Deep Iterative Surface Normal Estimation

Jan Eric Lenssen¹²,*
janeric.lenssen@udo.edu

Christian Osendorfer¹
christian@nnaisense.com

Jonathan Masci¹
jonathan@nnaisense.com

¹ NNAISENSE
² TU Dortmund University

Abstract

This paper presents an end-to-end differentiable algorithm for robust and detail-preserving surface normal estimation on unstructured point-clouds. We utilize graph neural networks to iteratively parameterize an adaptive anisotropic kernel that produces point weights for weighted least-squares plane fitting in local neighborhoods. The approach retains the interpretability and efficiency of traditional sequential plane fitting while benefiting from adaptation to data set statistics through deep learning. This results in a state-of-the-art surface normal estimator that is robust to noise, outliers and point density variation, preserves sharp features through anisotropic kernels and equivariance through a local quaternion-based spatial transformer. Contrary to previous deep learning methods, the proposed approach does not require any hand-crafted features or pre-processing. It improves on the state-of-the-art results while being more than two orders of magnitude faster and more parameter efficient.

1. Introduction

Normal vectors are local surface descriptors that are used as an input for several computer vision tasks ranging from surface reconstruction [27] to registration [39] and segmentation [17]. For this reason, the task of surface normal estimation has been an important and well studied research topic for a long time, with several methods dating back up to 30 years [23]. Progress in the field, however, has been plateauing only until recently when a number of works has shown that improvements can be achieved with the use of data-driven deep learning techniques [5, 7, 19], as also shown in related fields like point cloud denoising [42] or finding correspondences on meshes and point clouds [10, 14, 32, 35]. Deep learning methods are known to often achieve better results compared to data-independent methods. However, they have downsides in terms of robustness to small input changes, adversarial attacks, interpretability, and sometimes also computational efficiency. Also, they do not make use of often well-known intrinsic problem structure, which leads to the necessity of having a large amount of training data and model parameters to learn that structure on their own.

It is well-known that surface normal estimation can be formulated as a least-squares optimization problem. A way to utilize this problem-specific knowledge with deep learning is to take an iteratively reweighting least squares (IRLS) scheme [22] for robust model fitting and modify it using deep data-dependent weighting, as it has been done recently (with or without iterations) for other tasks [25, 43, 44, 46].

Figure 1: Simplified overview of the proposed method for deep iterative surface normal estimation. The figure shows the process for a subset of three points. (a) Surfaces are fitted by optimizing weighted least squares. (b) A graph neural network infers kernel parameters and local orientation from intermediate pair-wise point descriptors. (c) A trainable, adaptive kernel refines the weights for the next step of the least squares optimization.
It is a promising candidate to combine robustness, interpretability and efficiency with the data prior of deep neural networks (DNNs). From a deep learning perspective, the approach imposes a strong bias on the architecture, heavily constraining the space of solutions to those which are better suited for the given problem.

**Contribution** In this work, we present such a trainable re-weighting procedure for input graphs with a large number of weighted least square problems and use it to design a fast and accurate algorithm for surface normal estimation on unstructured point clouds (c.f. Figure 1). The method consists of a light-weight graph neural network (GNN), which parameterizes a local quaternion transformer and a deep kernel function to iteratively re-weight graph edges in a large-scale point neighborhood graph. We show that the resulting algorithm

- reaches state-of-the-art performance in surface normal estimation on unstructured point clouds,
- is more than two orders of magnitude faster and more parameter efficient than related deep learning approaches, and
- is robust to noise and point density variation, while being equivariant and able to preserve sharp features.

## 2. Related work

Traditional methods for surface normal estimation make use of plane fitting approaches like unweighted principal component analysis (PCA) [23] and singular value decomposition (SVD) (c.f. [28] for an overview). The performance of these approaches usually hinges upon the often cumbersome selection of data-specific hyper-parameters, such as point neighborhood sizes, and it is sensitive to noise, outliers and density variations. Because of this, several heuristics have been proposed to ease such selection, e.g. those for finding a neighborhood size for plane fitting [34]. Another limitation of plane fitting methods is that they tend to smooth out sharp details, in fact they can be seen as isotropic low-pass filters. In order to preserve sharp features methods that extract normal vectors from estimated Voronoi cells have been proposed [2, 33] and combined with PCA [1]. Alternative approaches include edge-aware sampling [24] or normal vector estimation in Hough space [6]. In addition, several methods arise from more complex surface reconstruction techniques, e.g. moving least squares (MLS) [30], spherical fitting [18], jet fitting [9] and multi-scale kernel methods [3].

