LiSu: A Dataset and Method for LiDAR Surface Normal Estimation

Supplementary Material

This supplementary material provides an in-depth analysis of inference speed for the models used in our benchmarks (Suppl. A). Additionally, it offers comprehensive details on the acquisition of our LiSu dataset (Suppl. B) and the implementation specifics of baseline methods (Suppl. C). Furthermore, we present additional experiments for the neural surface reconstruction downstream task (Suppl. D), qualitative evaluations (Suppl. E), and a rigorous ablation study exploring the impact of various design choices (Suppl. F, Suppl. G, Suppl. H).

A. Inference Speed *vs*. Accuracy

Traditional methods like PCA [9] and Jet [1] are renowned for their efficient runtime. However, their accuracy is degraded by inherent rotation ambiguities. This often manifests as points on the same plane exhibiting opposite surface normal directions (recall Fig. 4 of the main manuscript). Common heuristics, such as orienting all normals towards a fixed viewpoint or propagating orientation information via Minimum Spanning Tree (MST) [9], alleviate this problem but, due to the noisy nature of LiDAR point clouds, are not particularly effective.

Supervised methods like SHS-Net [12], Du *et al*. [5], GraphFit [10], PCPNet [7], NeuralGF [13], CMG-Net [18], and NGL [11] offer substantial performance gains, but at the cost of significant computational overhead. A key limitation of these methods is their reliance on point cloud partitioning, required during both training and inference. Furthermore, they often employ point-based backbone architectures like PointNet [14] or architectures which require special operations such as DGCNN [16], which hinder efficient processing. Originally introduced to address the limited dataset size of PCPNet [7] (30 samples), point cloud partitioning remains necessary during inference, significantly increasing processing time, especially for large-scale datasets like ours, LiSu (approximately 100k points per frame). Batching partitions is a potential strategy for accelerating inference. However, GPU VRAM limits batch sizes, preventing single-frame inference and necessitating multiple inference passes for batched partitions. Moreover, PointNet and DGCNN are not optimized for large-scale point clouds, making it challenging to adapt them to single-frame training/inference.

Conversely, we leverage the Point Transformer V3 (PTv3) [17], a state-of-the-art transformer architecture build for large-scale LiDAR point clouds. Their employment of space-filling curves (*e.g*. z-order or Hilbert curve) for point cloud serialization and hardware-optimized operations (*e.g*. FlashAttention [2, 3]) enable significant speedups. A single inference step on a large-scale LiDAR point cloud can be executed in orders of magnitude less time (*e.g*. 50 milliseconds vs. 20 seconds for DGCNN). PTv3 coupled with our novel LiSu and training method exhibits exceptional performance in LiDAR surface normal estimation, requiring significantly less computational time compared to existing methods, as visualized in Fig. 1.

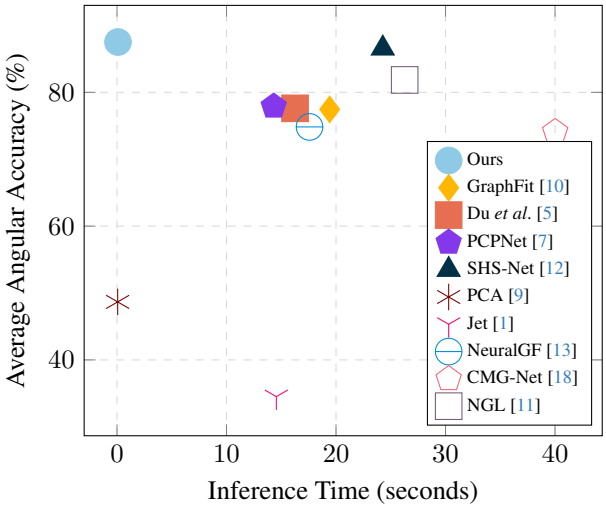


Figure 1. Inference speed *vs*. accuracy plot for various neural network-based and traditional methods. Accuracy is calculated as the average angular accuracy across all thresholds listed in Tab. 2 of the main manuscript: $\{5.0°, 7.5°, 11.25°, 22.5°, 30.0°\}$.

