PU-Ray: Domain-Independent Point Cloud Upsampling via Ray Marching on Neural Implicit Surface

by

Sangwon Lim

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

While recent advancements in deep-learning point cloud upsampling methods have improved the input to intelligent transportation systems, they still suffer from issues of domain dependency between synthetic and real-scanned point clouds. This thesis addresses the above issues by proposing a new ray-based upsampling approach with an arbitrary rate, where a depth prediction is made for each query ray and its corresponding patch. Our novel method simulates the sphere-tracing ray marching algorithm on the neural implicit surface defined with an unsigned distance function (UDF) to achieve more precise and stable ray-depth predictions by training a point-transformer-based network. The rule-based mid-point query sampling method generates more evenly distributed points without requiring an end-to-end model trained using a nearest-neighbour-based reconstruction loss function, which may bias towards the training dataset. Self-supervised learning becomes possible with accurate ground truths within the input point cloud. The results demonstrate the method's versatility across domains and training scenarios with limited computational resources and training data. Comprehensive analyses of synthetic and real-scanned applications provide empirical evidence for the significance of the upsampling task across the computer vision and graphics domains to real-world applications of ITS.

Preface

This thesis is a research collaboration with Dr. Karim El-Basyouny and Dr. Yee Hong Yang at the University of Alberta. The collaboration includes supervising the research and providing a part of the data used. The work is submitted to a journal and is under review for publication.

Acknowledgements

I would like to express my sincere gratitude to all those who have contributed to the completion of this project. First and foremost, I am deeply thankful to my supervisors, Dr Herbert Yang and Dr. Karim El-Basyouny, for their invaluable guidance, encouragement, and support throughout this endeavour. Their expertise and mentorship have been instrumental in shaping the direction and quality of this work.

I am also indebted to Alberta-Innovates and NSERC for providing financial support through a grant, "Innovative Methods for Road Infrastructure Digitization (ALLRP 561109 - 20)", without which this research would not have been possible. Their belief in the significance of this project has enabled its successful execution.

Furthermore, I extend my appreciation to my research colleagues for their collaboration, insightful discussions, and constructive feedback. Their diverse perspectives and collective effort have enriched this study's outcomes.

I am grateful to the staff and facilities at the Computer Graphics Lab and Centre for Smart Transportation for their assistance and resources, which have facilitated the smooth progress of this project.

Last but not least, I would like to thank my family and friends for their unwavering encouragement, understanding, and patience throughout this journey. Their unconditional support has been a constant source of motivation.

Thank you to all those mentioned above and to anyone else who has contributed in any way to this project. Your contributions have been invaluable and deeply appreciated.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.0.1 Backgrounds and Motivations

Point clouds are a data structure defined as a set of points. This has been widely utilized in remote sensing to store points of a surface captured by a Light Detection And Ranging (LiDAR) sensor in 3-dimensional (3D) coordinates. Compared to the 2D counterpart of images, point clouds provide more intuitive and accurate depth representations. The data provide an understanding of the surrounding environment's surfaces and have been used in many applications. More specifically, applications in the intelligent transportation systems (ITS) domain range from tasks directly related to vehicle controls, such as shape classification, object detection, and point cloud segmentation [13], to 3D reconstruction-related tasks of mapping and surveying [9], [21].

Point cloud upsampling aims to enhance a sparse point set by generating dense points to the implicit surface of the object. It is an essential task in 3D reconstruction for ITS, as it can lower the cost of memory storage and sensor requirements while achieving high-quality remote sensing. For example, with data enhancement, by upsampling the data from an HDL-64E sensor [47] can potentially reach a similar or better quality than that of the state-of-the-art Velodyne Alpha Prime VLS-128 sensor within the maximum operation range, which shows significant performance improvements in autonomous driving [21].

A few studies have addressed the upsampling problem in the domain of ITS [8], [22], [23], [29]. However, such enhancement solutions are restricted

to specific problems or rely on additional sensors. Studies in range image super-resolution [5], [20], [29], [45], LiDAR upsampling [44] and LiDAR completion [54] do not account for 3D densities. Concurrently, many computer vision studies [10], [14], [16], [24], [25], [31], [40]–[42], [49], [56], [60], [61] have focused on performance improvements to existing benchmarks. However, the output point distribution is not continuous when the above upsampling methods [10], [14], [16], [24], [25], [31], [40]–[42], [49], [56], [61] are applied to real-scanned LiDAR data with many local density mismatches due to the nearest-neighbour-based reconstruction loss functions, such as Chamfer Distance (CD). Also, encoding the entire object shape [24], [40], [56], [60], [61] may create domain dependency when encountering an unseen object. Such end-to-end behaviours could pose issues for 3D reconstruction for ITS because real-scanned data include many different objects, and defining the bounding space of the infrastructural environment is difficult. At the same time, the upsampled points generated near the known input points are less important for 3D reconstruction. Additionally, many methods [24], [25], [31], [40]–[42], [56] upsample point clouds with a fixed scaling rate, r. Thus, a different model has to be trained to upsample with a different scaling rate. Combined with the issues of end-to-end strategies, the fixed rate causes inflexibility without the freedom of ROI definition and output density. The observations above motivate the proposed method to move away from end-to-end strategies with CD-based loss functions and a fixed upsampling rate.

1.0.2 Research Goals and Scope

The main goal of this thesis work is to upsample point clouds independent of point cloud domains to which the method is applied. In other words, the proposed method should be able to train in one domain (e.g. synthetic data) and transferred to another (e.g. real-scanned data) at the inference time. Additionally, we aim to upsample with an arbitrary scaling rate, r, to achieve flexibility. Our approach to accomplishing the goals is by generating rays directed towards the regions of interest (ROIs) and predicting the depth of the point where the ray and the implicit surface intersect. We define the