**Deep learning methods.** Deep learning based approaches also found their way into surface normal estimation with the recent success of deep learning in a wide range of domains. These approaches can be divided into two groups, depending on the actual type of input data they use. The first group aims at normal estimation from single images [4, 12, 15, 29, 31, 41, 45] and has received a lot of interest over the last few years due to the well understood properties of CNNs for grid-structured data.

The second line of research directly uses unstructured point clouds and emerged only very recently, partially due to the advent of graph neural networks and geometric deep learning [8]. Boulch et al. [7] proposed to use a CNN on Hough transformed point clouds in order to find surface planes of the point cloud in Hough space. Based on the recently introduced point processing network. PointNet [40], Guerrero et al. [19] proposed a deep multi-scale architecture for surface normal estimation. Later, Ben-Shabat et al. [5] improved on those results using 3D point cloud fisher vectors as input features and a three-dimensional CNN architecture consisting of multiple expert networks.

## 3. Problem and background

Let \( S \) be a manifold in \( \mathbb{R}^3 \), \( P = \{ p_0, \ldots, p_m \} \) a finite set of sampled and possibly distorted points from that manifold and \( N = \{ n_0, \ldots, n_m \} \) the tangent plane normal vectors at sample points \( p_i \). Surface normal estimation for the point cloud \( P \) can be described as the problem of estimating a set of normal vectors \( N = \{ n_0, \ldots, n_m \} \) given \( P \), whose direction match those of the actual surface normals \( n_i \) as close as possible. We consider the problem of unoriented normal estimation, determining the normal vectors up to a sign flip. Estimating the correct sign can be done in a post-processing step, depending on the task at hand, and is explicitly tackled by several works [37, 25, 47].

A standard approach to determine unoriented surface normals is fitting planes to the local neighborhood of every point \( p_i \) [30]. Given a radius \( r \) or a neighborhood size \( k \), we model the input as a nearest neighbor graph \( G = (P, E) \), where we have a directed edge \( (i,j) \in E \) if and only if \( ||p_j - p_i||_2 < r \) or if \( p_j \) is one of the \( k \) nearest neighbors of \( p_i \), respectively. Let \( N(i) \) denote the local neighborhood of \( p_i \), with \( k_i = |N(i)| \), containing all \( p_j \) with \( (i,j) \in E \). Furthermore, let \( P(i) \in \mathbb{R}^{k_i \times 3} \) be the matrix of centered coordinates of the points from this neighborhood, that is

\[
P(i)_j = p_j^\top - \frac{1}{k_i} \sum_{m \in N(i)} p_m^\top, \quad p_j \in N(i). \tag{1}
\]

Fitting a plane to this neighborhood is then described as finding the least squares solution of a homogeneous system of linear equations:

\[
n^*_i = \arg\min_{n} ||P(i) \cdot n||^2_2 = \arg\min_{n} \sum_{j \in N(i)} ||P(i)_j \cdot n||^2_2
\]

\[
\tag{2}
\]
The simple plane fitting of Eq. 2 is not robust and does not result in high-quality normal vectors: It produces accurate results only if there are no outliers in the data, which is never the case in practice. Additionally, this approach eliminates sharp details because it acts as a low-pass filter on the point cloud. Even when an isotropic radial kernel function \( \theta(||P(i)||) \) is used to weight point distances according to their distance to the local mean, fine details cannot be preserved.