B. LiSu Acquisition

To obtain our custom LiSu dataset, we extended the CARLA simulator (version 0.9.15[1]) built with Unreal Engine 4.26[2] with an additional LiDAR sensor. This sensor captures not only standard position and intensity data but also the surface normal vector of each hit point. We extended CARLA's ray caster to return surface normals in the sensor's frame of reference and stream this data from the C++ core. This data stream was collect by the Python front-end and saved to a file together with the global sensor position and orientation (required for keyframe transformation in TGTV in Sec. 3.2 of the main manuscript)

We initialize a virtual LiDAR sensor to reflect commonly employed real-world LiDARs [6, 15]. A detailed configuration of this sensor is presented in Table 1. The virtual LiDAR

[1] https://github.com/carla-simulator/carla/releases/tag/0.9.15

[2] https://github.com/CarlaUnreal/UnrealEngine

(a) Town02 (b) Town03

Figure 2. Bird's eye view images of CARLA towns. Images taken from https://carla.readthedocs.io/en/latest/core_map/#carla-maps

Description	Value
Number of lasers	64
Maximum distance to raycast in meters	100
Points generated by all lasers per second	2M
LIDAR rotation frequency	10 Hz
Angle of the highest beam	10°
Angle of the lowest beam	−30°
Horizontal field of view	360°
Proportion of randomly dropped points	0.45
Std Dev point noise along the raycast vector	0.02

Table 1. Virtual LiDAR attributes used for the data acquisition.

sensor was mounted on a self-driving car, randomly placed within the simulation environment. We populated the simulation environment with approximately 6000 dynamic actors, such as vehicles (cars, trucks, buses, vans, motorcycles, bicycles) and pedestrians (adults, children, police) as well as 2000 static props (barrels, garbage cans, road barriers, *etc*.). Due to the different map sizes (*e.g*. Fig. 2a *vs*. Fig. 2b), not all objects were guaranteed to appear in every simulation. For each map, we conducted N independent runs, each initialized with a different random seed. To avoid redundant frames, we terminated simulations prematurely if prolonged traffic halts, such as those caused by red lights, occurred. A detailed breakdown of simulation runs is provided in Tab. 2.

C. Implementation Details of Other Methods

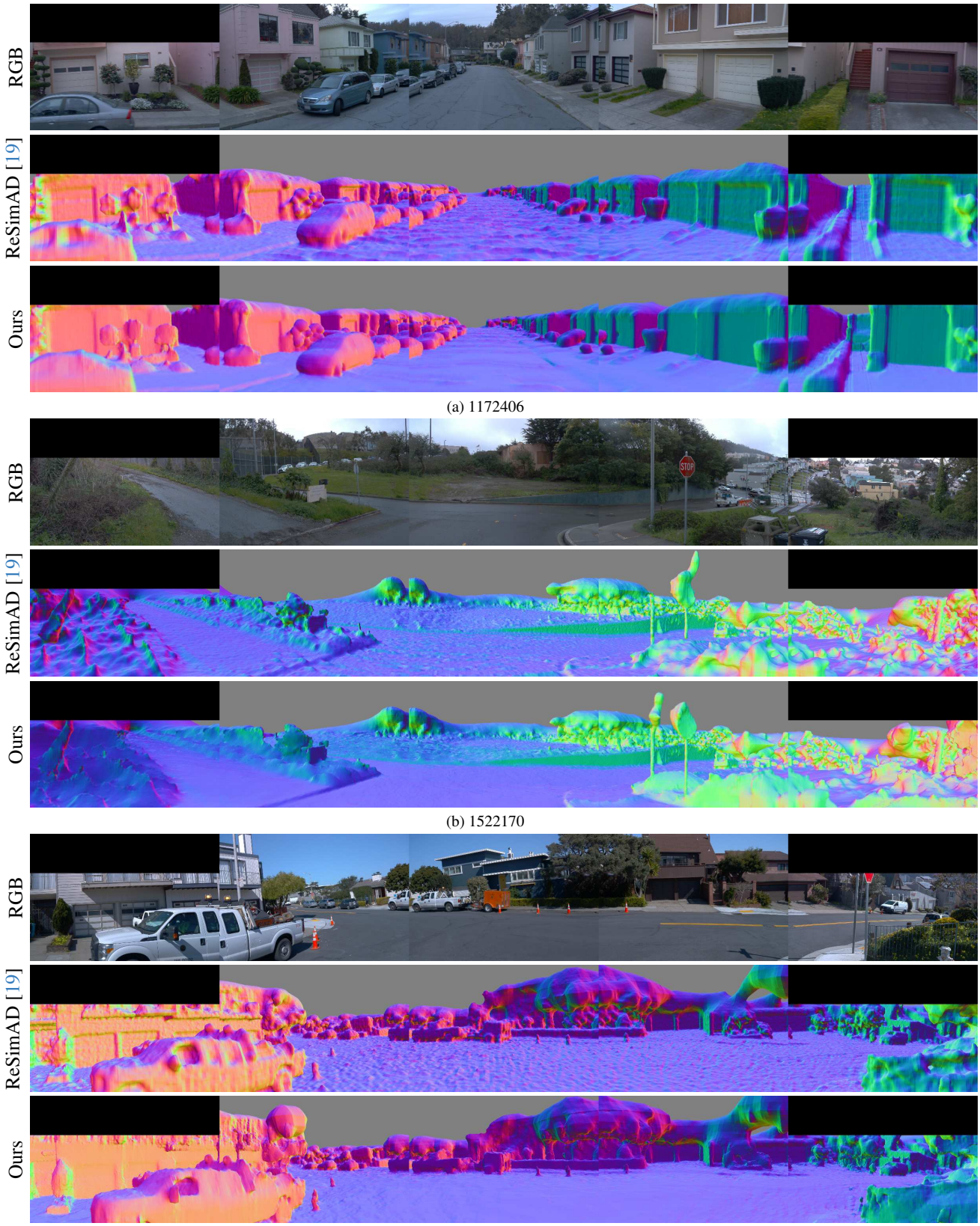
We benchmarked our method against several state-of-the-art point cloud surface normal estimation methods: PCPNet [7], GraphFit [10], Du *et al*. [5], and SHS-Net [12]. These methods share a common training strategy, originally proposed in PCPNet, which involves randomly sampling query points and their k-nearest neighbors from each training sample. While PCPNet's dataset consists of dense point clouds, our LiDAR data is significantly sparser. Consequently, we reduced the neighborhood size k to 32 to better adapt to the sparse nature of our data. Furthermore, given our larger dataset, we decreased the number of training epochs to 10 for all methods. All other hyperparameters were kept consistent with their original settings.