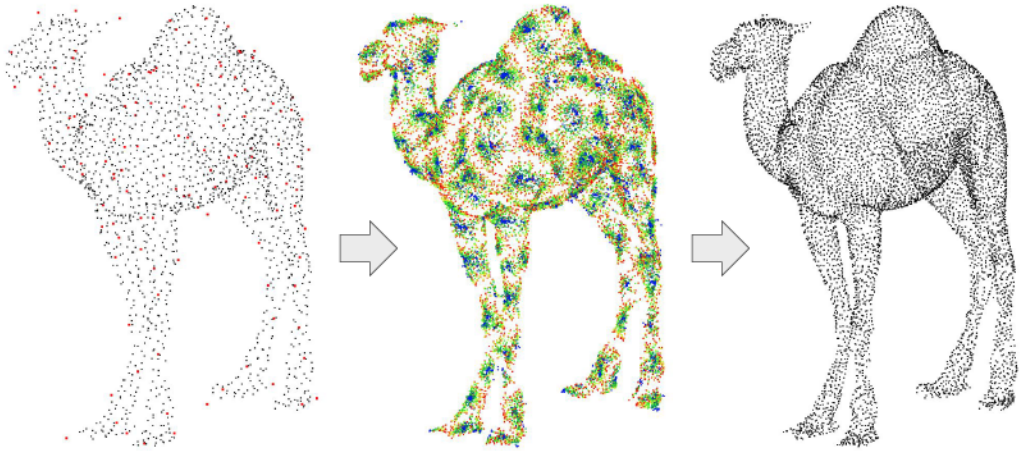

Figure 1.1: Given the sparse input (left), through the ray marching algorithm (middle), the final dense output (right) is achieved. Points on the implicit surface are coloured to represent marching steps from the earlier (blue) to the later (red) steps. The red dots on the left are the initial query ray origins.

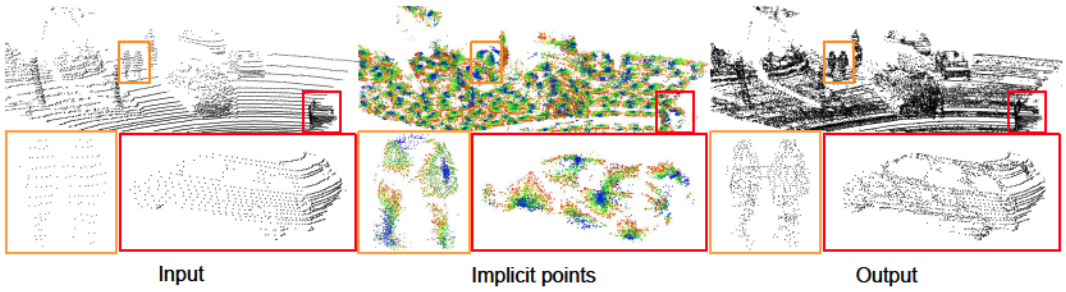

Figure 1.2: Demonstration of PU-Ray through the input, implicit points, and output on a KITTI-360 [26] snippet scene. The zoomed-in boxes show regions with pedestrians (orange) and a car (yellow).

implicit surface of a point cloud patch with a neural network, and the precise depth prediction is possible using a ray marching algorithm. In sum, we aim to upsample in a non-end-to-end manner, which causes domain dependency. The scope of the research is bounded by upsampling point clouds in the ROIs, where there are observed points. Completion in the unseen or occluded regions and object-based point cloud completions [9] are not in the scope.

1.0.3 Contributions

This thesis addresses the listed issues with ray-depth prediction via sphere tracing [15] on the neural implicit surface, defined using a point-transformer-based [59] network, for point cloud upsampling. Our method requires a 3D implicit representation of a patch, and it cannot be applied directly to point clouds, particularly with missing information between points. To overcome this constraint, we propose a point-transformer-based ray marching module that learns the neural implicit surface of a point cloud patch. This module is iteratively called at each ray marching step to enhance the final depth prediction. Ablation studies show performance gains using the proposed ray marching method. Points on the implicit surface resulting from ray marching steps also provide a visual understanding of the implicit surface of a point cloud (Figures 1.1 and 1.2). Although similar ideas have been proposed by Liu et al. [30], Chibane et al. [6], and Venkatesh et al. [48] after the deep signed distance function (DeepSDF) [35] was first introduced, the implicit surfaces may have domain dependency due to the shape code corresponding to an object, and the training is restricted to supervised learning requiring ground truth distance from an arbitrary point to the surface. They also use sphere tracing [15] exclusively for rendering instead of point cloud upsampling.

Applying the proposed ray-based upsampling method has several advantages over existing end-to-end models. First, the implicit surface of a patch is domain-independent compared to object-based representations that may be biased towards the dataset's context. The implicit surface also efficiently compresses large 3D data into a small function with a few parameters. Moreover, the upsampling rate is determined by the number of rays generated, enabling upsampling with different rates. The direction of a query ray can be generated in a rule-based system to focus on the ROIs. Lastly, unlike existing methods, the training objective and ground truth are clearly stated. Inspired by LiDAR range-image super-resolution [20], given a neighbourhood point set, the model outputs the depth prediction of a query ray. This allows the retrieval of ground truth in limited situations because the ray can be determined from an existing

point's direction and depth from the query ray origin. Thus, such a method makes self-supervised learning possible. Also, the more straightforward task reduces the model size. Our contributions are as follows:

1. Our novel method adopts the sphere tracing [15] on the neural implicit surface of point clouds, yielding precise depth predictions while allowing supervised and self-supervised training.

2. The domain-independent ray-based upsampling is versatile across synthetic to real-scanned and from object-level to infrastructure-level point clouds. The proposed model is the smallest among all compared deep-learning point cloud upsampling methods while showing competitive performances.

3. The experimental results on novel datasets consisting of low-resolution LiDAR inputs, super-resolution point clouds and 3D reconstruction ground truths in urban and highway environments provide empirical support for the potential of upsampling real-scanned point clouds.

1.0.4 Thesis Outline

The following chapters in this thesis consist of Literature Review, Methodology, Experiments and Conclusion. The Literature Review includes a comprehensive survey of implicit surface representation, point cloud processing, and point cloud upsampling. For the literature review, the first section covers the definition of implicit surfaces, their advantages and the recent advancements in neural implicit surfaces. Reviews on point cloud processing focus on the physical and computational challenges with point cloud data and how deep learning methods have been adapted to overcome the issues. A series of point cloud upsampling methods are discussed based on characteristic classifications. Brief introductions to related methods, such as LiDAR super-resolution and point cloud completion, are also discussed. In the Methodology chapter, the problem statement explicitly defines the objective in the mathematical formulation. Network architecture and its associating loss functions for training

neural-implicit representations are thoroughly described. Lastly, adaptations of the upsampling strategy for synthetic and real-scanned data are also proposed. In the Conclusion chapter, we discuss the overall evaluation and limitations of our method along with potential future directions that could be inspired by this study.

