min_{\tilde{\boldsymbol{n}}_j} \text{median}(\text{distances}(\text{plane}(\tilde{\boldsymbol{n}}_j, \boldsymbol{p}), \mathcal{N}_{\boldsymbol{p}}^k)).$$

3.2.3 Robust computation of the final normal

The next step in the method described by Mura et al. [MWP18] is to apply an iterative procedure to refine the best (discrete) candidate normal in the restricted sub-region of the complete normal space. The algorithm to obtain the final normal n_* is described in Algorithm 1.

Algorithm 1 Iterative refinement of best discrete candidate normal

Require: $\hat{\boldsymbol{n}}$: best (discrete) normal, $\boldsymbol{p}$: query point, $\epsilon_n \geq 0$
  $\boldsymbol{n} \leftarrow \hat{\boldsymbol{n}}$
  $i \leftarrow 0$
  while $i \neq 3$ **do**
    $\mathcal{D} \leftarrow distances(plane(\boldsymbol{n}, \boldsymbol{p}), \mathcal{N}_{\boldsymbol{p}}^k)$
    $d_{med} \leftarrow median(\mathcal{D})$
    $\mathcal{I} \leftarrow \{\boldsymbol{x}_j \in \mathcal{N}_{\boldsymbol{p}}^k | d_j \leq d_{med}\} \subset \mathcal{N}_{\boldsymbol{p}}^k$
    $\boldsymbol{n}_{prev} \leftarrow n$
    $\boldsymbol{n} \leftarrow$ PCA of $\mathcal{I}$
    if $\boldsymbol{n}_{prev} \cdot \boldsymbol{n} > 1 - \epsilon_n$ **then**
      break
    end if
    $i \leftarrow i + 1$
  end while
  $\boldsymbol{n}_* \leftarrow \boldsymbol{n}$

Mura et al. set the small constant ϵ_n to 1^{-4} and they end the algorithm once the refined normal has not deviated much from the normal prior to the current refinement step (i.e. if the dot product is almost 1)[MWP18]. They found that a maximum of 3 iterations suffices to achieve good results. With this simple procedure, they get a robust final normal $\boldsymbol{n}_*$ for a query point $\boldsymbol{x}$ [MWP18].

3.3 CGAL: jet-estimation

The general idea is to fit a polynomial to a surface described by the neighborhood, which makes it a regression-based method. We base our explanation of the method on the paper by Cazals and Pouget [CP05] and refer the interested reader to their paper for a more detailed explanation and further findings. According to Cazals and Pouget [CP05], we can locally write any regular embedded smooth surface as the graph of a bivariate height function that calculates the z coordinate based on x and y. They expand this height function as a Taylor expansion $J_{vecB,n}$ of order n and with coefficients $\boldsymbol{B}$:

$$f(x,y) = J_{\boldsymbol{S},n}(x,y) + O(||(x,y)||^{n+1}).$$

$J_{vecS,n}$ is called a jet of degree n, also noted as n-jet [CP05]. They state that since an n-jet has $n+1$ monomials of degree i, the n-jet has $N_n = 1+2+...+(n+1) = (n+1)(n+2)/2$ terms. Cazals and Pouget [CP05] state that from the 1-jet, we can compute the tangent space from the 2-jet, the curvature information, and so on. Their method is to fit the n-jet to a set of k points in a neighborhood $\mathbf{N}_p^k$ of a query point $\boldsymbol{p}$. The point $\boldsymbol{p}$ is not required to be in the set of neighbors, and we can assume that it lies at the origin of the coordinate system [CP05]. Cazals and Pouget [CP05] suggested that the issue of fitting an n-jet to a neighborhood of k nearby points can be solved using interpolation or approximation.

Interpolation finds a polynomial that exactly fits the set of neighbors [CP05]. They then try to find the n-jet $J_{A,n}$ with coefficients $\boldsymbol{A}$, such that for any point $(x_i, y_i, z_i) \in N_{\boldsymbol{p}}^k, i \in \{1, ..., k\}$:

$$f(x_i,y_i) = J_{\boldsymbol{S},n}(x_i,y_i) + O(||(x_i,y_i)||^{n+1}) = J_{\mathcal{A},n}(x_i,y_i). \tag{3.2}$$

Approximation, conversely, tries to approximate the height function without the need to reach the exact solution [CP05]. Cazals and Pouget [CP05] use least-squares approximation, which minimizes the sum of square errors between the values of the height function and the sought jet. They propose the following objective function to minimize:

$$\sum_{i=1}^{N} (J_{A,n}(x_i,y_i) - f(x_i,y_i))^2. \tag{3.3}$$

These problems can be written in the same matrix form [CP05]. Cazals and Pouget [CP05] first write the jet in the polynomial basis comprising the monomials $x^i y^i$. They then further specify $\boldsymbol{A}$ to be the N_n-vector of coefficients of the solution jet

$$\boldsymbol{A} = (A_{0,0}, A_{1,0}, A_{0,1}, ..., A_{0,n})^\top.$$

and $\boldsymbol{B}$ is the k-vector of the ordinates $z_i = f(x_i, y_i)$,

$$\boldsymbol{B} = (z_1, z_2, ..., z_k)^\top = (J_{B,n}(x_i,y_i) + O(||(x_i,y_i||^{n+1})_{i=1,...,N}.$$

Cazals and Pouget [CP05] then compute the $k \times N_n$ Vander-monde matrix:

$$\mathbf{M} = (1, x_i, y_i, x_i^2, ..., x_i y_i^{n-1}, y_i^n)_{i=1,...,k}$$

According to Cazals and Pouget [CP05], the number of points matches the number of parameters in the case of interpolation. They then rewrite the Equation 3.2 using $\mathbf{M}$:

$$\mathbf{M}\boldsymbol{A} = \boldsymbol{B}.$$

For approximation, $\mathbf{M}$ is a rectangular $k \times N_n$ matrix and they rewrite Equation 3.3 as [CP05]:

$$min||\mathbf{M}\boldsymbol{A} - \boldsymbol{B}||_2$$

Interpolation is a great choice if the data points match the number of terms, with the benefit that all used data points are guaranteed to lie on the calculated jet [CP05]. However, the approximation is preferable in the presence of heavy noise [CP05]. Furthermore, if the neighborhood is scarcely populated (i.e. $N_n > k$), the approximation is also preferred, as we might not be able to find an exact solution [BG20].

3.4 Deepfit

Noise and outliers heavily reduce the fitting accuracy of jet-fitting [BG20]. To overcome this, Ben-Shabat and Gould [BG20] expand the problem to a weighted least square problem. They define the optimization problem where $W \in \mathbb{R}^{k \times k}$ is a diagonal weight matrix $\mathbf{W} = \text{diag}(w_1, w_2, ..., w_k)$ as follows:

$$\beta = \underset{A \in \mathbb{R}^{N_n}}{\arg\min} \|\mathbf{W}^{\frac{1}{2}}(\mathbf{M}\boldsymbol{A} - \mathbf{B}\|^2.$$

Ben-Shabat and Gould [BG20] propose to train a neural network to estimate the weight of each neighbor of a query point. They work with a query point $\boldsymbol{p}$ from the input point cloud and then calculate the k-nearest neighbors $\mathcal{N}_{\boldsymbol{p}}^k$. They then feed $\mathcal{N}_{\boldsymbol{p}}^k$ into a PointNet, a novel deep learning network that directly consumes point clouds [RHML17]. This gives them a global point cloud representation $G(\mathcal{N}_{\boldsymbol{p}}^k)$. They draw out local representations from an intermediate layer for all the points $\boldsymbol{p}_j \in \mathcal{N}_{\boldsymbol{p}}^k$ to get $g(\boldsymbol{p}_j)$. Ben-Shabat and Gould [BG20] state, that "these representations are then concatenated and fed into a multi-layer perceptron $h(\cdot)$ followed by a sigmoid activation function". The sigmoid activation function limits the output values between 0 and 1 [BG20]. With this, they get a weight per point and can construct the diagonal point-weight matrix $\mathbf{W}$ defined as:

$$w_j = sigmoid(h(G(\mathcal{N}_{\boldsymbol{p}}^k), g(\boldsymbol{p}_j))) + \epsilon.$$

ϵ is a small constant that they add for numerical stability. They then compute the estimated normal using weighted n-jet estimation.