D. Neural Surface Reconstruction

Our experiments in Sec. 4.3 closely replicate existing benchmarks in neural surface reconstruction from LiDAR data [8, 19]. In these benchmarks, a mesh reconstructed from all available LiDAR data is queried with rays generated from the same data, and the resulting distances are compared to actual LiDAR measurements. While this approach offers a convenient evaluation framework, it may not accurately reflect the method's true performance, as the same data is used both in training and evaluation.

Therefore, we propose a more rigorous evaluation protocol. We randomly split LiDAR points from an entire sequence into two disjoint sets: a training and a testing set. The training split is used exclusively in the training phase. Subsequently, the reconstructed mesh is evaluated using the unseen testing set. By ensuring that the training and testing sets are mutually exclusive, we can better identify model's potential limitations.

The proposed evaluation protocol proves particularly challenging for plain ReSimAD [19], as evident from the mesh reconstruction in Fig. 3 and LiDAR simulation in Fig. 4. Limited training data hinders SDF generalization, leading to poor extrapolation in areas lacking ground truth signals, such as ridges in the mesh (Fig. 3). This noise propagates to the LiDAR simulation (Fig. 4), resulting in highly noisy point clouds. Incorporating surface normals as an additional training signal mitigates these issues, leading to smoother meshes and consequently cleaner point clouds. This improvement is also quantified in Tab. 3



(a) 1172406

(b) 1522170

(c) 1006130

Figure 3. Mesh reconstruction results comparing plain ReSimAD [19] and our method, which incorporates surface normals estimated from a model trained on our LiSu dataset and fine-tuned on Waymo Open Dataset [15]. Results are shown for three different Waymo sequences.