Chapter 2

Literature Review

2.1 Implicit Surface Representation

2.1.1 Implicit Surface and Ray Marching

In 3D computer graphics, discrete polygonization with vertices and faces is one way to store 3D geometries [3], [34]. However, this introduces a discrepancy in representing continuous real-world objects. On the other hand, an implicit representation of a surface (i.e. $f(x, y, z) = 0$, where x, y and z are independent variables of 3D coordinates) provides a continuous representation of the surfaces in 3D space. For example, an implicit representation of a sphere is $x^2 + y^2 + z^2 - R^2 = 0$, where the only parameters are 3D coordinates (x, y, z) and the radius R. On the other hand, 1000 triangles are required to approximate a "reasonable" sphere while discontinuity remains [4]. Therefore, ray tracing is proposed for rendering from an implicit surface to mitigate memory redundancy directly [4], which simulates how a light ray would interact with the environment.

Hart [15] proposed sphere tracing to achieve a precise depth estimation along a ray to its intersection with a surface. The method utilizes the continuous signed distance function (SDF), which outputs the distance to the nearest point on the surface given a location in the 3D space. The algorithm performs multiple iterations to advance (or *march*) along the ray direction the distance to the nearest point (intersection) on the surface, defined as the marching step. This guarantees the marching point to approach infinitesimally to the

intersecting point of the ray and the surface and prevents it from penetrating. The process halts when the marching step converges to a satisfying level. The sum of all marching steps, $\tilde{t}$, is mapped to a ray function, $\mathbf{r}(t)$, given the ray origin, $\mathbf{o}$ and direction vector, $\mathbf{d}$, to get the intersection point:

$$\mathbf{r}(\tilde{t}) = \mathbf{o} + \tilde{t} \cdot \mathbf{d}. \tag{2.2}$$

Ray marching (e.g. sphere tracing [15]) has maintained its popularity as one of the implicit surface rendering techniques thanks to its simplicity, efficiency and accuracy. Such a method is flexible because of its applicability on complex surfaces defined with implicit surfaces. In contrast, traditional ray intersection tests, such as Bounding Volume Hierarchy [7] of ray tracing [51], explicitly parametrize each shape defined in a discrete space. It is easier to find intersecting points by plugging in the ray parametric equation to get the closed-form solution of $\tilde{t}$, but the explicit parametrization is limited to simpler geometry and makes it difficult to represent complex shapes accurately. The ray marching method can also be optimized to enhance the efficiency and GPU utilization [30] by removing redundant marching steps. We accomplish this in our study by patch-based ray marching and ϵ estimation, which will be discussed in the next chapter. Furthermore, ray marching yields more precise and accurate volume rendering as the algorithm prevents the ray from penetrating the surface [15]. In summary, the combination of implicit surface and ray marching is advantageous for practical applications compared to other rendering methods, such as ray tracing [51].

2.1.2 Neural Implicit Surface

Deep neural networks (DNNs) became popular in the 2010s when the computation capabilities reached the level of fruition for implementing the concept. DNNs have multiple layers of neurons consisting of a weight and a bias for linear regression. This combination of multiple linear functions results in a function that outputs a non-linear decision boundary, which suits the definition of implicit surface. In other words, a neural network is a distance function, such as SDF, in the context of implicit surfaces; we name it a neural implicit

surface. The DeepSDF [35] is a pioneering work in the field of the neural implicit surface by representing continuous implicit surfaces with a DNN, $f_\theta(\cdot)$, that outputs the signed distance, s, of an arbitrary sample point, x, with respect to the shape defined with a latent code, z:

$$s = f_\theta(z, x) \approx SDF(x). \tag{2.1}$$

The method demonstrates the potential of neural implicit surfaces in the compressed representation of complex shapes. However, the model requires the ground truths signed distance of arbitrary sample points, and the inference time is slow due to optimizing the random shape code vector. The work inspired Liu et al. [30], Chibane et al. [6], and Venkatesh et al. [48] to utilize DeepSDF [35] or similarly constructed SDF or unsigned distance function (UDF) with sphere tracing [15] for 2D view rendering. The aforementioned methods may fit towards the known objects as the shape encoding learns the overall structure of the input, causing domain dependency and the inability of self-supervised learning of the neural implicit surfaces. Thanks to the capabilities of neural implicit surfaces in compressing complex shapes, the idea has also been adopted in point cloud upsampling [10], [16], [41], [60], which is discussed in 2.3.1.

2.2 Point Cloud Processing

2.2.1 Properties of Point Clouds

Despite the advantages of point clouds in numerous applications, several attributes hinder their utilization in computer vision. Point clouds inherit the main property, permutation-invariance, of the superset data structure, *set*. Unlike grid pixels in 2D images, points in a point cloud are unordered, and the set size can vary, making computation difficult in discrete and ordered manners. Also, a set of points is not a continuous surface representation. Therefore, computer graphics techniques cannot directly be applied to the data without generating the surface information, such as polygonization or implicit parameterization [4]. Lastly, scanning a 3D object requires multiple

captures from different observation locations. LiDAR sensors cannot detect occluding surfaces, and a single scan only includes the visible points, which can be represented as a range or a depth map, a 2D grid representation of depths from the sensor to the target environment. Therefore, LiDAR point clouds are often referred to as 2.5D data [55].

2.2.2 Deep Neural Networks for Point Clouds

One of the first and most intuitive approaches to extracting point cloud features using deep learning is 3D convolutional neural networks (CNNs) [33], [38], [53]. Voxelization is required to apply CNNs to point clouds, which generalizes the data with occupancy or probabilistic distribution mapping to assign values to grid cells. Such practice causes the tradeoff between the information loss from using low-resolution voxel grids and computation cost due to too many empty cells in the opposite case.

The problem of applying the orderliness nature of CNN on permutation-invariant and unordered data has been addressed in PointNet [37], and it became a pioneering work in point cloud processing for DNN adaptation by inputting point clouds directly to the network, without any pre-processing. The features are extracted independently for each point, and the max-pooling of all point features yields global features. PointNet++ [39] takes a step further to capture local context by aggregating information in small point groups defined with k-Nearest-Neighbours (KNN) or ball querying operations that guarantee the number of points in a group or the fixed size of the local neighbourhood, respectively. PointNet [37] and PointNet++ [39] inspired many studies to adopt their methods in point cloud upsampling, such as PU-Net [56], MPU [49] and PU-GAN [24]. However, the independent feature extraction on each point neglects the geometric relationships in the local neighbourhood and global point cloud.