They then define a custom local consistency loss function for backpropagation while training the neural network. Their loss function comprises two terms. They call them a weighted normal difference term and a regularization term. Ben-Shabat and Gould [BG20] state, that "The weighted normal difference term computes a weighted average of the sine of the angle between the ground truth normal and the estimated normal at every local neighborhood point". To compute the normal at every local neighborhood point, they convert the n-jet to the implicit surface form $F(x, y, z) = 0$. They then compute the normal at each neighboring point $\boldsymbol{p}_j$ with:

$$\boldsymbol{n}_j = \left.\frac{\nabla F}{||\nabla F||}\right|_{\boldsymbol{p}_j} = \left.\frac{(-\beta_i \frac{\partial \mathbf{M}^\top}{\partial x}, \beta_i \frac{\partial \mathbf{M}^\top}{\partial y}, 1)}{||\nabla F||}\right|_{p_j}.$$

Ben-Shabat and Gould [BG20] then add a regularization term because if the neighboring point $\boldsymbol{p}_j$ do not lie on the surface it is possible that the point weights reach all zero. The regularization term computes the average log of all weights [BG20]. With this, they define the consistency loss for a query point $\boldsymbol{x}_i$ where $\boldsymbol{n}_{GT}$ is the ground truth normal:

$$L_{con} = \frac{1}{k}\left[-\sum_{j=1}^{k} log(w_j) + \sum_{j=1}^{k} w_j |\boldsymbol{n}_{GT} \times \boldsymbol{n}_j|\right]. \tag{3.4}$$

The complete loss function is defined by the sin loss between the estimated unoriented normal and the ground truth normal at the query point, the consistency loss Equation 3.4, and PointNet's transformation matrix regularization terms $L_{reg} = |\mathbf{I} - \mathbf{A}\mathbf{A}^\top|$ [BG20]. They use weighing factors α_1 and α_2 that were chosen empirically [BG20]. They define the complete loss function, thus:

$$L_{tot} = |N_{GT} \times N_i| + \alpha_1 L_{con} + \alpha_2 L_{reg}.$$

4 Implementation

We implemented a visualization/estimation application in C++ and GLFW [23]. The goal was to use this application to execute the different methods, visualize the point cloud with the estimated normals and evaluate the results against ground truth if available. We based the application on a template provided by the Lecture on Computer Graphics at the University of Zurich by Renato Pajarola and parts from the web book "Learn OpenGL" by Joey de Vries [dVri14].

We use the open-source library Dear ImGui [14] to visualize UI elements. We use an open-source header file browser implementation for Dear ImGui [img19] and an implementation for progress indicators [zfe18].

The application creates an instance of `NormalViewer` is created on startup, which is a subclass of `GLApp`. This class handles the general setup of GLFW, glad, and Dear ImGui, and creates the required instances, like `Camera` and `PointCloudManger` and contains the render loop. The goal was to render a point cloud and maybe other objects to visualize results (i.e. arrows or lines to show normals). We created three components to handle the rendering of an object: a `Mesh` component, a `Material` component and a `RenderObject`. The `Mesh` the component handles the creation and the filling of the VertexArrayObject, VertexBufferObject, and ElementBufferObject. The `Material` the component handles the shader program and its associated variables. And last but not least, the `RenderObject` renders the set Mesh using the set Material.

Another main component is the aforementioned `PointCloudManager`. It handles the point cloud by loading it with the `PointCloudLoader` executes a certain normal estimation method on the loaded point cloud or handles which results are visible. This means swapping between the results of different normal estimation methods and updating the estimated normals of the loaded point cloud.

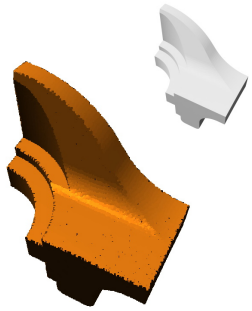
The `PointCloudManager` creates a `Mesh` object to store the point cloud. The point cloud mesh object contains a vertex (i.e. points) vector, color vector, normal vector, and ground truth normal vector. The class can then load a point cloud using an instance of type `PointCloudLoader`. This class loads a point cloud from a path the user can input via a file browser. The `PointCloudLoader` stores the data in provided vectors, which in this case, are the vectors stored in the point cloud mesh object. The loaded data must at least include positional data of a point and may include color and ground truth normal data.

We implemented all methods in static classes that contain a method called `calculateNormals`. The method requires as input the set of points, the viewpoint (i.e. scanner position) to reorient the normals, and a `Results` object to store the result and interim results. The methods handle the settings, such as the number of nearest neighbors themselves.

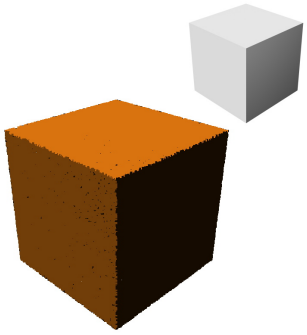
The `PointCloudManager` has a vector of `Result` objects of size equal to the methods available. A `Results` object contains a vector of settings used to run the normal estimation method, the stored completion times, results such as the estimated normal, and evaluation data such as the mean or median angular error. The `Results` object computes the evaluation data when the method `evaluateResults` is called. The `PointCloudManager` provides the user with the possibility of selecting the method and then swapping the active Results object.

5 Experimental Results

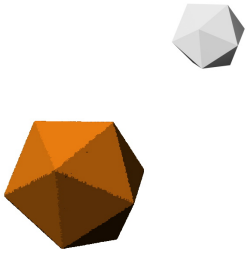
We created models similar to those of Mura et al. [MWP18] to ensure that our implementation of the robust normal estimation method was comparable. This means we choose to test the model against a set of synthetic and one real-world model. We represent the synthetic models together with their mesh representation and the real-world model in Figure 5.1.



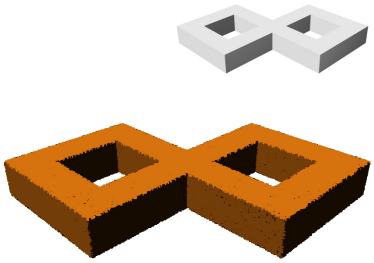
(a) Fandisk (100'047 points)



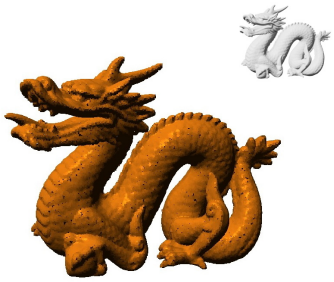
(b) Box (99'999 points)



(c) Icosahedron (100'000 points)



(d) Double Torus (100'001 points)



(e) Dragon (99'754 points)

(f) Office Room (2'780'160 points)

Figure 5.1: Overview of the test set without added defects. The point clouds (a)-(e) were randomly sampled form the 3D mesh to their right.

We created the *Box*, *Icosahedron*, and *double torus* meshes using Blender [Ble18]. The mesh for creating the *fandisk* [com14] and *dragon* [Sta14] model originated from the internet. We then used an open-source program called CloudCompare [GPL] to create a point cloud by randomly sampling the surface of the mesh. This resulted in a point cloud without noise or outliers and ground truth normals based on the normals of the original mesh. We then corrupted the point clouds with noise by adding a vector in the direction of the ground truth normal. The length of the vector is sampled from a normal (Gaussian) distribution with a mean of 0.0 and a standard deviation $\sigma = p * d_{bb}$. p is a percentage (i.e. 0.1%), and d_{bb} is the magnitude of the diagonal of the bounding box of the mesh used to generate the point cloud. We used the oriented bounding box of the mesh to estimate the bounding box and then extracted the diagonal length. We then added outliers to a certain percentage of point cloud points

(i.e. 2.5% and 5%). This was done by adding an offset in an arbitrary direction with length sampled from a uniform distribution between $[5 \cdot \sigma, \frac{d_{bb}}{4}]$.

A Faro Focus 3D laser range scanner was utilized by the Visualization and MultiMedia Lab (VMML) at the University of Zurich to scan the real-world model "2C05_2.ptx" which we refer to as *Office Room* [Mur16]. We could use the provided model directly for PCL and robust normal estimation. However, CGAL and DeepFit threw an exception when using this model. While experimenting with the data, we found that removing the points with zero values in all dimensions allowed the two algorithms to terminate. We thus performed a cleanup for these methods where we removed 62455 points.