Both problems can be resolved through integrating weighting functions into Eq. 2. Sharp features can be preserved with an anisotropic kernel that infers weights of point pairs based on their relative positions, i.e.:

\[
\mathbf{n}_i^* = \arg \min_{\mathbf{n} : ||\mathbf{n}|| = 1} \sum_{j \in \mathcal{N}(i)} \psi(\mathbf{p}_j - \mathbf{p}_i) \cdot ||P(i)_j \cdot \mathbf{n}||^2 \tag{3}
\]

where \( \psi(\cdot) \) is an anisotropic kernel, considering the full Cartesian relationship between neighboring points, instead of only their distance. However, an anisotropic kernel is no longer rotation invariant, so that equivariance of output normals needs to be ensured additionally. Robustness to outliers can be achieved by another kernel that weights points according to an inlier score \( s_{i,j} \). More specifically, Eq. 2 is changed to

\[
\mathbf{n}_i^* = \arg \min_{\mathbf{n} : ||\mathbf{n}|| = 1} \sum_{j \in \mathcal{N}(i)} s_{i,j} \cdot ||P(i)_j \cdot \mathbf{n}||^2 \tag{4}
\]

where \( s_{i,j} \) weights outliers with a low and inliers with a high score. However, in order to infer information about the outlier status of points an initial model estimation is necessary. A standard solution to this circular dependency is to formulate the problem as a sequence of weighted least-squares problems [22, 43]. Given the residuals \( r_i \) of the least squares solution from iteration \( l \), the solution for iteration \( l + 1 \) is computed as

\[
\mathbf{n}_{i}^{l+1} = \arg \min_{\mathbf{n} : ||\mathbf{n}|| = 1} \sum_{j \in \mathcal{N}(i)} s(r_{i,j}^l) \cdot ||P(i)_j \cdot \mathbf{n}||^2 \tag{5}
\]

That is, the inlier score and the estimated model are refined in an alternating fashion.

4. Deep iterative surface normal estimation

In this section we present our method, which combines the described properties of robustness, anisotropy and equivariance with the deep learning property of adaptation to large data set statistics. In contrast to existing deep learning methods [5, 19], we do not directly regress normal vectors from point features but weights for a least-squares optimization step, utilizing the problem specific knowledge outlined above.

The core of the algorithm is a trainable kernel function \( \psi : \mathbb{R}^3 \times \mathbb{R}^d \rightarrow \mathbb{R} \), which computes weights as

\[
\omega_{i,j} = \psi(R_i(p_j - p_i), \theta_i), \tag{6}
\]

\[
\theta \text{ are kernel parameters and } R \text{ is a rotation matrix. The kernel is shared by all local neighborhoods of the point graph while } \theta \text{ and } R \text{ are individual for each node. Because there is no apriori information about the structure of the input data, a reasonable approach is to model } \psi \text{ as an MLP and to find kernel parameters through supervised learning from data. To this end, parameters } \theta \text{ and poses } R \text{ for each neighborhood are jointly regressed by a graph neural network on the point neighborhood graph. Then, the kernel function } \psi \text{ regresses anisotropic, equivariant weights } \omega_{i,j} \text{ for each edge in the graph, which are used to find the normal vectors using traditional weighted least-squares optimization}
\]

\[
\mathbf{n}_i = \arg \min_{\mathbf{n} : ||\mathbf{n}|| = 1} \sum_{j \in \mathcal{N}(i)} \text{softmax}(\omega_{i,j}) ||P(i)_j \cdot \mathbf{n}||^2, \tag{7}
\]

in parallel for all \( i \in \mathcal{P} \). Similar to iterative re-weighting least squares (c.f. Eq. 5), we apply the method in an iterative fashion to achieve robustness and provide the residuals of the previous solution as input to the graph neural network.

The core algorithm is formulated as pseudo code in Algorithm 1. The initial weighting of the points in a neighborhood is chosen to be uniform, which results in unweighted least-squares plane fitting in the initial iteration. In the following, we present the graph neural network, the local quaternion rotation and our differentiable least square solver in more detail.