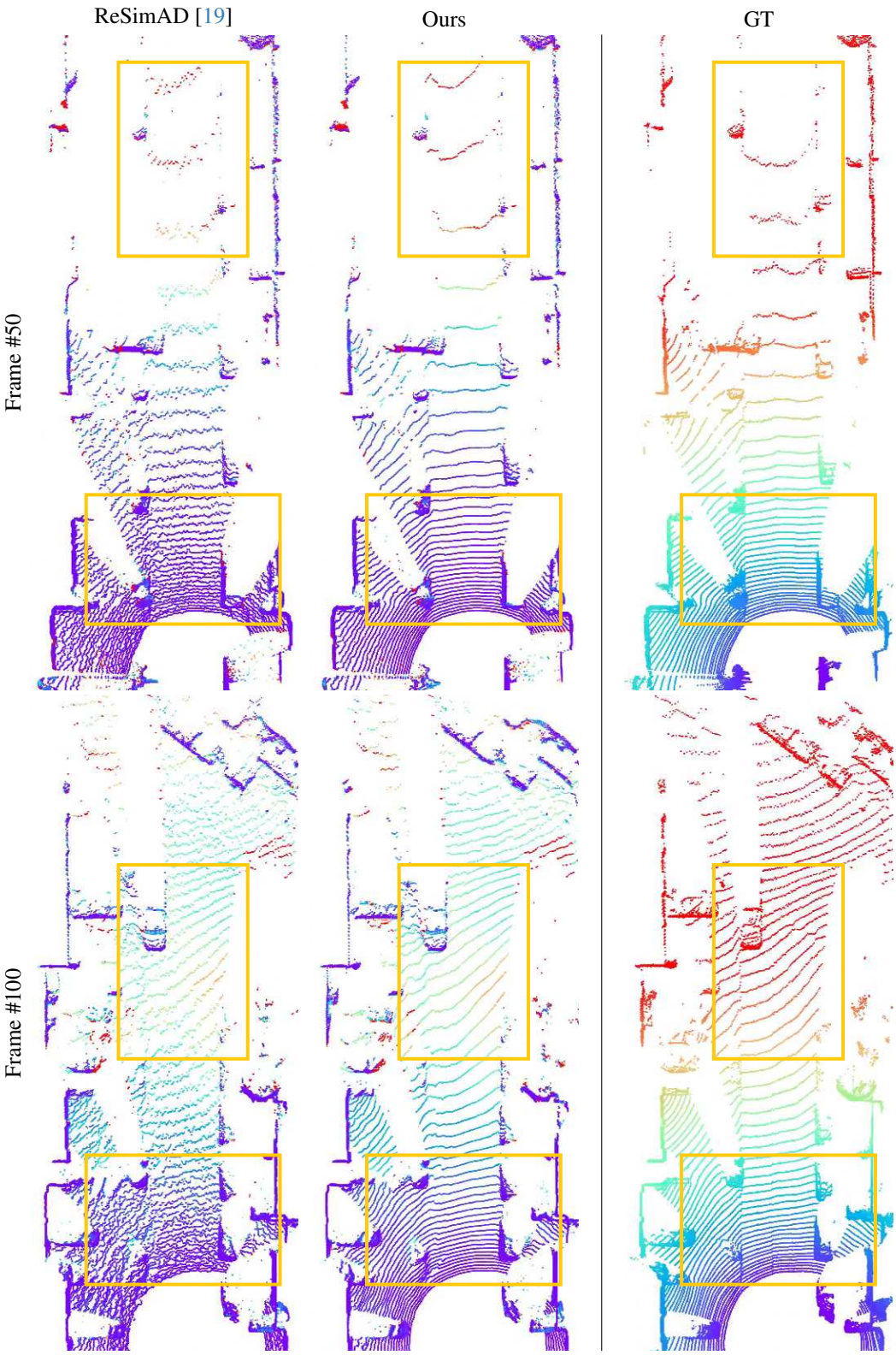


Figure 4. LiDAR simulation results comparing mesh reconstructed with plain ReSimAD [19] (left) and our method, which leverages estimated surface normals from a model trained on our LiSu dataset and fine-tuned on Waymo Open Dataset [15] (middle). The color scale represents the deviation from ground truth distance, with blue indicating low error and red indicating high error. The rightmost image shows the ground truth LiDAR point cloud (used for mesh reconstruction), colored by distance from the sensor (blue: near, red: far).

	Map	# Runs	# Frames	Summary
train	Town01	11	6339	A small, simple town with a river and several bridges.
	Town03	11	7658	A larger, urban map with a roundabout and large junctions.
	Town05	11	5477	Squared-grid town with cross junctions and a bridge. It has multiple lanes per direction.
	Town07	11	5579	A rural environment with narrow roads, corn, barns and hardly any traffic lights.
total	4	44	25 053	
test	Town02	11	5235	A small simple town with a mixture of residential and commercial buildings.
	Town04	5	3591	A small town embedded in the mountains with a special "figure of 8"' infinite highway.
	Town06	11	3980	Long many lane highways with many highway entrances and exits (with Michigan left).
	Town12	16	9361	A large map, including high-rise, residential and rural environments.
total	4	43	22 167	
val	Town10	11	2825	A downtown area with skyscrapers, residential buildings and an ocean promenade.
total	1	11	2825	
overall	9	98	50 045	

Table 2. Summary of our data splits, including CARLA [4] maps, number of randomly started simulation runs, and total number of frames for the given map. We include the short summary provided by CARLA for each map.