We ran tests for PCL, CGAL, and robust normal estimation on a *Lenovo X1 Carbon 9th Gen* equipped with a *11th Gen Intel Core i7-1165G7 processor clocked at a max turbo frequency of 4.7 GHz* using a total of *4 cores* and *32 GB DDR4 RAM* using *Manjaro Linux*. These methods made use of GPU parallelization. We executed the tests for DeepFit in a Anaconda [Ana23] environment on a different machine, as the DeepFit algorithm requires a Cuda-compatible GPU. The second machine was a *Windows 10* machine with a *8th Generation Intel Core i7 processor* clocked at a max turbo frequency of *4.70 GHz* using a total of *6 Cores*. The machine used a *Nividia GeForce RTX 3090* and *64 GB DDR4 RAM*.

We set the number of neighbors to 64 for all test methods. For the robust normal estimation, we set $N_s = 16$ and c_1, c_2 and c_3 in Equation 3.1 to 1. For testing DeepFit we used the implementation of Ben-Shabat and Gould [IG23]. , we first trained a model with the provided training- and validation set, which can be downloaded by executing the python script `get_data.py`. We used the section for the training of the provided python script `run_DeepFit_single_experiment.py` for this but changed the parameter N_POINTS representing the number of the nearest neighbors to 64. We then used this model to estimate the normal for each point by using `compute_normals.py` from their tutorial.

We qualitatively analyzed all models by examining the rendered point clouds using Phong-based shading. As for the synthetic models, ground truth normal data is available to compare against; we did a qualitative evaluation. We calculate the mean and median angular error and the percentage of points with an angular error less than $10°$. As the methods only compute unoriented normals for a point, we compute the angular error twice using the originaly estimated normal and the flipped normal. We then select the smaller of the two angular error as the angular error of a estimated normal. We examined how well the models could tolerate noise and outliers and analyzed their execution time.

5.1 Qualitative Analysis

When examining the results without noise displayed in Figure 5.1, we can evaluate how good a method is based on various factors. There should be a prompt color change at the sharp features of a model, such as corners or edges. The model's flat planer regions should have a consistent color over the whole region. In curved regions, the color should change continuously from one color to the other. And last but not least, the algorithm should preserve surface details.

When looking at the cube model without noise, in Figure 5.1, we can see that PCL and CGAL both don't preserve sharp features by smoothing normals. Robust normal estimation preserves the sharp features well in contrast but doesn't result in a completely sharp edge. DeepFit results in a slight smoothing but not as strong as PCL or CGAL. However, DeepFit does not result in completely sharp edges either. In all three methods, the flat planar region is very uniform. This means all methods handle flat regions without noise very well. In round regions, PCL, CGAL, and DeepFit perform well as they produce smooth transitions. Robust normal estimation, however, produces band-like structures. We can observe this in the *Fandisk* in Figure 5.1. These are also present when using ground truth normals but probably exist, as the original mesh can only estimate the curvature with a finite amount of vertices. Thus, we can extrapolate that they are undesirable, and we count them negatively. No algorithm preserves surface details very well. We can see this when looking at how well the surface details of the *dragon* body in Figure 5.1 have been preserved. PCL and CGAL over smooth the normals way too much, which results in the *dragon* model losing nearly all surface details. Only the strongest features, like the structure on the belly, remain barely visible. Both robust normal estimation and DeepFit preserve more surface details, like

the belly structure in the front of the *dragon* model. However, the scale structure on the *dragon*'s body is not as noticeable as in the ground truth data.

We then looked at how the methods compare if we add 0.1% noise to the models. The results are in Figure 5.2. Adding this amount of noise does not seem to affect the precision regarding the sharp features of the methods. PCL and CGAL again have problems with preserving sharp features. Robust normal estimation again results in the best sharp features, and DeepFit again is somewhere in between them. If we look at flat regions, we can see some differences from the previous tests. PCL and CGAL both result in more or less flat regions that are just slightly noisy. This is visible in the flat regions of the *icosahedron* or the *box* model. Robust normal estimation results in a quite noisy flat region, and DeepFit is again somewhere between the three. Regarding curved regions, all methods produce smooth transitions, but the noisy flat region might overshadow the previously noted problem regarding the banding of robust normal estimation. Adding noise doesn't seem to affect the preservation of surface details positively or negatively.

When examining a scene instead of a single model, it is necessary to consider more than just the aforementioned factors to establish if a method performs well. In addition, we have to consider the boundary of different objects (e.g. the table leg and the floor). We rendered the model *Office Room* using Phong-based shading and present it in Figure 5.3. When looking at these results, we gain similar insights as when using synthetic models. If we compare the door frame and the boundary between the wall and floor, we can see that PCL and CGAL over smooth the edges, and robust normal estimation yields sharp edges. DeepFit has some smoothing and compared with previous results, it produces the noisiest edges of all the three methods. In flat regions, all three methods perform similarly well and produce properly oriented normals over the whole flat region. PCL, CGAL, and DeepFit again have similarly good results in curved regions. Robust normal estimation comes very close but has some noise. It is difficult to examine if the methods preserve surface details, as the point cloud density is too small for the elements farther in the back, and the front elements do not have any strong surface details. There are occurrences of normals of one object being affected by another object in all methods. PCL, CGAL, and DeepFit display this behavior the most, and it is visible at the boundary between the items on the bookshelf and the shelf itself, or where the legs of the chair meet the floor. For robust normal estimation, this behavior is just barely visible. Something unique to DeepFit is the occurrence of outliers across the scene.

5.2 Quantitative Analysis

As ground truth data was available to compare against, we were able to exactly measure the angular error between the estimated normal and the ground truth. We visualized the results without noise in Figure 5.4 and with 0.1% noise in Fig. 5.5. A green dot represents a point with an angular error less than $5°$, blue an angular error greater or equal to $5°$ but less or equal to $10°$, and red represents an angular error more than $10°$.

As mentioned in the qualitative analysis, PCL and CGAL don't preserve sharp features very well. We can see this in Figures 5.4 and Figure 5.5 in the *Fandisk*, *Box*, *Icosahedron*, and double torus model. Both methods produce large (red) bands of high error areas around the sharp features of the model. Robust normal estimation preserves them really well, which is visible by the small number of points with high angular error. DeepFit produces also a lot of points with high angular error at the sharp features of the model but far fewer than PCL or CGAL. We can also see that PCL and CGAL produce good results for the flat regions of the model with and without noise. Robust normal estimation and DeepFit only produce good results in the tests without noise; especially, robust normal estimation has a high amount of points (blue) with angular error $\geq 5°$ in the flat regions of the model. DeepFit also has some points with angular error $\geq 5°$ but not nearly as many. Similarly to before, the data also shows that PCL, and CGAL have good results in the curved areas of the model. However, we can again see that DeepFit and even more robust normal estimation have some points (blue) with an angular error of more than $5°$ in the curved areas of the model, notably in the *Fandisk* model in Figure 5.4. For PCL and CGAL, adding noise only results in a small degradation of the quality in curved regions of the model, which we can see in the *Fandisk* model in Figure 5.5, that only results in a small number of additional points with angular error $\geq 5°$. We see an overall increase in angular errors with robust normal estimation and DeepFit when adding noise, but curved regions do not display an additional decrease in quality to the rest of the model. When looking at the