Algorithm 1 Differentiable iterative normal estimation

**Input:**
- \( \mathcal{P} \): Point cloud
- \( L \): Number of iterations
- \( k \) or \( r \): Neighborhood size (num. neighbors or radius)

**Output:**
- \( \mathbf{N} \): Normal vector estimations

\[
(\mathcal{P}, \mathcal{E}) \leftarrow \text{Neighborhood graph from } \mathcal{P} \text{ and } k/r
\]
\[
\mathbf{C} \leftarrow \text{CovMatrices}(\mathcal{P}, \mathcal{E})
\]
\[
\mathbf{U}, \mathbf{\Sigma} \leftarrow \text{ParallelEig}(\mathbf{C})
\]
\[
\mathbf{N}^0 \leftarrow \text{Extract Solutions from } \mathbf{U}
\]
for \( l \in \{1, \ldots, L\} \) do
\[
(\Theta, \mathbf{Q}) \leftarrow \text{GNN}(\mathcal{P}, \mathcal{E}, \mathbf{N}^{l-1})
\]
\[
\mathbf{R} \leftarrow \text{QuatsToMats}(\mathbf{Q})
\]
\[
\mathbf{W} \leftarrow \text{ApplyKernel}(\mathbf{R}, \mathcal{P}, \Theta, \mathcal{E})
\]
\[
\mathbf{C} \leftarrow \text{WeightedCovMatrices}(\mathcal{P}, \mathbf{W}, \mathcal{E})
\]
\[
\mathbf{U}, \mathbf{\Sigma} \leftarrow \text{ParallelEig}(\mathbf{C})
\]
\[
\mathbf{N}^l \leftarrow \text{Extract Solutions from } \mathbf{U}
\]
end for
return \( \mathbf{N}^L \)
4.1. GNN for parameterization and rotation

For regressing parameters $\theta$ and rotations $\mathbf{R}$ for the whole point cloud, graph neural networks [13, 20] are a natural fit because the network must be invariant to the ordering of the points in a neighborhood and it must be able to allow weight sharing over neighborhoods with varying cardinality.

Our graph neural network architecture consists of a neighborhood aggregation procedure, which is applied three consecutive times. Given MLPs $h$ and $\gamma$, the neighborhood aggregation scheme, similar to that of PointNet [40] and to general message passing graph neural network frameworks [20, 36], is given by message function

$$f_e(i, j) = h(f(i) | d_{i,j} | \text{prf}(i, j)), \quad (8)$$

and node update function

$$f(i) = \gamma\left(\frac{1}{|N(i)|} \sum_{j \in N(i)} f_e(i, j)\right), \quad (9)$$

with $|$ denoting feature concatenation. Using this scheme, we alternate between computing new edge features $f_e(i, j)$ and node features $f(i)$. In addition to the Cartesian relation vector $d_{i,j} = (p_j - p_i)$, pair-wise residual features, a modified version of Point Pair Features (PPF) [10, 11], are provided as input:

$$\text{prf}(i, j) = (|n_i \cdot d_{i,j}|, |n_j \cdot d_{i,j}|, |n_i \cdot n_j|, ||d_{i,j}||^2_2). \quad (10)$$

They are computed directly from the last set of least-squares solutions and contain the residuals as point-plane distances $|n_i \cdot d_{i,j}|$.

After applying the message passing scheme, the output node feature matrix $\mathbf{F} \in \mathbb{R}^{N \times (d+4)}$ is interpreted as a tuple $(\Theta \in \mathbb{R}^{N \times d}, \mathbf{Q} \in \mathbb{R}^{N \times 4})$, containing kernel and rotation parameters for all nodes. We use the row-normalized $\mathbf{Q}$ as unit quaternions to efficiently parameterize the rotation group $SO(3)$. We found that using a rotation matrix instead of an arbitrary $3 \times 3$ matrix (as in the Spatial Transformer Network [26]) heavily improves training stability, as also observed by Guerrero et al. [19]. By applying a custom, differentiable map from quaternion space to the space of rotation matrices we efficiently compute the local rotation matrices $\mathbf{R}$ for all nodes in parallel.

All in all, the graph neural network is permutation invariant, can be efficiently applied in parallel on varying neighborhood sizes, and is a local operator. Locality is an advantage which allows the algorithm to be applied on partial point clouds and scans, without relying on global features or semantics.