Seq.	ReSimAD [19]	Ours
1027514	0.68 / 0.15	**0.65 / 0.14**
1006130	**2.55 / 0.23**	2.56 / 0.25
1137922	0.95 / 0.77	**0.92 / 0.73**
1323841	**0.51 / 0.11**	**0.51** / 0.12
1486973	0.83 / **0.16**	**0.82** / 0.17
1522170	2.05 / 2.70	**1.78 / 2.06**
1647019	1.01 / 0.70	**0.98 / 0.63**
3425716	0.70 / 0.22	**0.68 / 0.20**
9385013	0.52 / 0.20	**0.51 / 0.18**
average	1.09 / 0.58	**1.04 / 0.50**

Table 3. We evaluate neural surface reconstruction on diverse Waymo sequences using Root Mean Square Error (RMSE) / Chamfer Distance (CD). Lower values indicate better performance. ReSimAD [19] omits surface normal loss during reconstruction. Ours is a Waymo model, trained with our proposed self-training framework.

E. Qualitative Evaluation

We conduct a qualitative evaluation on the Waymo Open Dataset [15] to highlight the benefits of our LiSu. By comparing SHS-Net [12] trained on PCPNet [7] to our LiSu, we demonstrate the significant advantage of leveraging a dataset tailored to real-world LiDAR point clouds. The lack of publicly available LiDAR datasets underscores the potential of our LiSu to advance the field, regardless of the specific method employed. Notably, our self-supervised approach achieves impressive results when applied to a real-world dataset like Waymo, as visualized in Fig. 5.

F. Loss-Regularization Trade-off

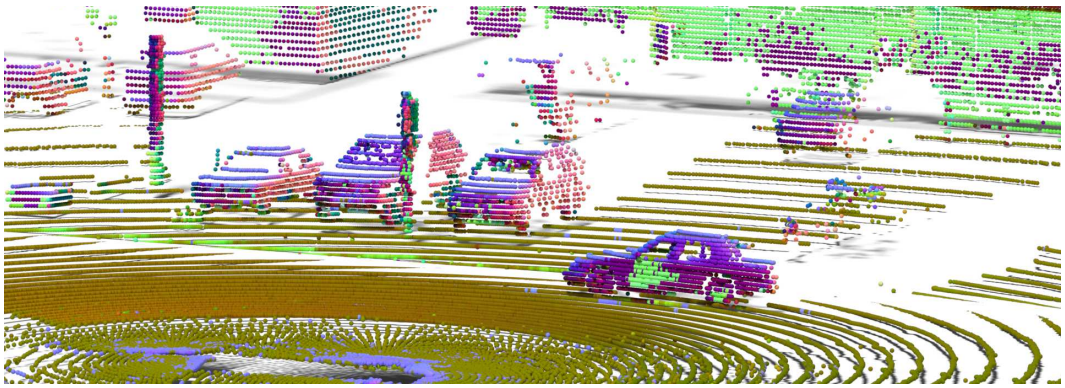
In the following section, we present an ablation study to substantiate our selection of the γ parameter (Eq. (7) of the main manuscript), which balances loss minimization and regularization. To expedite the training and evaluation phases, we utilized 50% and 20% subsets of the original training and evaluation data, respectively.

The hyperparameter γ balances the trade-off between noise and smoothness in the final predictions. Lower values of γ encourage the model to prioritize edge preservation, potentially leading to noisier predictions. Conversely, higher values of γ promote smoother predictions, which may result in the loss of fine-grained details and edge information. In the extreme case, when $\gamma = 1$, the model's predictions become almost entirely smooth, with minimal edge detection. In our experiments, we found that $\gamma = 0.1$ provided an optimal balance between noise and detail, as illustrated in Fig. 6.

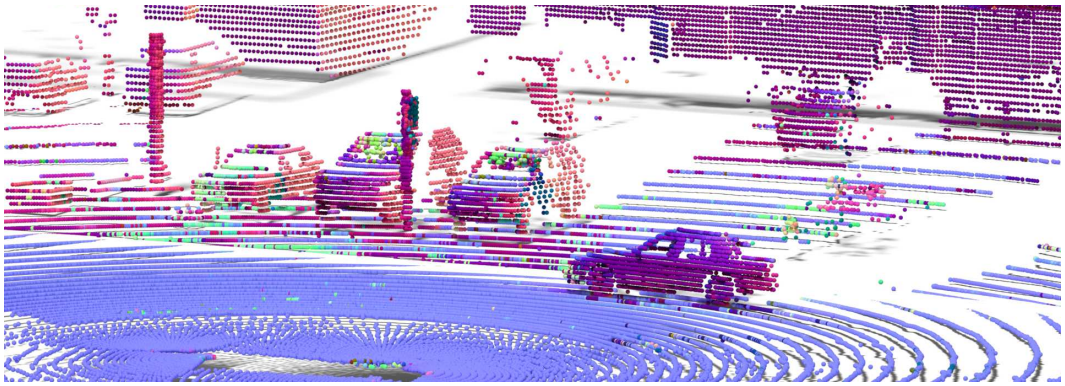
G. k Neighborhood Graph Ablation

To construct the k-neighborhood graph $\mathcal{G}$ for both Spatial Graph Total Variation (SGTV) and Temporal Graph Total Variation (TGTV) (Sec. 3.2), we require a hyperparameter k to specify the size of the local neighborhood. In the following experiments, we fix $\gamma = 0.1$ and systematically vary k to assess its influence on the model's overall performance. To expedite the training and evaluation phases, we utilized 50% and 20% subsets of the original training and evaluation data, respectively.