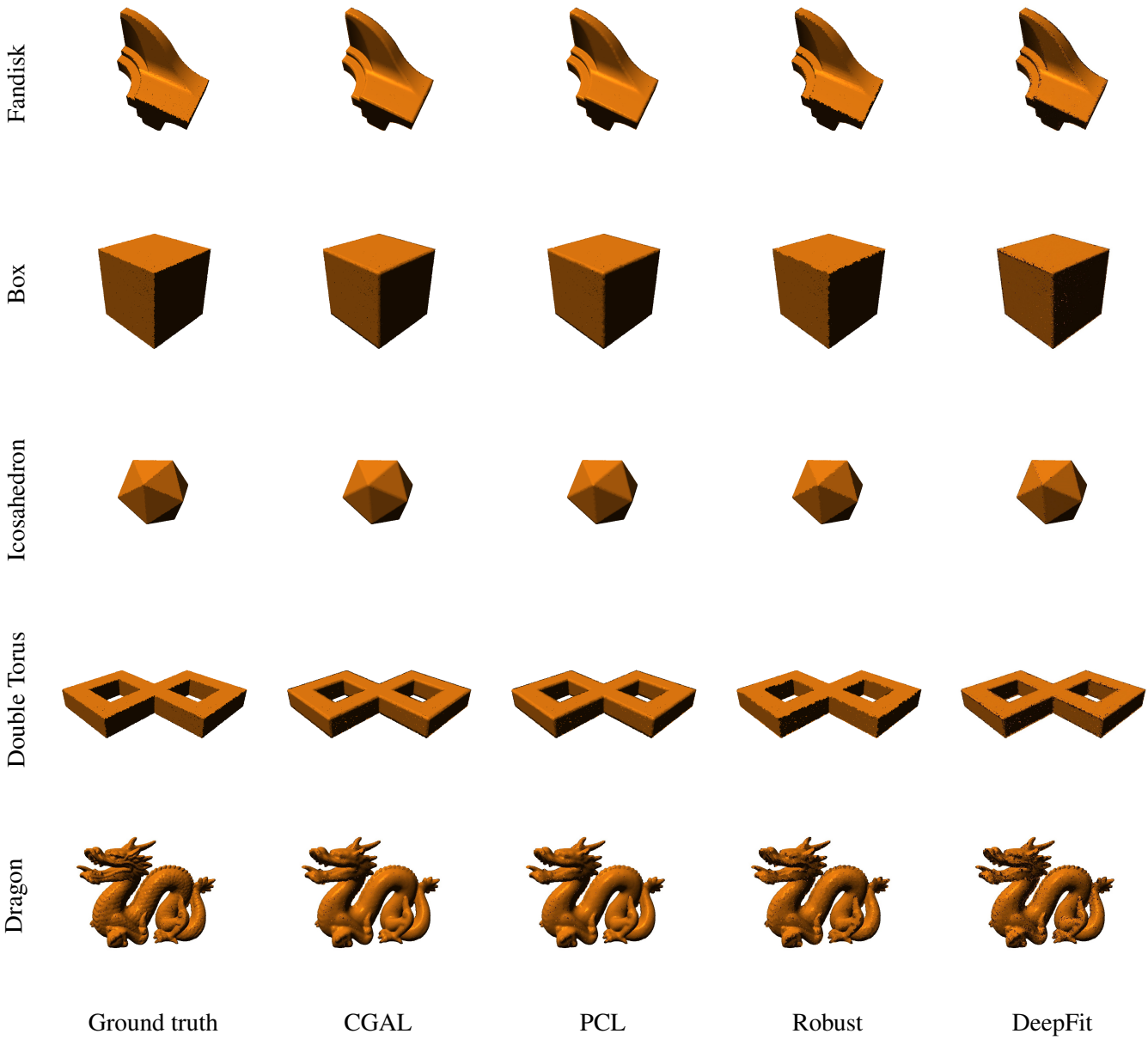


Table 5.1: Qualitative analysis of the normal estimation on synthetic data without noise

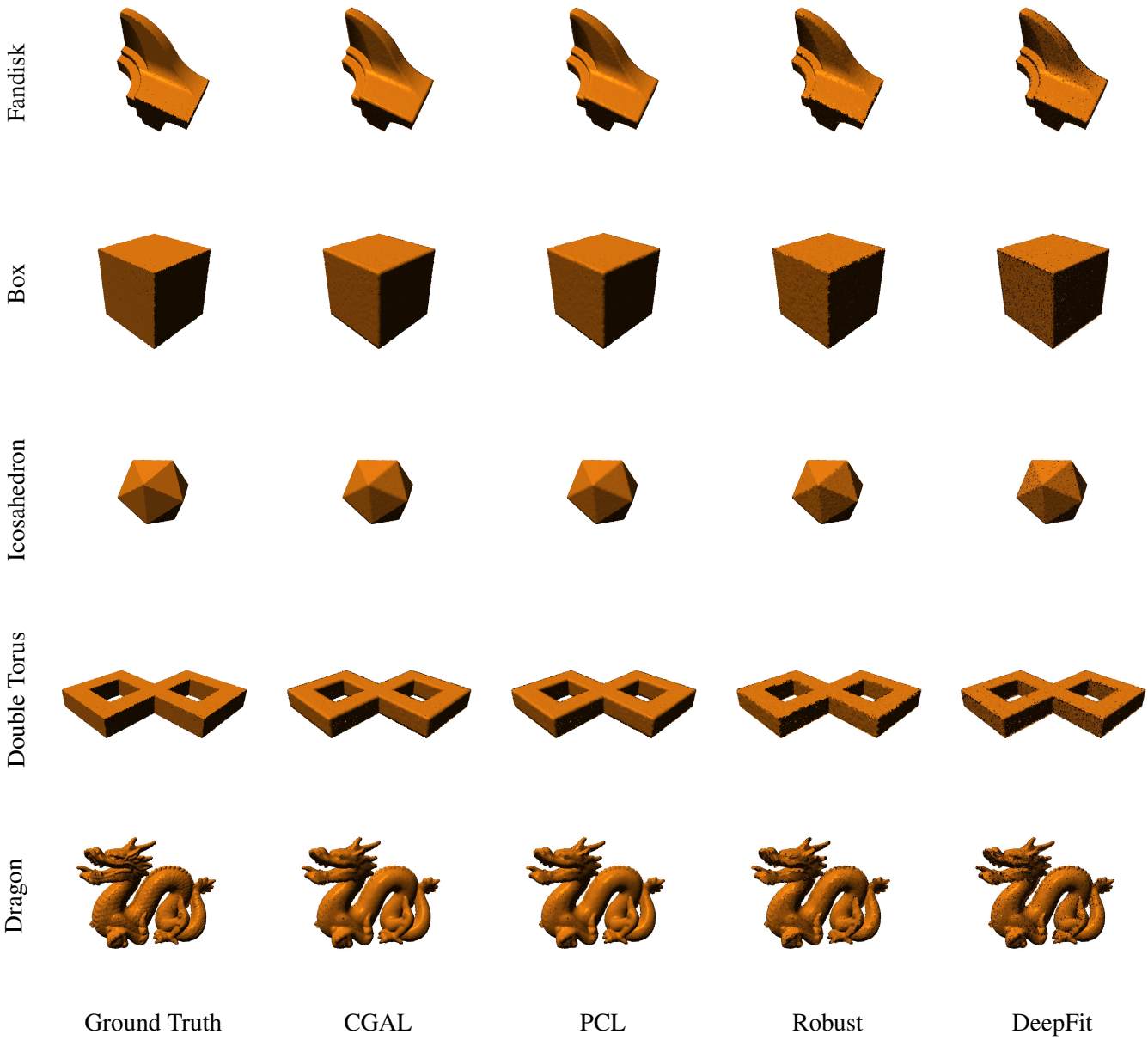
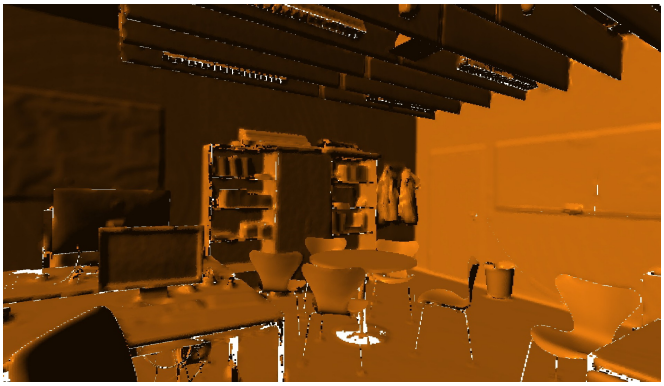
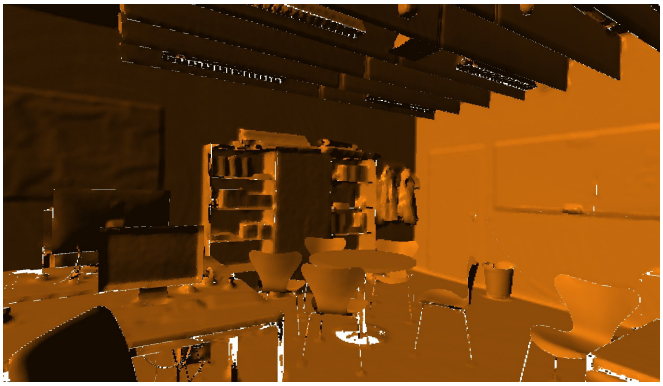