4.2. Parallel differentiable least-squares

In every iteration of the presented algorithm, the plane fitting problem of Eq. 7 needs to be solved. A standard approach is to utilize the Singular Value Decomposition of the weighted matrix $\text{diag}(\sqrt{w_i})\mathbf{P}(i)$: Let $\mathbf{U \Sigma V}^T$ be its decomposition, then the column vector of $\mathbf{V}$ corresponding to the smallest singular value is the optimal solution for the given least squares problem [21, 43]. However, $n$ SVDs (for potentially varying matrix sizes) need to be solved in our scenario, one for every neighborhood, which makes this approach prohibitive. A much more efficient approach in this case is to consider the eigendecomposition of the weighted $3 \times 3$ covariance matrix $\mathbf{C}(i) = \mathbf{P}(i)^{-1} \text{diag}(w_i)\mathbf{P}(i)$ which has the columns of $\mathbf{V}$ as its eigenvectors [21]. The solution for Eq. 7 is then the eigenvector associated with the smallest eigenvalue. The computational complexity for the eigendecomposition of this $3 \times 3$ matrix is $O(1)$ and hence for one overall iteration $O(n)$.

Our algorithm is trained end-to-end by minimizing the distance between ground truth normals and the least squares solution, requiring backpropagation through the eigendecomposition. We follow the work of Giles [16]: Given partial derivatives $\partial L/\partial \mathbf{U}$ and $\partial L/\partial \mathbf{\Sigma}$ for eigenvectors and eigenvalues, respectively, we compute the partial derivatives for a real symmetric $3 \times 3$ covariance matrix $\mathbf{C}$ as

$$\frac{\partial L}{\partial \mathbf{C}} = \mathbf{U}(\mathbf{\Sigma})^{-\text{diag}} + \mathbf{F} \circ \mathbf{U}^T \frac{\partial L}{\partial \mathbf{U}} \mathbf{U}^T, \quad (11)$$

where $F_{i,j} = (\lambda_j - \lambda_i)^{-1}$ contains inverse eigenvalue differences. We implemented forward and backward steps for eigendecomposition of a large number of symmetric $3 \times 3$ matrices, where we parallelize over graph nodes, leading to an $O(1)$ implementation (using $O(n)$ processors) of parallel least squares solvers.

Handling numerical instability. Backpropagation through the eigendecomposition can lead to numerical instabilities due to at least two reasons: 1) Low-rank input matrices with two or more zero eigenvalues. 2) Exploding gradients when two eigenvalues are very close to each other and values of $\mathbf{F}$ go to infinity. We apply two tricks to avoid these problems. First, a small amount of noise is added to the diagonal elements of all covariance matrices, making them full-rank. Second, gradients are clipped after the backward step on very large values, to tackle the cases of nearly equal eigenvalues that lead to exploding gradients.

4.3. Training

Training is performed by minimizing the Euclidean distance between estimated normals $\hat{\mathbf{N}}$ and ground truth normals $\mathbf{N}$, averaged over all normal vectors in the training set:

$$L(\hat{\mathbf{N}}, \mathbf{N}) = \frac{1}{n} \sum_{i=1}^{n} \min(||\hat{n}_i - n_i||_2, ||\hat{n}_i + n_i||_2), \quad (12)$$

where the minimum of the distances to the flipped or non-flipped ground truth vectors is used. While we experi-
Table 1: Results for unoriented normal estimation. Shown are normal estimation errors in angle RMSE. For PCA and Jet, optimal neighborhood size for average error is chosen. For our approach, we display results for a balanced neighborhood size $k = 64$, which improves on the state of the art for all noise levels. Results for different $k$ are shown in Table 2.

5. Experiments

Experiments were conducted to compare the proposed Differentiable Iterative Surface Normal Estimation with state-of-the-art methods both quantitatively, measuring normal estimation accuracy, and qualitatively, on a Poisson reconstruction and on a transfer learning task. Section 5.1 introduces the dataset used to train our model whereas Section 5.2 details the architecture and the protocol followed in our experiments. Then, qualitative (Section 5.3) and quantitative (Section 5.5) results are presented and an analysis of complexity and execution time (Section 5.4) is given.