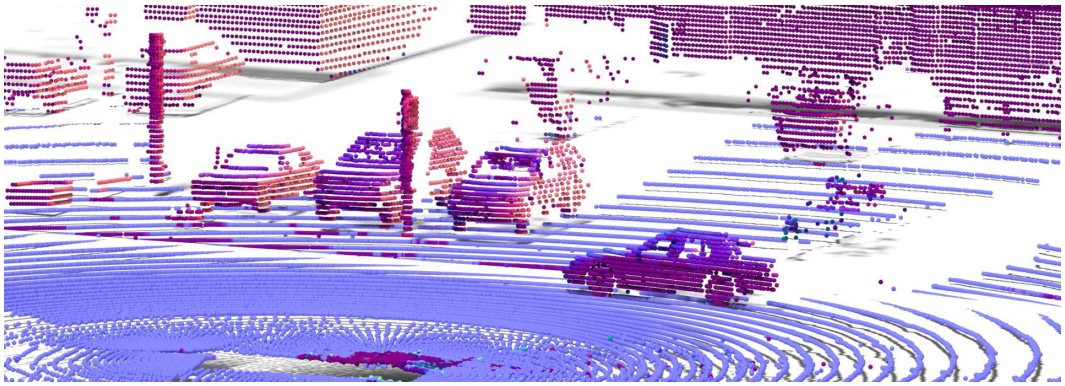
The parameter k controls the model's output smoothness, with higher values leading to increased smoothing and potential loss of detail (*e.g.* $k = 32$ in Fig. 7). Conversely, smaller



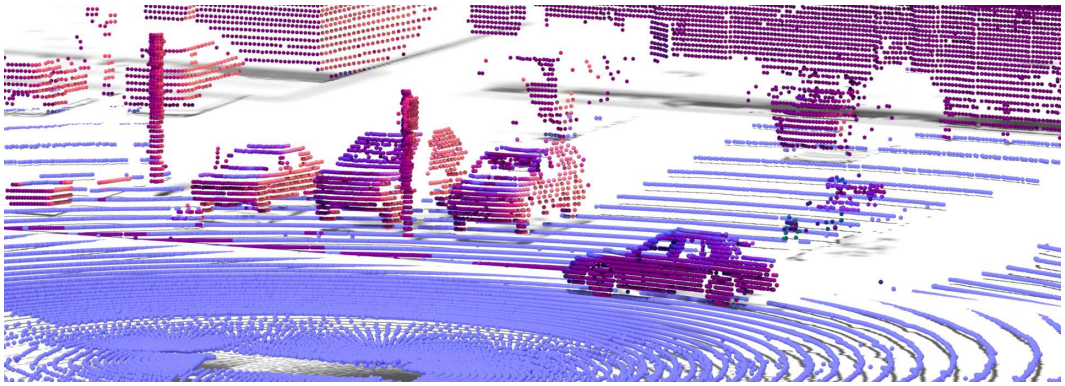
(a) SHS-Net [12] trained on PCPNet [7]



(b) SHS-Net [12] trained on our LiSu



(c) Direct transfer with our method



(d) Unsupervised domain adaptation from LiSu to Waymo with our method

Figure 5. Qualitative comparison of SHS-Net [12] directly transferred from PCPNet [7] (a) and our LiSu dataset (b). Our dataset, tailored for LiDAR surface normal estimation, yields superior results. Additionally, we demonstrate the effectiveness of our method for both direct transfer (c) and self-supervised domain adaptation (d) on a challenging Waymo Open Dataset frame.