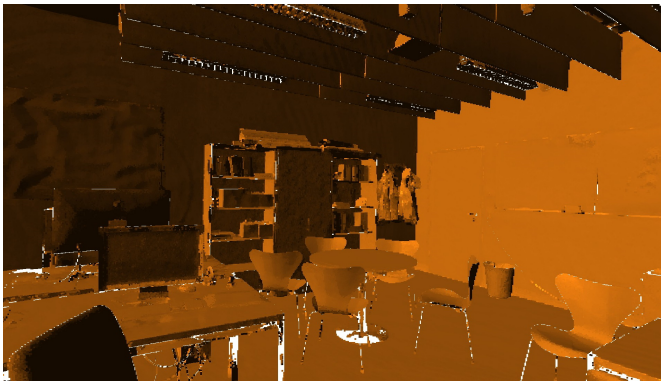
Table 5.2: Qualitative analysis of the normal estimation on synthetic data with 0.1% noise



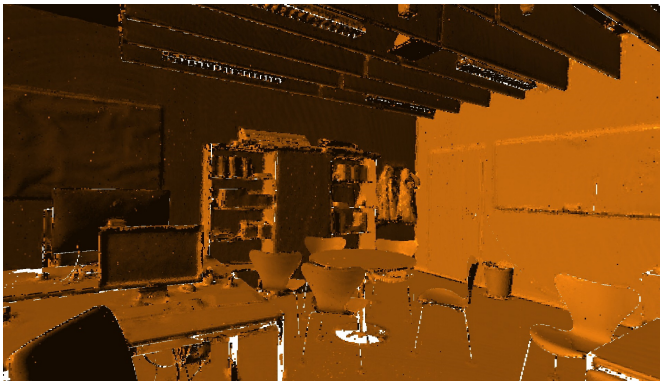
PCL



CGAL



Robust



DeepFit

Table 5.3: Render of the real-world model *Office Room* with phong base shading for all four methods.

dragon model, all methods produce a high amount of angular errors, especially between $\geq 5^\circ$ and $\leq 10^\circ$ (blue), this is because the *dragon* model has high surface details that all four methods cannot reproduce. Adding noise seems to affect mainly DeepFit and, in particular, robust normal estimation, which results in a sharp increase in the number and severity of points with angular errors.

We also computed the mean, median angular error, and the percentage of points with an angular error $< 10^\circ$ for all models with 0.1% noise. We represent this data in Table 5.6. A low mean and median value and a high percentage of points with angular error $< 10^\circ$ are considered good. PCL and CGAL have similar results in all methods over all three metrics. Both methods have a very low mean and median values, which results from the good results in the flat and curved regions of the model. The percentage of points with angular error $< 10^\circ$ is lower than robust normal estimation. This is because of the areas of high angular error around the sharp features of the model. The robust normal estimation has higher mean and median values because of the noisy flat regions, but a much lower percentage of points with angular error $< 10^\circ$ because the method does not produce many points with high angular error except a few at the sharp features of the method. DeepFit produces results between robust normal estimation and PCL or CGAL.

We can see the execution time of the different methods in Figure 5.7. We turned off parallel computing while we were running these tests since DeepFit did not have it. PCL is by far the most efficient of the four methods. CGAL and robust normal estimation have similar execution times, but it looks like CGAL scales better with an increase in the number of points, however, further investigation is necessary. DeepFit is by far the slowest, but it is likely that this is in part because the method was implemented in Python, whereas the other methods could benefit from the higher efficiency of C++.

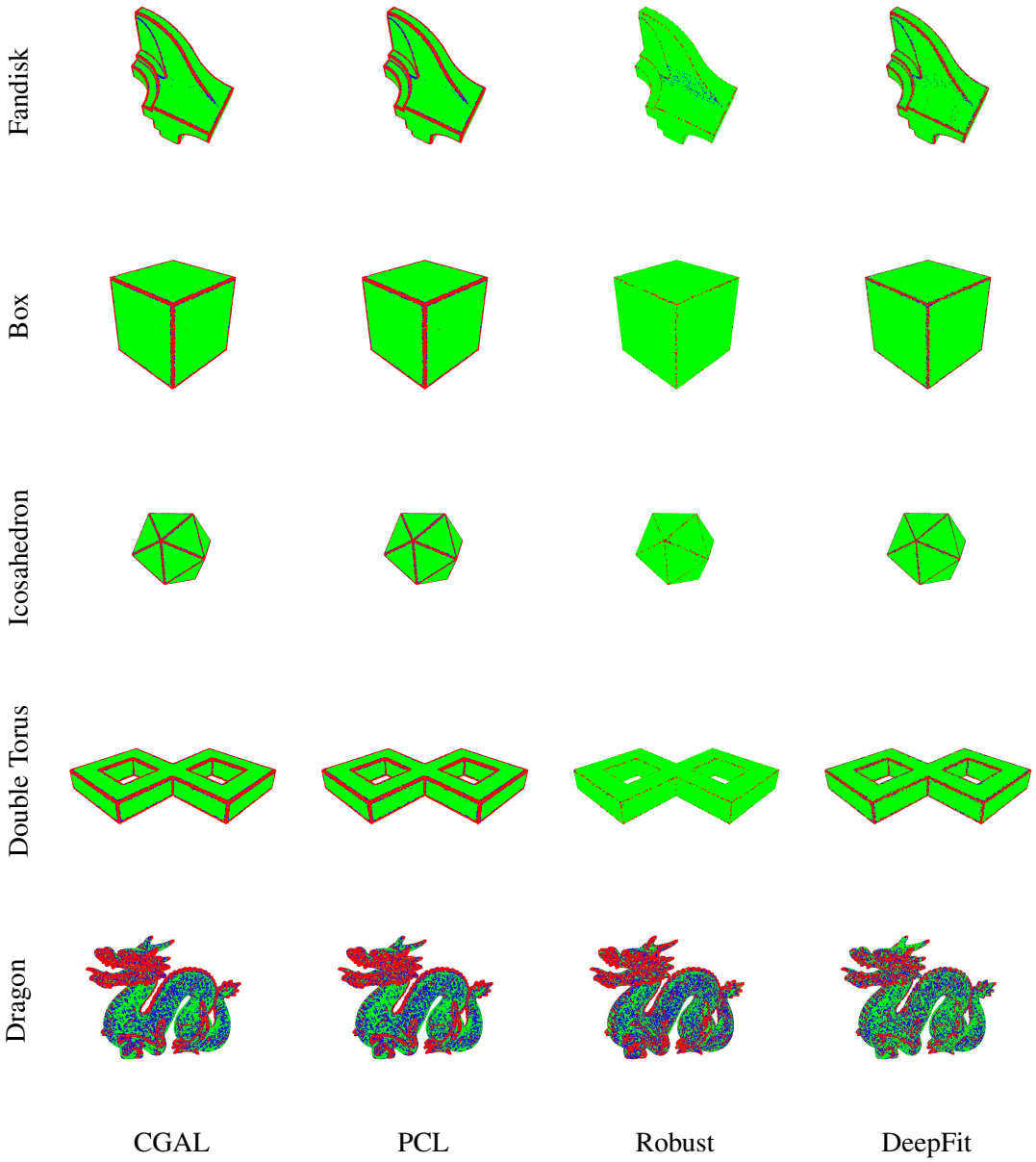


Table 5.4: Quantitative Analysis of the angular error between estimated normals and the ground truth normals on synthetic data without noise. The color green depicts angular errors $< 5°$, blue angular errors between $\geq 5°$ and $\leq 10°$ and red angular errors $> 10°$

5.3 Robustness

To evaluate the robustness to noise and outliers, we varied these parameters for the *box* model by performing tests with 2.5% or 5.0% outliers combined with 0.2% or 0.4% noise respectively. We visualized the results with deviation to ground truth in Figure 5.8 for PCL and CGAL and in Figure 5.9 for robust normal estimation and DeepFit. We calculate the mean and median angular error and the percentage of points with an angular error $< 10°$ in Figure 5.10.