5.1. PCPNet dataset

Our method is trained and validated quantitatively on the PCPNet dataset as provided by Guerrero et al. [19]. It consists of a mixture of high-resolution scans, point clouds sampled from handmade mesh surfaces and differentiable surfaces. Each point cloud consists of 100k points. We reproduce the experimental setup of [5, 19], training on the provided split containing 32 point clouds under different levels of noise. The test set consists of six categories, containing four sets with different levels of noise (no noise, $\sigma = 0.00125$, $\sigma = 0.0065$ and $\sigma = 0.012$) and two sets with different sampling density (striped pattern and gradient pattern). We evaluate unoriented normal estimation, same as the related approaches. The Root Mean Squared Error (RMSE) on the provided 5k points subset is used as performance metric following the protocol of related work, where the RMSE is first computed for each test point cloud before the results are averaged over all point clouds in one category. Model selection is performed using the provided validation set.

5.2. Experimental setup and architecture

The presented graph neural network was implemented using the Pytorch Geometric library [13]. The neural networks $h$, $\gamma$ and $\psi$ each consist of two linear layers, with ReLU non-linearity. A detailed description of the architecture is presented in the supplemental materials. During training, output weights from the kernel are randomly set to zero with probability of 0.25.

It should be noted that despite inheriting the neighborhood size parameter from traditional PCA, it is possible for a network trained on a specific $k$ to be applied for other $k$ as well. This is because all networks can be shared across an arbitrary number of points and the softmax function normalizes weights for neighborhoods of varying sizes. We observed that generalization across different $k$ only leads to a very small increase in average error. However, to fairly evaluate our method for different $k$, a network is trained for each $k \in \{32, 48, 64, 96, 128\}$. Trained consists of 300 epochs using the RMSProp optimization method. All reported test results are given after 4 re-weighting iterations of our algorithm. Iterating longer does not show significant improvements. Quantitative results over iterations, results for extrapolation over iterations and generalization between different $k$ are presented in the supplemental materials. For further realization details, we refer to our implementation, which we will make available online prior to publication.

5.3. Quantitative Evaluation

RMSE results for the PCPNet test set of our approach (with $k = 64$) and related works are shown in Table 1. We improve on the state of the art on all noise levels and varying densities. While the improvement is only small, it should be noted that we reach it while being orders of magnitude faster and more parameter efficient (c.f. Section 5.4, which is of importance for many applications in resource constraint environments. For the non deep learn-
As expected, varying $k$ leads to a behavior similar to that of PCA, with large $k$’s performing better on more noisy data. However, it can be observed that our approach is more robust to changes of $k$: Even for small neighborhood sizes, high noise is handled significantly better than by PCA and large neighborhoods still produce satisfactory results for low noise data. It should be noted that for all evaluated $k$ we improve on the state of the art w.r.t. average error. An evaluation for smaller $k$ down to $k = 2$ is provided in the supplemental materials.

While the RMSE error metric is well suited for a general comparison, it is not a good proxy to estimate the ability of recovering sharp features since it does not take into account the error distribution over angles. Therefore, as an additional metric, Figure 2 presents the percentage of angle errors falling below different angle thresholds. The results confirm that our approach is better at preserving details and sharp edges, especially for low noise point clouds and varying density, where it outperforms other approaches. For higher noise, results similar to Nesti-Net are achieved.

### 5.4. Efficiency

Our model is small, consisting of only 7981 trainable parameters, shared over iterations and spatial locations. On a single Nvidia Titan Xp, a point cloud with 100k points is processed in 5.67 seconds (0.0567 ms per point). A large part of this is the kd-tree used to compute the nearest neighbor graph, which takes 2.1 seconds of the 5.67 seconds. It is run on the CPU and could be further sped-up by utilizing GPUs.

Table 3: Comparison of efficiency between the approaches using deep learning. We list number of model parameters as well as average execution times for estimating normals on a point cloud with 100k points.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. parameters</td>
<td>7981</td>
<td>179M</td>
<td>22M</td>
</tr>
<tr>
<td>Exec. time, 100k p.</td>
<td>3.57 s</td>
<td>1350 s</td>
<td>470 s</td>
</tr>
<tr>
<td>Relative exec. time</td>
<td>1 × 378 ×</td>
<td>131 ×</td>
<td></td>
</tr>
</tbody>
</table>

In Table 3 we compare our approach against the related deep learning approaches Nesti-Net and PCPNet. Our approach is orders of magnitude (378× and 131×) faster than the related approaches. The comparison was made...
as fair as possible by excluding nearest neighbor queries (note that this favors the other approaches since they need larger neighborhoods) and the original implementations. The speedup of our method can be contributed to the much smaller network size and the parallel design of the GNN and least-squares optimization steps.