Dynamic Graph CNN (DGCNN) [50] constructs graphs of local neighbourhoods by connecting a center point x_i to its neighbouring points x_j, and EdgeConv aggregates the features given the relative positions $x_j - x_i$ in addition to information given by x_i. EdgeConv operation shows significant performance

gain in learning the geometric relationships for point cloud classification and segmentation tasks. PU-GCN [40] adopts the EdgeConv to enhance the point cloud upsampling task.

The transformer architecture has been extensively applied to other fields since the first method was proposed, showing impressive performance in natural language processing [46]. In particular, the transformer has been employed in unorganized point cloud data with great potential, as shown in recent work [11], [12], [59]. The main idea involves calculating point attention within a small patch of neighbouring points to extract the relationships between sampled points. The practice is advantageous because the 3D coordinates of points naturally convey positional information without needing the positional encoding step. Point cloud attention calculation has two major variants. Guo et al. [12] employ the traditional transformer's scalar attention using dot products.

$$y_i = \sum_{x_j \in X(i)} \rho(\phi(x_i)^T \cdot \psi(x_j)) \cdot \alpha(x_j), \tag{2.3}$$

where y_i is the transformed features of x_i, X_i is the local neighbourhood of x_i, $\phi(\cdot)$, $\psi(\cdot)$ and $\alpha(\cdot)$ are the MLPs for the query, key and value, and $\rho(\cdot)$ is the softmax function. The authors also claim that positional encoding is not required as the 3D input coordinates intrinsically have such information, unlike the original transformer. In contrast, Zhao et al. [59] preserve the vector representation of attention through element-wise subtraction (Figure 2.1):

$$y_i = \sum_{x_j \in X(i)} \rho(\gamma(\phi(x_i) - \psi(x_j) + \theta(x_i - x_j))) \odot (\alpha(x_j) + \theta(x_i - x_j)), \tag{2.4}$$

where $\theta(x_i - x_j)$ is the relative positional encoding and $\gamma(\cdot)$ is an MLP. The study has empirically demonstrated the superiority of vector representation of self-attention and relative positional encoding in downstream tasks, such as point cloud segmentation. Applying self-attention was first introduced to point cloud upsampling by Li et al. [24]. One of the latest advancements in point cloud upsampling, with state-of-the-art performance, also relies on the transformer architecture [42].

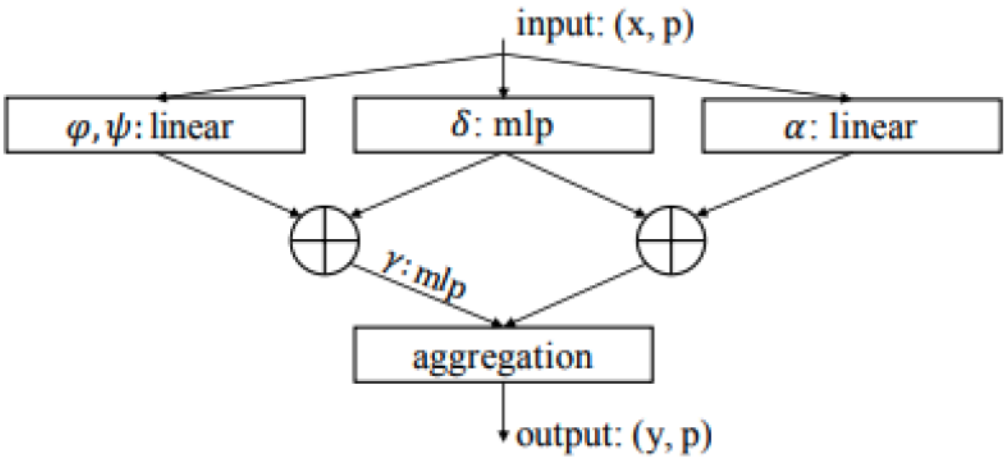

Figure 2. Point transformer layer.

Figure 2.1: Point transformer layer proposed by Zhao et al. [59].

2.3 Point Cloud Upsampling

2.3.1 Deep-learning Methods

Before the advent of the deep-learning point cloud upsampling methods, edge-aware resampling (EAR) [17] held state-of-the-art results by projecting points to a latent surface. However, the method requires carefully tuned neighbourhood radius and angle parameters, which vary for different point clouds or regions in a point cloud depending on the point density for accurate surface normal estimation. Thus, its application to multiple point clouds or real-scanned data is difficult, if not impossible, to automate. This inspired the first deep-learning method, PU-Net [56], adopting the feature encoding method of PointNet++ [39]. Many point cloud upsampling methods have a fixed upsampling rate and use end-to-end training by encoding the object shape, upsampling points with feature expansion (e.g. $N \times C \rightarrow N \times rC$, where N is the number of points, C is the channel size and r is the upsampling rate) and tensor reshaping (e.g. $N \times rC \rightarrow rN \times C$) to fit the ground truth upsampled point clouds [14], [24], [25], [31], [40]–[42], [49], [56], [61] (Figure 2.2). How-

ever, domain-dependent and object-based upsampling with a fixed rate causes inflexibility, which motivates some studies to address the issues.

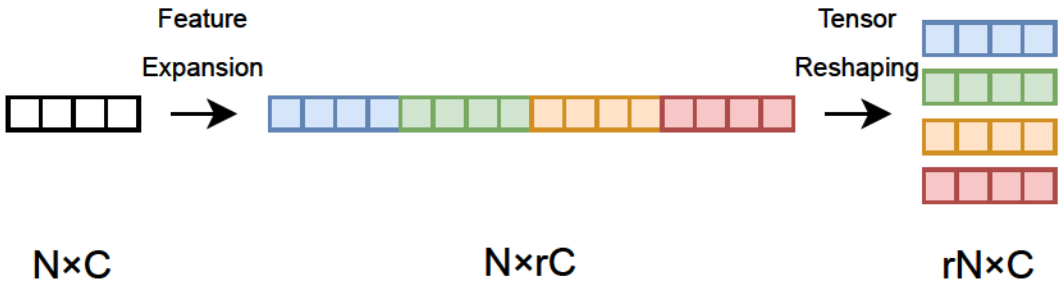

Figure 2.2: Visual illustration of feature expansion and tensor reshaping operations for point cloud upsampling.