We can determine that doubling the number of outliers has nearly no effect on the quality of the results. This is true for the visual results (i.e. going from the first line to the second in a Figure 5.8 and Figure 5.9) but also for the mean, median and percentage. For example, let's look at robust normal estimation and double the number of outliers from 2.5% to 5.0% in the model with 0.4% noise. The mean only increases by 0.15, the median by

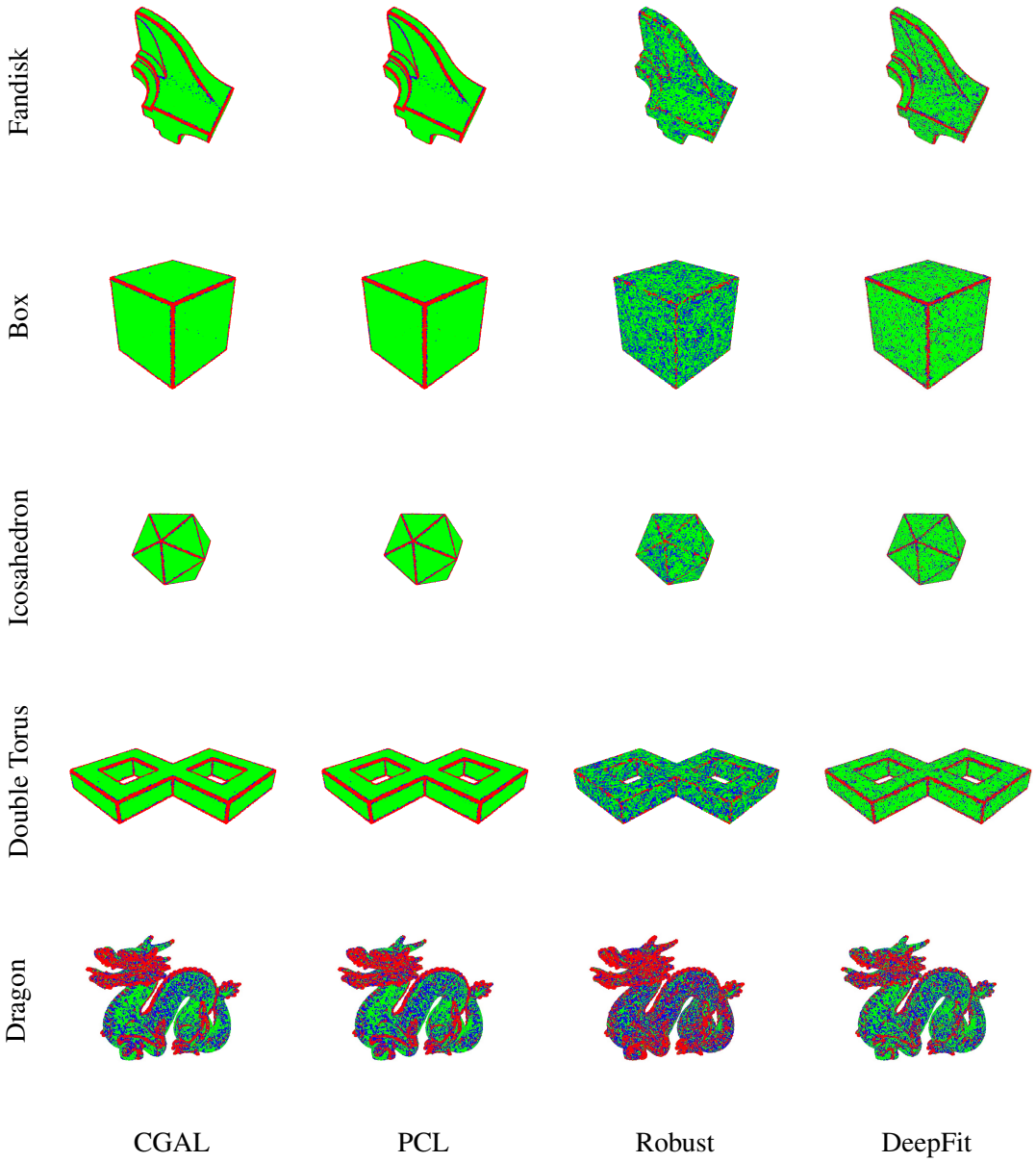


Table 5.5: Quantitative Analysis of the angular error between estimated normals and the ground truth normals on synthetic data with 0.1% noise. The color green depicts angular errors $< 5°$, blue angular errors between $\geq 5°$ and $\leq 10°$ and red angular errors $> 10°$

1.21 and the percentage decreases by 0.19%. We can observe similar small changes for the other methods with 0.2% noise. If we double the noise from 0.2% to 0.4% with 2.5% outliers, we can observe a sharp decline in the quality of all methods. The mean of CGAL and PCL increases by a factor of ~ 2.15, the median by a factor of ~ 1.25 and the percentage decreases by a factor of ~ 0.81. If we look at the robust normal estimation, the mean increases by a factor of ~ 2.19, the mean by a factor of ~ 1.92, and the percentage decreases by a factor of ~ 0.35. DeepFit finds itself somewhere in the middle, the mean increases by a factor of ~ 2.09, the median by a factor of ~ 1.69, and the percentage decrease by a factor of ~ 0.56. We can conclude that an increase in noise affects robust normal estimation the most. Noise affects the quality of the results computed with DeepFit, but not as much as the robust normal estimation. PCL and CGAL seem quite robust to noise compared to the other two methods. All methods seem similarly robust to an increase in the number of outliers.

Method		Fandisk	Box	Icosahedron	Double Torus	Dragon
CGAL	Mean	**1.79**	1.53	1.55	1.99	6.44
	Median	**5.93**	3.93	3.24	7.02	9.47
	$< 10°$	84.46%	91.49%	90.11%	81.12%	68.84%
PCL	Mean	**1.79**	**1.50**	**1.53**	**1.97**	6.58
	Median	5.78	**3.83**	**3.21**	**6.89**	9.60
	$< 10°$	83.97%	91.17%	90.17%	80.40%	68.30%
Robust	Mean	3.91	4.17	3.77	4.65	7.89
	Median	6.25	5.64	5.03	7.74	11.21
	$< 10°$	**92.80%**	**95.71%**	**93.42%**	**90.22%**	62.11%
DeepFit	Mean	3.32	2.75	2.72	3.05	**6.16**
	Median	7.02	5.16	4.60	7.57	**9.32**
	$< 10°$	85.93%	91.31%	89.78%	84.36%	**72.39%**

Table 5.6: Quantitative Analysis of the mean and median angular error in degrees as well as the percentage of points with an angular error $< 10°$ on synthetic data with 0.1% noise.

Model	CGAL	PCL	Robust	DeepFit
Box (249'999 points)	10.1	1.1	13.2	248.7
Box (500'001 points)	17.7	2.2	32.8	487.1
Double Torus (249'994 points)	7.3	3.0	12.7	243.1
Double Torus (499'998 points)	14.8	2.0	28.0	433.7

Table 5.7: Computation time of CGAL, PCL, robust normal estimation (Robust), and DeepFit. The methods were executed without multi-core with all timings in seconds

5.4 Comparing robust normal estimation

We can also compare our implementation of robust normal estimation to the results from Mura et al. [MWP18]. This is possible because we choose similar models and methods to create them. We can compare our results from Figure 5.6, of the emphFandisk, *Box*, *Icosahedron*, and *Double Torus* model with 0.1% noise to theirs. We can see that both implementations have very similar results regarding the percentage of points with an angular error $< 10°$. Mura et al. [MWP18] report percentage values of 92.06%, 96.86%, 93.58%, and 90.90% respectively [MWP18]. Because of this and the visualized quantitative analysis regarding angular deviation, we can conclude that both implementations have similarly good edge-preserving capabilities.

The mean and median values are, however, much higher in our implementation. They report mean values of 4.56, 2.55, 3.16, and 5.04 and median values of 2.10, 1.20, 1.85, and 2.06 for the *Fandisk*, *Box*, *Icosahedron*, and *Double Torus* model respectively [MWP18]. This is also visible in their results for the robustness to noise and outliers. As we do not have access to their original test data and there is no overlap with the other test methods, we cannot exclude that there is not a fundamental difference in the test data, even though our objective was to replicate the test data as closely as possible. It is likely that there is at least one difference in the implementation,