5.5. Qualitative Evaluation

This section visually presents surface normal errors for various elements of the PCPNet test set in Figure 3 and compares them against results from the PCA baseline and related deep learning approaches. It can be seen that the biggest improvements are obtained for low noise scenarios and varying density, where our method is able to preserve sharp features of objects better than the other methods. In general it can be observed that our approach tends to provide sharp, discriminative normals for points on edges instead of smooth averages. In rare cases, this can lead to a false assignment of points to planes, as we can see in the example in column 8. It can be observed that, in contrast to Nesti-Net, our approach behaves equivariant to input rotation as is seen clearly on the diagonal edge of the box example in column 3. Sharp edges are kept also in uncommon rotations, which we can attribute to our local rotational transformer. Results for more examples are displayed in the supplemental material.

Interpretability. In order to interpret the results of our method, Figure 4 shows a detailed view of local neighborhoods over several iterations of our algorithm. An example for a sharp edge is shown in Figure 4a and a high noise surface in Figure 4b. Both sets of points were sampled from the real test data. For the sharp edge, the algorithm initially fits a plane with uniform weights, leading to smoothed normals. Over the iterations, high weights concentrate on the more plausible plane, leading to recovering of the sharp edge. In the noisy example, we can see that outliers are iteratively receiving lower weights, leading to more stable estimates.
Ours Nesti-Net PCPNet PCA

Figure 5: Selected results after applying Poisson surface reconstruction using the estimated normal vectors. In most cases, differences between the methods are very small. Examples 2 and 3 show reconstructions from point clouds with varying density, which show the largest differences.

Surface reconstruction. To further evaluate the quality of the produced normals when used as input to other computer vision pipelines, Figure 5 shows the results for Poisson surface reconstruction. Since the methods in this comparison all perform unoriented normal estimation (Guerrero et al. [19] evaluates both, unoriented and oriented, where we chose the unoriented version for a fair comparison), we determine the signs of the output normals from all four methods using the ground truth normals. Most of the reconstructions show only small differences, with our approach and Nesti-Net retaining slightly more details than the others. Significant differences can be observed for point clouds with varying density, displayed in rows 2 and 3. Here, our approach successfully retains the original structure of the object while still providing sharp edges.

Transfer to NYU depth dataset. In order to show generality of our approach, our models trained on the PCPNet dataset are validated on the NYU depth v2 dataset [38], a common benchmark dataset in the field of estimating normals from single images. It contains 1449 aligned and preprocessed RGBD frames, which are transformed to a point cloud before applying our method. After performing unoriented estimation, the normals are flipped towards the camera position. Evaluation is done qualitatively, since the dataset does not contain ground truth normal vectors. Results for three different neighborhood sizes in comparison to PCA are shown in Figure 6. Our approach behaves as expected, as it is able to infer plausible normals for the given scenes. For all $k$, our approach is able to preserve sharp features while PCA produces very smooth results. However, this also leads to the sharp extraction of scanning artifacts, which can be seen on the walls of the scanned room.

6. Conclusion and future work

We presented a novel method for deep surface normal estimation on unstructured point clouds, consisting of parallel, differentiable least-squares optimization and deep re-weighting. In each iterations, the weights are computed using a kernel function that is individually parameterized and rotated for each neighborhood by a task-specific graph neural network. The algorithm is much more efficient than previous deep learning methods, reaches state-of-the-art accuracy, and has favorable properties like equivariance and robustness to noise. For future work, investigating the possibility of utilizing deep data priors to parameterize least-squares problems holds large potential. We suspect that introducing data-dependency to other traditional methods can lead to progress in other fields of research, by reducing common disadvantages of pure deep learning approaches. On the theoretical side, it is interesting to dive deeper into convergence properties of IRLS with deep re-weighting.

Acknowledgements

We thank Matthias Fey for his work on Pytorch Geometric. We also thank him and Prof. Dr. Heinrich Müller for helpful advice and discussions.
References