Methods with arbitrary upsampling rates generally have seed points, the initial samples before refinement using an implicit surface [10], [16], [41], [60]. PUGeo-Net [41] upsamples points in a 2D domain, corresponding to the tangent plane of a 3D patch, and the points are then projected to the implicit surface with distances and normals calculated using a neural network. SAPCU [60] voxelizes the 3D space uniformly and uses the center points of voxels as seeds. Another drawback is the high computational cost of generating seed points. Indeed, the farthest point sampling could be applied to improve performance. Neural Points [10] represents an implicit surface using a bijective mapping function between points on 3D surfaces and corresponding points on a 2D discrete grid. Therefore, increasing seed points on the 2D points naturally upsamples the 3D point cloud. However, one common potential problem of the above methods is that the uniformity of seed points is not maintained when 2D points are projected to the 3D surface. Grad-PU [16] adopts the midpoint algorithm in local neighbourhoods and achieves state-of-the-art performance by projecting the points onto the implicit surface by gradient descent optimization. However, their implicit surface does not show continuity (Figure 2.3), which could be a problem, in particular, when the input point cloud distribution is not uniform.

Unlike 2D image super-resolution, upsampling an unordered and unstruc-

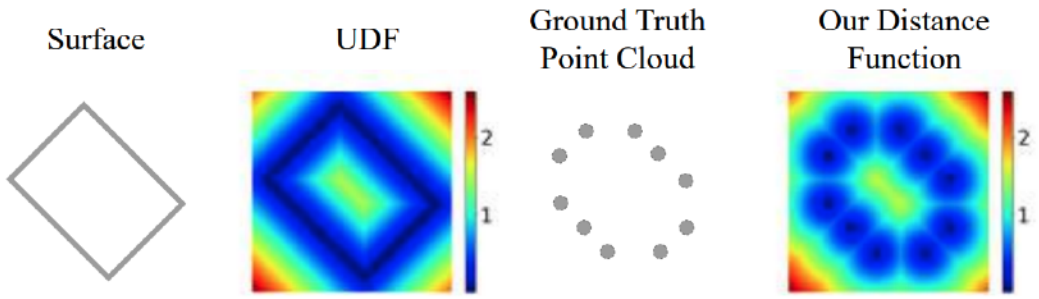

Figure 2.3: Ground truth UDF versus the implicit surface of Grad-PU [16].

tured point cloud does not have a fixed ground truth for the corresponding low-density input because the upsampled points do not have specified locations in the 3D space. Therefore, many methods rely on nearest-neighbour-based reconstruction loss functions, e.g. Chamfer distance (CD) and Hausdorff distance (HD) [2], [49], [58] or their modifications:

$$d_{CD}(S_1, S_2) = \frac{1}{|S_1|} \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \frac{1}{|S_2|} \sum_{y \in S_2} \min_{x \in S_1} \|y - x\|_2^2 , \tag{2.5}$$

$$d_{HD}(S_1, S_2) = \max \left[\sup_{x \in S_1} \inf_{y \in S_2} d(x, y), \sup_{y \in S_2} \inf_{x \in S_1} d(x, y) \right] ,$$

where $d_{CD}(S_1, S_2)$ and $d_{HD}(S_1, S_2)$ are the Chamfer and Hausdorff distances between point sets S_1 and S_2. sup and inf are the supremum and infimum operations [58]. Combined with end-to-end learning, the upsampling performance relies on the input point distribution of training datasets with uniformly sampled point clouds from 3D surfaces. Hence, many upsampling models, which are trained on synthetic datasets, have the domain dependency issue. While the Chamfer distance is a great evaluation metric that is sensitive to outliers, Wu et al. [52], Lin et al. [27] and Lin et al. [28] argue the ambiguity of the CD loss function for training a point generation model because it ignores all other points but the nearest point in the target point cloud and every nearest-neighbour pair is treated with equal weights. In ideal situations, more weight should be given to sparser points so that the model does not *cheat* by avoiding point generation in sparse areas than in dense areas [27], [28], [52]. Such sensitivity to outliers often causes clustering behaviour when a general encoder-decoder point cloud generation model is trained with supervised

learning. Moreover, point matching rules of Chamfer losses are static, and the optimization is likely to be stuck in local minimums [18], which also aggravates the domain dependency issue.

2.3.2 Self-supervised Learning

Several studies have demonstrated the potential of self-supervised learning for point cloud upsampling [31], [60], [61]. The first two methods, SSPU and SPU-Net [31], [61], follow a similar point generation method with earlier models using feature expansion and tensor reshaping. On the other hand, SAPCU [60] allows for an arbitrary upsampling rate with seed point generation. Although they have a similar problem statement to our study, the main difference is that they must calculate the pseudo-ground-truth for point projection on the implicit surface. In contrast, our method has accurate ground truths based on existing points in the input point cloud.

2.3.3 Applications to LiDAR Scans

Datasets for point cloud upsampling are less standardized for applications using LiDAR scans than synthetic datasets used in conventional computer vision studies. For example, Li et al. [23] propose a problem-specific interpolation-based point cloud density enhancement method for long-range pedestrian detection with a sparse point cloud [23]. Camera-LiDAR fusion methods [8], [22], [29] are suggested to obtain denser outputs. As an alternative direction, super-resolution on a 2D range image or depth map has been proposed [5], [20], [29], [45]. Although the coarse-to-fine objective of super-resolution is similar to that of point cloud upsampling, the resulting point distribution from super-resolution may not be uniform in the 3D space, as the task focuses on obtaining a high resolution in the 2D space. Point cloud upsampling generates a denser point cloud but does not translate to a high-resolution point cloud. On the other hand, LiDAR Upsampling by Savkin et al. [44] and LiDAR Completion by Xiong et al. [54] have attempted to generate an output point cloud directly from the input point cloud without an intermediate 2D representation. However, the objectives of both methods are similar to super-resolution methods,

as the ground truths are high-resolution LiDAR scans due to the difficulty in collecting dense point clouds. Our proposed method solves this issue with ray-based upsampling.

Point cloud completion [9] is a closely related task of upsampling by completing the surface shape by generating points. The comprehensive review by Fei et al. [9] on point cloud completion emphasizes the importance of the 3D reconstruction task in the domain of ITS by lowering storage cost and sensor requirements reduction. Most existing methods are primarily object-based completion methods and cannot be applied to real scenes, which include many different objects, and defining the bounding space of the environment for shape completion is difficult. Similar emphases are found in other previous real scene 3D reconstruction studies for ITS [5], [8], [20], [22], [23], [29], [44], [45], [54].