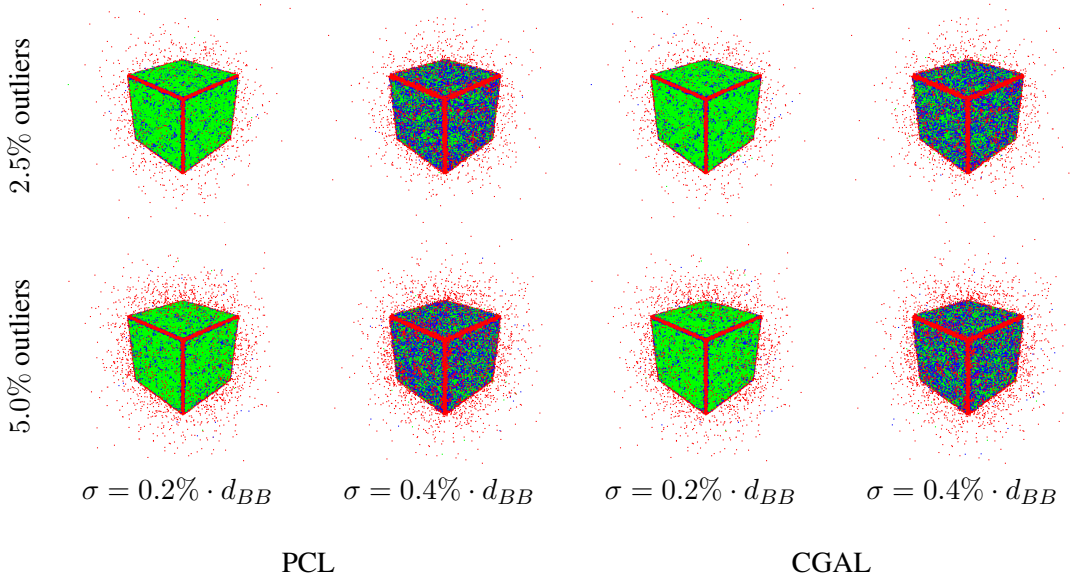


Table 5.8: Analysis of robustness to noise and outliers of the method PCL and CGAL. The color green depicts angular errors $< 5°$, blue angular errors between $\geq 5°$ and $\leq 10°$ and red angular errors $> 10°$

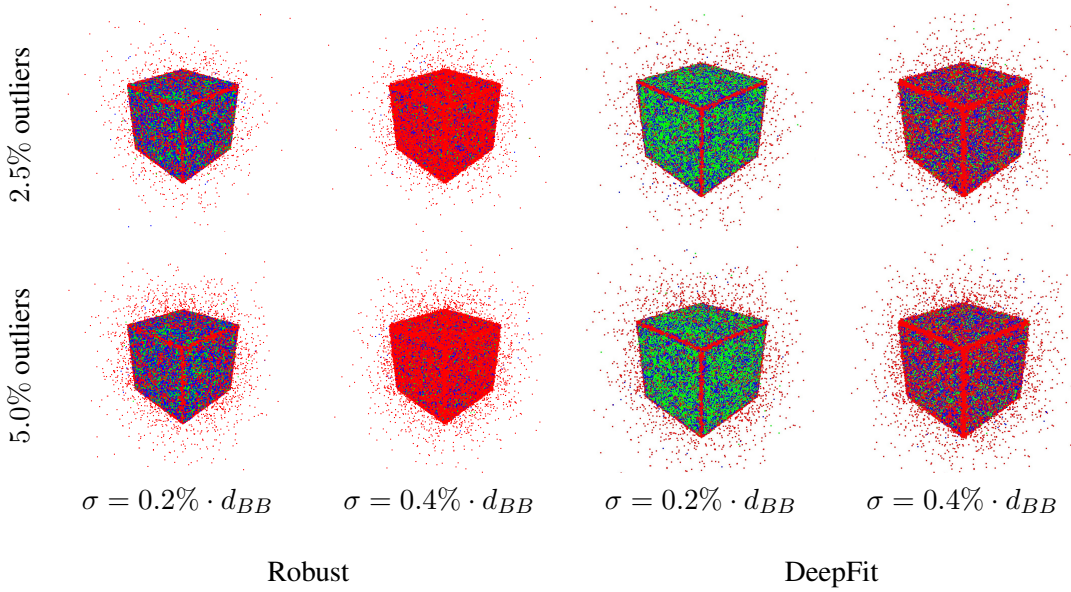


Table 5.9: Analysis of robustness to noise and outliers on the Box model of the method robust normal estimation (Robust) and DeepFit. The color green depicts angular errors $< 5°$, blue angular errors between $\geq 5°$ and $\leq 10°$ and red angular errors $> 10°$

as they did not specify how to extract the patches of the accumulator that overlap with the limited sample space. We cannot exclude that there are further differences as we do not have access to their original implementation.

Method		$p = 0.2\%, 2.5\%$ outl.	$p = 0.2\%, 5.0\%$ outl.	$p = 0.4\%, 2.5\%$ outl.	$p = 0.4\%, 5.0\%$ outl.
CGAL	Mean	3.2	3.24	6.98	7.07
	Median	6.80	8.12	10.83	12.06
	$< 10^\circ$	88.82%	87.00%	71.02%	69.79%
PCL	Mean	3.16	3.20	6.89	6.96
	Median	6.74	8.01	10.72	11.89
	$< 10^\circ$	88.69%	86.88%	71.97%	70.72%
Robust	Mean	7.91	7.90	17.33	17.48
	Median	11.74	13.06	22.49	23.70
	$< 10^\circ$	65.63%	65.16%	22.73%	22.54%
DeepFit	Mean	5.14	5.00	10.73	10.57
	Median	8.75	9.39	14.80	15.20
	$< 10^\circ$	81.92%	81.91%	45.94%	46.79%

Table 5.10: Analysis regarding robustness to noise and outliers on the Box model. The mean and median angular error in degrees, as well as the percentage of points with an angular error $< 10^\circ$, are displayed.

6 Conclusion and Discussion

We have presented a comparison of our implementation of the robust normal estimation method by Mura et al. with two widely used, namely PCA and CGAL, and a state-of-the-art deep learning normal estimation method called DeepFit. We tested the methods with various levels of noise and outliers on five different synthetic models where ground truth data was available. Thus, we examined the mean and median angular error and the percentage of points with an angular error less than 10°. We tested the methods with a real-world point cloud by examining the rendered result using phong-based shading. We improved on the testing by Mura et al. by including an additional model with high detail. Our results have shown that our implementation of robust normal estimation has the highest precision in estimating normals at the sharp features of the model, but is also the most unstable in the presence of noise. We have shown that PCL and CGAL do not preserve sharp features well in contrast to the other two methods, but are the most robust regarding noise. Our examination has shown that the results of DeepFit fall somewhere between the other three methods. We have also shown that all four methods are similarly robust to outliers. PCL was the fastest in terms of execution time, while CGAL and robust normal estimation were very close together with both having adequate results. DeepFit was lagging far behind regarding execution time. We did not test the methods against a difference in sampling density. Furthermore, we did not evaluate how varying the number of the nearest neighbors affects the quality of the methods. We leave both questions for future work. In conclusion, the data suggest that PCL or CGAL is the best choice for models that have large flat or curved surfaces and few sharp features or details that need to be preserved. We have shown that our implementation of robust normal estimation is beneficial for models that are less noisy and need high precision at the sharp features. We have also shown that DeepFit can give generally good results, but it cannot outperform the other methods in any one area, so it is best used if overall good performance is required.

Acknowledgements

I acknowledge the Visualization and MultiMedia Lab at University of Zurich (UZH) and Claudio Mura for the acquisition of the real-world 3D point cloud, and UZH as well as ETH Zürich for their support to scan the rooms represented in this dataset. - The 3D scanning has partially been supported by the EU FP7 People Programme (Marie Curie Actions) under REA Grant Agreement no. 290227. Furthermore, I would like to acknowledge Prof. Dr. Renato Pajarola as he was the module coordinator for the lecture “Computer Graphics” in the Fall Semester 2021 at the University of Zurich (UZH) this is because I used their exercise template as a starting point for my normal estimation / visualisation application. I would also like to acknowledge Joey de Vries, the creator of the online book “Learn OpenGL”, as I used parts of the code that he presents in the online book in my application. Similarly, I would also like to acknowledge the creators of the imgui-filebrowser for their implementation of a file-browser and the GitHub user zfedoran for implementing a progress indicator component as I used both in my implementation of the normal estimation / visualisation application.

I am immensely thankful to Prof. Dr. Renato Pajarola for the privilege of doing my bachelor’s thesis under him. I am extremely grateful to Luciano A. Romero Calla as he assisted me over the course of the thesis with help, valuable inputs and proofreading as a supervisor. I would like to extend my sincere thanks to Florian Hirtz, my fellow friend and co-student, for proofreading my work.

Bibliography

[11] "Estimating surface normals in a pointcloud." (2011), [Online]. Available: https://pcl.readthedocs.io/en/latest/normal_estimation.html (visited on 03/26/2023).

[14] "Dear imgui." (2014), [Online]. Available: https://github.com/ocornut/imgui (visited on 03/26/2023).

[23] *An opengl library*, 2023. [Online]. Available: https://www.glfw.org/ (visited on 03/26/2023).

[AGJ+23] P. Alliez *et al.*, "Point set processing," in *CGAL User and Reference Manual*, 5.5.2, CGAL Editorial Board, 2023. [Online]. Available: https://doc.cgal.org/5.5.2/Manual/packages.html#PkgPointSetProcessing3 (visited on 03/26/2023).