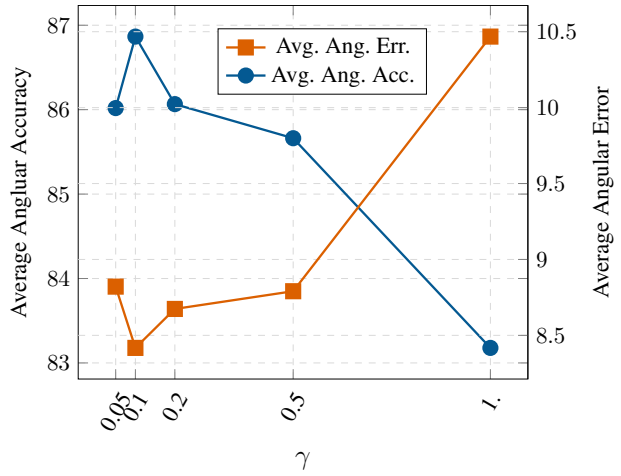


Figure 6. Average angular accuracy (↑) computed across thresholds $\{5.0°, 7.5°, 11.25°, 22.5°, 30.0°\}$ and average angular error (↓) calculated across mean, median and RMSE, for varying γ.

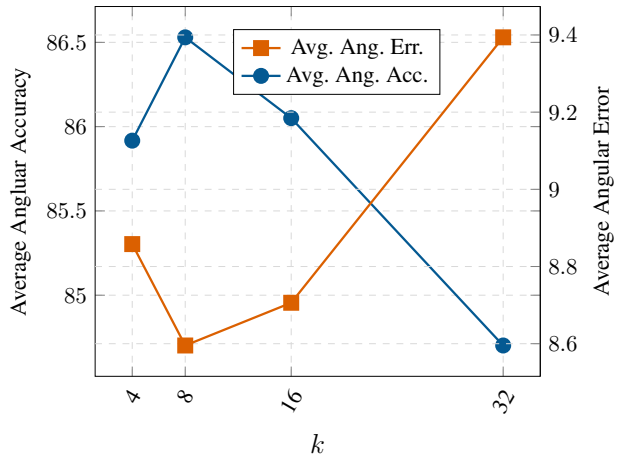


Figure 7. Average angular accuracy (↑) computed across thresholds $\{5.0°, 7.5°, 11.25°, 22.5°, 30.0°\}$ and average angular error (↓) calculated across mean, median and RMSE, for varying k.

values of k (*e.g.* 4) may not provide sufficient smoothing, resulting in noisy outputs. As Fig. 7 illustrates, $k = 8$ offers a favorable balance between smoothness and detail preservation. As a general guideline, we recommend setting k to the median number of points within a 0.1-meter radius around each point in the input data.

H. Weighted Adjacency Matrix Ablation

Edge weights in our graph $\mathcal{G}$ for both SGTV and TGTV (Sec. 3.2 of the main manuscript) are computed using an exponential decay function,

$$w(x) = \exp\left(-\frac{x^2}{\sigma^2}\right), \tag{1}$$

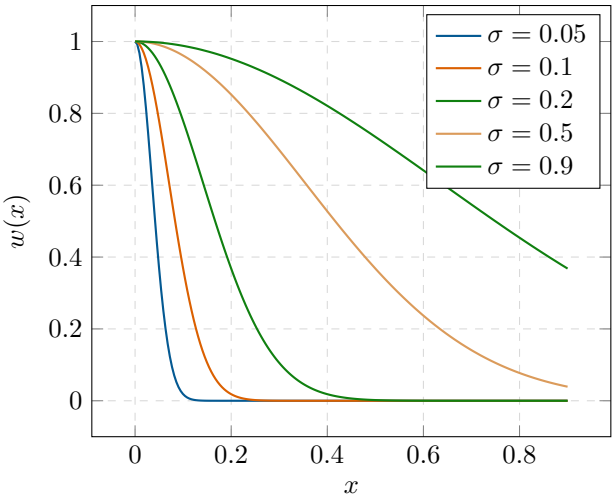


Figure 8. Effect of the decay constant σ on edge weights for different distances x in meters.

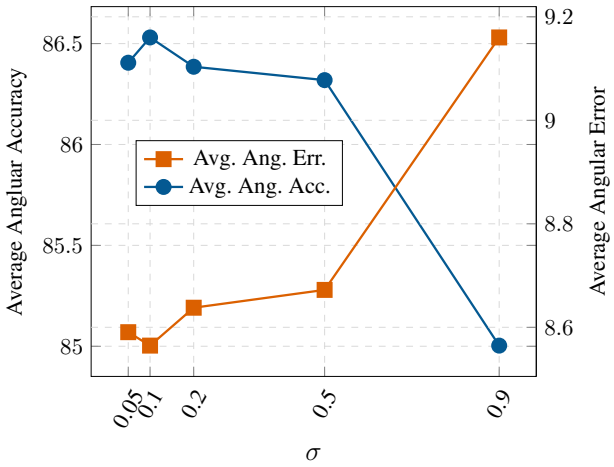


Figure 9. Average angular accuracy (↑) computed across thresholds $\{5.0°, 7.5°, 11.25°, 22.5°, 30.0°\}$ and average angular error (↓) calculated across mean, median and RMSE, for varying σ.

where x represents the Euclidean distance between two graph nodes (*i.e.* points). The decay constant σ determines the rate at which the edge weight decreases with increasing distance. We depict the influence of different σ in Fig. 8.