Chapter 3

Methodology

Our proposed method, PU-Ray, performs the sphere tracing algorithm [15] on the neural implicit surface defined with unsigned distance function (UDF). With our novel loss functions, PU-Ray effectively trains a precise depth prediction model through sphere-tracing [15] while using a small number of parameters. The trained UDF for precise depth prediction is combined with our query ray generation algorithm for upsampling. In the following subsections, the problem statement explicitly explains how the implicit surface is defined. The query ray generation algorithms and the network architecture are discussed.

3.1 Problem statement

Following the original sphere tracing paper [15], we define our ray mapping function, $\mathbf{q} : \mathbb{R} \rightarrow \mathbb{R}^3$, as:

$$\mathbf{q}(t) = \mathbf{o}^{\mathbf{q}} + t \cdot \mathbf{d}^{\mathbf{q}}, \tag{3.1}$$

where $\mathbf{o}^{\mathbf{q}} \in \mathbb{R}^3$ is the origin and $\mathbf{d}^{\mathbf{q}} \in \mathbb{R}^3$ is the unit vector that defines the direction. The problem formulation of the method is similar to the work of Kwon et al. [20]:

$$U = S \cup \{\mathbf{q}(\hat{t}^{\mathbf{q}}) \mid \forall \mathbf{q} \in Q,\ \hat{t}^{\mathbf{q}} = f(P^{\mathbf{q}}, \mathbf{d}^{\mathbf{q}})\}, \tag{3.2}$$

where $S \subset \mathbb{R}^3$ is the sparse point cloud, $U \subset \mathbb{R}^3$ the upsampled point cloud, $Q = \{\mathbf{q} \mid \mathbf{o}^{\mathbf{q}}, \mathbf{d}^{\mathbf{q}} \in \mathbb{R}^3\}$ the set of query rays, $P^{\mathbf{q}} \in \mathbb{R}^{3\times k}$ the patch that

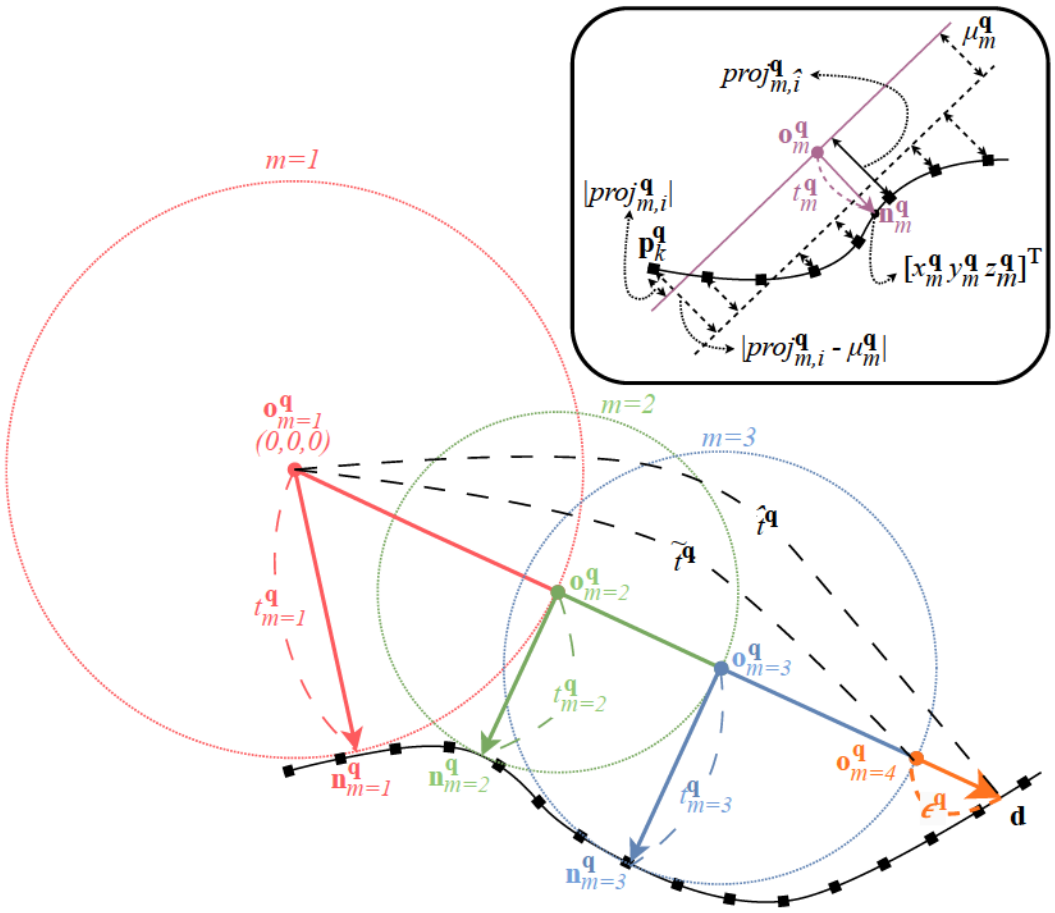

Figure 3.1: Visual demonstration of the ray marching method on the implicit surface. The origin, $\mathbf{o}_m^{\mathbf{q}}$, is updated at every marching step, m, given the nearest distance to the implicit surface, $t_m^{\mathbf{q}}$. The final query ray depth, $\hat{t}^{\mathbf{q}}$, is the sum of the cumulative depth, $\tilde{t}^{\mathbf{q}}$, and an offset, $\epsilon^{\mathbf{q}}$. The inset at the top-right is a visual aid to the nearest point search of $(x_m^{\mathbf{q}}, y_m^{\mathbf{q}}, z_m^{\mathbf{q}})$ on the implicit surface. The purple plane is defined with $\mathbf{o}_m^{\mathbf{q}}$ and its implicit nearest direction, $\mathbf{n}_m^{\mathbf{q}}$. The projection distances to the approximated tangent plane (dotted straight line) are defined by $|proj_{m,i}^{\mathbf{q}} - \mu_m^{\mathbf{q}}|$'s. The nearest implicit distance, $t_m^{\mathbf{q}}$, is approximated by the projection distance of the nearest point in the patch defined by $proj_{m,\hat{i}}^{\mathbf{q}}$.

corresponds to ray $\mathbf{q}$, and $\hat{t}^{\mathbf{q}} \in \mathbb{R}^1$ the predicted depth that corresponds to ray $\mathbf{q}$. In our method, the number of query rays is set to $|Q| = |S| \cdot (r - 1)$, where r is the upsampling rate. The ray depth prediction network $f(\cdot)$ takes in $\mathbf{d}^{\mathbf{q}}$ and $P^{\mathbf{q}}$, where the k-NN patch $P^{\mathbf{q}} = [\mathbf{p}_1^{\mathbf{q}} \dots \mathbf{p}_i^{\mathbf{q}} \dots \mathbf{p}_k^{\mathbf{q}}]$ is sampled from S, as described in Sect. 3.2, and translated for relative positioning so

that $\mathbf{o}^{\mathbf{q}}$ is centred at $\mathbf{o}^{\mathbf{q}}_{m=1} = (0, 0, 0)$, where $m = 1$ means the marching step index initialized to 1 (Alg. 1). In other words, $\{\mathbf{p}^{\mathbf{q}}_i + \mathbf{o}^{\mathbf{q}}\}^k_{i=1} \subset S$. Relative positioning introduces simpler loss functions and prevents overfitting by ignoring the ray origin information during training. The depth of a ray, $\hat{t}^{\mathbf{q}}$, indicates the distance from the query ray origin to the crossing point to the implicit surface (Fig. 3.1).