[Ana23] Anaconda Inc., *Anaconda distribution*, 2023. [Online]. Available: https://www.anaconda.com/ (visited on 03/26/2023).

[BELN11] D. Borrmann, J. Elseberg, K. Lingemann, and A. Nüchter, "The 3d hough transform for plane detection in point clouds: A review and a new accumulator design," *3D Research*, vol. 2, no. 2, Art. no. 3, Jun. 2011. DOI: 10.1007/3DRes.02(2011)3.

[BG20] Y. Ben-Shabat and S. Gould, "Deepfit: 3d surface fitting via neural network weighted least squares," in *Proceedings ECCV Computer Vision*, Glasgow, UK: Springer International Publishing, Aug. 2020, pp. 20–34. DOI: 10.1007/978-3-030-58452-8_2.

[Ble18] Blender Online Community, *Blender - a 3d modelling and rendering package*, 2018. [Online]. Available: http://www.blender.org (visited on 03/26/2023).

[BM12] A. Boulch and R. Marlet, "Fast and robust normal estimation for point clouds with sharp features," *Computer Graphics Forum*, vol. 31, no. 5, pp. 1765–1774, Aug. 2012. DOI: https://doi.org/10.1111/j.1467-8659.2012.03181.x.

[CCZ+18] J. Cao, H. Chen, J. Zhang, Y. Li, X. Liu, and C. Zou, "Normal estimation via shifted neighborhood for point cloud," vol. 329, pp. 57–67, Feb. 2018. DOI: 10.1016/j.cam.2017.04.027.

[com14] common-3d-test-models. "Progress indicators (spinner + loading bar)." (2014), [Online]. Available: https://github.com/alecjacobson/common-3d-test-models/blob/master/data/fandisk.obj (visited on 03/26/2023).

[CP05] F. Cazals and M. Pouget, "Estimating differential quantities using polynomial fitting of osculating jets," *Computer Aided Geometric Design*, vol. 22, no. 2, pp. 121–146, Feb. 2005. DOI: 10.1016/j.cagd.2004.09.004.

[DG06] T. K. Dey and S. Goswami, "Provable surface reconstruction from noisy samples," vol. 35, no. 1, pp. 124–141, Aug. 2006. DOI: 10.1016/j.comgeo.2005.10.006.

[DLS05] T. Dey, G. Li, and J. Sun, "Normal estimation for point clouds: A comparison study for a voronoi based method," in *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics*, Jun. 2005, pp. 39–46. DOI: 10.1109/PBG.2005.194062.

[dVri14] J. de Vries. "Welcome to opengl." (2014), [Online]. Available: https://learnopengl.com/ (visited on 03/26/2023).

[GPL] GPL software, *Cloudcompare*, version 2.12.0. [Online]. Available: https://www.cloudcompare.org/ (visited on 03/26/2023).

[HDD+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," in *Proceedings Computer Graphics and Interactive Techniques*, New York, NY, USA: Association for Computing Machinery, 1992, pp. 71–78. DOI: 10.1145/133994.134011.

[IG23] Y. B.-S. (Itzik) and S. Gould, *Deepfit: 3d surface fitting via neural network weighted least squares (eccv 2020 oral)*, 2023. [Online]. Available: https://github.com/sitzikbs/DeepFit (visited on 03/26/2023).

[img19] imgui-filebrowser Contributors. "Imgui-filebrowser." (2019), [Online]. Available: https://github.com/alecjacobson/common-3d-test-models/blob/master/data/fandisk.obj (visited on 03/26/2023).

[KAWB09] K. Klasing, D. Althoff, D. Wollherr, and M. Buss, "Comparison of surface normal estimation methods for range sensing applications," in *Proceedings IEEE Robotics and Automation*, IEEE, May 2009, pp. 3206–3211. DOI: 10.1109/ROBOT.2009.5152493.

[LP05] C. Lange and K. Polthier, "Anisotropic smoothing of point sets," *Computer Aided Geometric Design*, vol. 22, no. 7, pp. 680–692, Oct. 2005. DOI: 10.1016/j.cagd.2005.06.010.

[LZC+15] X. Liu, J. Zhang, J. Cao, B. Li, and L. Liu, "Quality point cloud normal estimation by guided least squares representation," vol. 51, pp. 106–116, Oct. 2015. DOI: 10.1016/j.cag.2015.05.024.

[MN03] N. J. Mitra and A. Nguyen, "Estimating surface normals in noisy point cloud data," in *Proceedings Symposium on Computational Geometry*, San Diego, California, USA: Association for Computing Machinery, Jun. 2003, pp. 322–328. DOI: 10.1145/777792.777840.

[MNG04] N. Mitra, A. Nguyen, and L. Guibas, "Estimating surface normals in noisy point cloud data," vol. 14, no. 4-5, pp. 261–276, Oct. 2004. DOI: 10.1142/S0218195904001470.

[MOG11] Q. Mérigot, M. Ovsjanikov, and L. J. Guibas, "Voronoi-based curvature and feature estimation from point clouds," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 6, pp. 743–756, Jun. 2011. DOI: 10.1109/TVCG.2010.261.

[Mur16] C. Mura. "Point dataset, rooms uzh ifi, 2c05 _ 2." (2016), [Online]. Available: https://www.ifi.uzh.ch/en/vmml/research/datasets.html (visited on 03/26/2023).

[MWP18] C. Mura, G. Wyss, and R. Pajarola, "Robust normal estimation in unstructured 3d point clouds by selective normal space exploration," *The Visual Computer*, vol. 34, no. 6-8, pp. 961–971, Jun. 2018. DOI: 10.1007/s00371-018-1542-6.

[PKG03] M. Pauly, R. Keiser, and M. Gross, "Multi-scale feature extraction on point-sampled surfaces," *Computer Graphics Forum*, vol. 22, no. 3, pp. 281–289, Nov. 2003. DOI: doi.org/10.1111/1467-8659.00675.

[RC11] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Proceedings IEEE Robotics and Automation*, Shanghai, China: IEEE, May 2011, pp. 1–4. DOI: 10.1109/ICRA.2011.5980567.

[RHML17] Q. R. Charles, S. Hao, K. Mo, and G. Leonidas J., "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings Computer Vision and Pattern Recognition*, Los Alamitos, CA, USA: IEEE Computer Society, Jul. 2017, pp. 77–85. DOI: 10.1109/CVPR.2017.16.

[Sha98] C. M. Shakarji, "Multi-scale feature extraction on point-sampled surfaces," *Journal of research of the National Institute of Standards and Technology*, vol. 103, no. 6, pp. 633–64, Nov. 1998. DOI: 10.6028/jres.103.043.

[SS13] C. M. Shakarji and V. Srinivasan, "Theory and algorithms for weighted total least-squares fitting of lines, planes, and parallel planes to support tolerancing standards," *Journal of Computing and Information Science in Engineering*, vol. 13, no. 3, Aug. 2013. DOI: 10.1115/1.4024854.

[Sta14] Stanford University Computer Graphics Laboratory. "Dragon (point cloud)." (2014), [Online]. Available: http://graphics.stanford.edu/data/3Dscanrep/ (visited on 03/26/2023).

[The23a] The CGAL Project, *CGAL User and Reference Manual*, 5.5.2. CGAL Editorial Board, 2023. [Online]. Available: https://doc.cgal.org/5.5.2/Manual/packages.html (visited on 03/26/2023).

[The23b] The CGAL Project, *CGAL User and Reference Manual*, 5.5.2. CGAL Editorial Board, 2023. [Online]. Available: https://doc.cgal.org/latest/Manual/index.html (visited on 03/26/2023).

[YWG+19] Z. Yu, T. Wang, T. Guo, H. Li, and J. Dong, "Robust point cloud normal estimation via neighborhood reconstruction," *Advances in Mechanical Engineering*, vol. 11, no. 4, Apr. 2019. DOI: 10.1177/1687814019836043.

[zfe18] Z. F. (zfedoran). "Progress indicators (spinner + loading bar)." (2018), [Online]. Available: https://github.com/ocornut/imgui/issues/1901#issue-335266223 (visited on 03/26/2023).

[ZPLZ19] R. Zhao, M. Pang, C. Liu, and Y. Zhang, "Robust normal estimation for 3d lidar point clouds in urban environments," *Sensors*, vol. 19, no. 5, 1248, Mar. 2019. DOI: 10.1177/1687814019836043.