To study the impact of the hyperparameter σ, we conducted an ablation study with $\gamma = 0.1$ and $k = 8$, varying σ across multiple runs. For efficiency, we used 50% of the training data and 20% of the evaluation data. Results showed that smaller σ values produce sparse graphs, reducing regularization, while larger values introduce noise by connecting distant points, potentially belonging to different surfaces (*e.g.* 0.2 meters apart). Empirically, $\sigma = 0.1$ was found optimal, as shown in Figure 9.

References

[1] F. Cazals and M. Pouget. Estimating Differential Quantities Using Polynomial Fitting of Osculating Jets. *CAGD*, 22(2): 121–146, 2005.

[2] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *Proc. ICLR*, 2024.

[3] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Proc. NeurIPS*, 2022.

[4] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An Open Urban Driving Simulator. In *Proc. CoRL*, 2017.

[5] Hang Du, Xuejun Yan, Jingjing Wang, Di Xie, and Shiliang Pu. Rethinking the Approximation Error in 3D Surface Fitting for Point Cloud Normal Estimation. In *Proc. CVPR*, 2023.

[6] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets Robotics: The KITTI Dataset. *IJRR*, 32(11):1231–1237, 2013.

[7] Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J Mitra. PCPNet Learning Local Shape Properties from Raw Point Clouds. *CGF*, 37(2):75–85, 2018.

[8] Jianfei Guo, Nianchen Deng, Xinyang Li, Yeqi Bai, Botian Shi, Chiyu Wang, Chenjing Ding, Dongliang Wang, and Yikang Li. StreetSurf: Extending Multi-view Implicit Surface Reconstruction to Street Views. *arXiv CoRR*, abs/2306.04988, 2023.

[9] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Proc. SIGGRAPH*, 1992.

[10] Keqiang Li, Mingyang Zhao, Huaiyu Wu, Dong-Ming Yan, Zhen Shen, Fei-Yue Wang, and Gang Xiong. GraphFit: Learning Multi-scale Graph-Convolutional Representation for Point Cloud Normal Estimation. In *Proc. ECCV*, 2022.

[11] Qing Li, Huifang Feng, Kanle Shi, Yi Fang, Yu-Shen Liu, and Zhizhong Han. Neural Gradient Learning and Optimization for Oriented Point Normal Estimation. In *Proc. SIGGRAPH Asia*, 2023.

[12] Qing Li, Huifang Feng, Kanle Shi, Yue Gao, Yi Fang, Yu-Shen Liu, and Zhizhong Han. SHS-Net: Learning Signed Hyper Surfaces for Oriented Normal Estimation of Point Clouds. In *Proc. CVPR*, 2023.

[13] Qing Li, Huifang Feng, Kanle Shi, Yue Gao, Yi Fang, Yu-Shen Liu, and Zhizhong Han. NeuralGF: Unsupervised Point Normal Estimation by Learning Neural Gradient Function. In *Proc. NeurIPS*, 2024.

[14] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proc. CVPR*, 2017.

[15] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in Perception for Autonomous Driving: Waymo Open Dataset. In *Proc. CVPR*, 2020.

[16] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic Graph CNN for Learning on Point Clouds. *TOG*, 38(5):1–12, 2019.

[17] Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. Point Transformer V3: Simpler, Faster, Stronger. In *Proc. CVPR*, 2024.

[18] Yingrui Wu, Mingyang Zhao, Keqiang Li, Weize Quan, Tianqi Yu, Jianfeng Yang, Xiaohong Jia, and Dong-Ming Yan. CMG-Net: Robust Normal Estimation for Point Clouds via Chamfer Normal Distance and Multi-Scale Geometry. In *Proc. AAAI*, 2024.

[19] Bo Zhang, Xinyu Cai, Jiakang Yuan, Donglin Yang, Jianfei Guo, Xiangchao Yan, Renqiu Xia, Botian Shi, Min Dou, Tao Chen, Si Liu, Junchi Yan, and Yu Qiao. ReSimAD: Zero-Shot 3D Domain Transfer for Autonomous Driving with Source Reconstruction and Target Simulation. In *Proc. ICLR*, 2024.