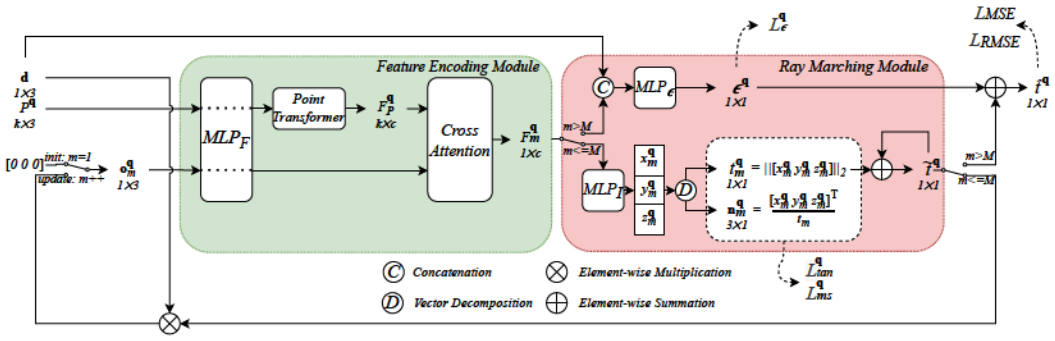

Figure 3.2: Network overview of PU-Ray with a single query input defined with $\mathbf{d}^{\mathbf{q}}$ and $P^{\mathbf{q}}$.

3.1.1 Neural implicit surface definition with MLP_I

We define our implicit surface similar to that in SAPCU [60] with a UDF since a signed distance function is impossible to define in a patch-based method. A multi-layer perceptron (MLP), MLP_I (Alg. 1; Fig. 3.2), following the feature encoding of $F^{\mathbf{q}}_m$, outputs the nearest point, $[x^{\mathbf{q}}_m\ y^{\mathbf{q}}_m\ z^{\mathbf{q}}_m]^T$, from the implicit surface to an arbitrary $\mathbf{o}^{\mathbf{q}}_m$. The implicit nearest point (middle of Figs. 1.1 and 1.2) is decomposed into meaningful information about the distance, $t^{\mathbf{q}}_m = \|[x^{\mathbf{q}}_m\ y^{\mathbf{q}}_m\ z^{\mathbf{q}}_m]\|_2$, and direction, $\mathbf{n}^{\mathbf{q}}_m = [x^{\mathbf{q}}_m\ y^{\mathbf{q}}_m\ z^{\mathbf{q}}_m]^T / t^{\mathbf{q}}_m$, from $\mathbf{o}^{\mathbf{q}}_m$ to $(x^{\mathbf{q}}_m, y^{\mathbf{q}}_m, z^{\mathbf{q}}_m)$.

Firstly, a unit vector, $\mathbf{n}^{\mathbf{q}}_m$, combined with $\mathbf{o}^{\mathbf{q}}_m$ defines a plane (The purple line in the inset of Fig. 3.1), where each $\mathbf{p}^{\mathbf{q}}_i$ can be projected to with distance denoted as $proj^{\mathbf{q}}_{m,i}$:

$$S_{cos}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|\ \|\mathbf{b}\|}, \tag{3.3}$$

$$proj^{\mathbf{q}}_{m,i} = S_{cos}\left(\mathbf{n}^{\mathbf{q}}_m, \frac{\mathbf{o}^{\mathbf{q}}_m \mathbf{p}^{\mathbf{q}}_i}{\|\mathbf{o}^{\mathbf{q}}_m\ \mathbf{p}^{\mathbf{q}}_i\|_2}\right) \cdot \|\mathbf{o}^{\mathbf{q}}_m \mathbf{p}^{\mathbf{q}}_i\|_2 ,$$

Algorithm 1 Ray marching on point cloud

Require: Query ray direction, $\mathbf{d}^{\mathbf{q}}$ ▷ Sect. 3.1 & 3.2
Require: Patch, $P^{\mathbf{q}}$ ▷ Sect. 3.1 & 3.2
Require: $MLP_F(\cdot)$, $MLP_I(\cdot)$ and $MLP_\epsilon^{\mathbf{q}}(\cdot)$ ▷ Sect. 3.3
Require: Point Transformer Module, $PT(\cdot)$ ▷ Sect. 3.3.1
Require: Cross Attention Module, $CA(\cdot)$ ▷ Sect. 3.3.1
Ensure: Maximum number of marching steps, $M \geq 0$

$m \leftarrow 1$ ▷ Initialize marching step index
$\mathbf{o}_m^{\mathbf{q}} \leftarrow [0\,0\,0]^T$ ▷ Initialize origin
$\tilde{t}^{\mathbf{q}} \leftarrow 0$ ▷ Initialize cumulative depth
$F_P^{\mathbf{q}} \leftarrow PT(MLP_F(P^{\mathbf{q}}))$ ▷ Patch Encoding
while $m \leq M$ **do**
  $F_m^{\mathbf{q}} \leftarrow CA(MLP_F(\mathbf{o}_m^{\mathbf{q}}), F_P^{\mathbf{q}})$ ▷ Cross Attention at $\mathbf{o}_m^{\mathbf{q}}$
  $[x_m^{\mathbf{q}}\, y_m^{\mathbf{q}}\, z_m^{\mathbf{q}}]^T \leftarrow MLP_I(F_m^{\mathbf{q}})$ ▷ Implicit nearest point
  $t_m^{\mathbf{q}} \leftarrow \|[x_m^{\mathbf{q}}\, y_m^{\mathbf{q}}\, z_m^{\mathbf{q}}]\|_2$ ▷ Implicit nearest distance
  $\mathbf{n}_m^{\mathbf{q}} \leftarrow [x_m^{\mathbf{q}}\, y_m^{\mathbf{q}}\, z_m^{\mathbf{q}}]^T / t_m^{\mathbf{q}}$ ▷ Implicit nearest direction
  $\tilde{t}^{\mathbf{q}} = \tilde{t}^{\mathbf{q}} + t_m^{\mathbf{q}}$ ▷ Update cumulative depth
  $m = m + 1$ ▷ Increment marching step index
  $\mathbf{o}_m^{\mathbf{q}} \leftarrow \mathbf{d}^{\mathbf{q}} \cdot \tilde{t}^{\mathbf{q}}$ ▷ Update origin
end while
$F_m^{\mathbf{q}} \leftarrow CA(MLP_F(\mathbf{o}_m^{\mathbf{q}}), F_P^{\mathbf{q}})$ ▷ Cross Attention at $\mathbf{o}_m^{\mathbf{q}}$
$\epsilon^{\mathbf{q}} \leftarrow MLP_\epsilon(concatenate(F_m^{\mathbf{q}}, \mathbf{d}^{\mathbf{q}}))$ ▷ Epsilon estimation
return $\hat{t}^{\mathbf{q}} \leftarrow \tilde{t}^{\mathbf{q}} + \epsilon^{\mathbf{q}}$

where $S_{cos}(\mathbf{a}, \mathbf{b})$ is the cosine similarity between two unit vectors. The cosine similarity between $\mathbf{n}_m^{\mathbf{q}}$ and the unit vector of $\mathbf{o}_m^{\mathbf{q}}\,\mathbf{p}_i^{\mathbf{q}}$ is multiplied by the magnitude between the two points, $\|\mathbf{o}_m^{\mathbf{q}}\,\mathbf{p}_i^{\mathbf{q}}\|_2$, to obtain the projection distance, $proj_{m,i}^{\mathbf{q}}$, defined above (Inset of Fig. 3.1). In the optimal situation, the plane defined with $\mathbf{n}_m^{\mathbf{q}}$ and $\mathbf{o}_m^{\mathbf{q}}$ would be parallel to the tangent plane that touches the object surface at $(x_m^{\mathbf{q}}, y_m^{\mathbf{q}}, z_m^{\mathbf{q}})$, which makes $\mathbf{n}_m^{\mathbf{q}}$ equivalent to the surface normal, namely, the implicit nearest direction. We approximate the tangent plane using the following loss function:

$$\mu_m^{\mathbf{q}} = \sum_{i=1}^{k} proj_{m,i}^{\mathbf{q}}/k, \tag{3.4}$$

$$\omega_{m,i}^{\mathbf{q}} = \exp\left(-\frac{\|\mathbf{o}_m^{\mathbf{q}}\,\mathbf{p}_i^{\mathbf{q}}\|_2^2}{2 \cdot avg(\{\|\mathbf{o}_m^{\mathbf{q}}\,\mathbf{p}_j^{\mathbf{q}}\|_2^2 \mid \forall \mathbf{p}_j^{\mathbf{q}} \in P^{\mathbf{q}}\})}\right),$$

$$L_{tan}^{\mathbf{q}} = \frac{1}{M} \cdot \sum_{m=1}^{M} \sqrt{\frac{\sum_{i=1}^{k}[(proj_{m,i}^{\mathbf{q}} - |\mu_m^{\mathbf{q}}|) \cdot \omega_{m,i}^{\mathbf{q}}]^2}{\sum_{i=1}^{k} \omega_{m,i}^{\mathbf{q}}}},$$

where $avg(\cdot)$ is the mean value of a set, and M is the maximum number of marching steps. Let $\widehat{proj}_m^{\mathbf{q}}$ denote the projection distance between this proxy tangent plane and the plane defined with $\mathbf{n}_m^{\mathbf{q}}$ and $\mathbf{o}_m^{\mathbf{q}}$, which are parallel to each other (Inset of Fig. 3.1). Then the projection distances from $\mathbf{p}_i^{\mathbf{q}}$'s to the proxy tangent plane is defined as $|proj_{m,i}^{\mathbf{q}} - \widehat{proj}_m^{\mathbf{q}}|$. The motivation of L_{tan} is to perform linear regression on the patch $P^{\mathbf{q}}$ with the root mean squared *projection* error, where minimization of the term $|proj_{m,i}^{\mathbf{q}} - \widehat{proj}_m^{\mathbf{q}}|$ is the optimzation goal. In an optimal solution, $\widehat{proj}_m^{\mathbf{q}} = \mu_m^{\mathbf{q}}$.

Proof. We want to find $\widehat{proj}_m^{\mathbf{q}}$ that minimizes $L_{tan}^{\mathbf{q}}$. To find such a value, we need to minimize the squared differences. Taking the derivative of the sum of squared differences with respect to $\widehat{proj}_m^{\mathbf{q}}$ and setting it to 0 gives us the

minimum due to convexity:

$$\frac{\partial}{\partial \widehat{proj}_m^{\mathbf{q}}} \sum_{i=1}^{k} [(proj_{m,i}^{\mathbf{q}} - \widehat{proj}_m^{\mathbf{q}}) \cdot \omega_{m,i}^{\mathbf{q}}]^2 = 0 \tag{3.5}$$

$$\sum_{i=1}^{k} -2(\omega_{m,i}^{\mathbf{q}})^2 \cdot (proj_{m,i}^{\mathbf{q}} - \widehat{proj}_m^{\mathbf{q}}) = 0$$

$$\sum_{i=1}^{k} (proj_{m,i}^{\mathbf{q}} - \widehat{proj}_m^{\mathbf{q}}) = 0$$

$$\sum_{i=1}^{k} proj_{m,i}^{\mathbf{q}} - k \cdot \widehat{proj}_m^{\mathbf{q}} = 0$$

$$\widehat{proj}_m^{\mathbf{q}} = \frac{1}{k} \sum_{i=1}^{k} proj_{m,i}^{\mathbf{q}}$$

□

We restrict $\widehat{proj}_m^{\mathbf{q}} = |\mu_m^{\mathbf{q}}|$ so that $\widehat{proj}_m^{\mathbf{q}}$ is non-negative even when $\mathbf{n}_m^{\mathbf{q}}$ is directed against the patch $P^{\mathbf{q}}$ (i.e. $\mathbf{n}_m^{\mathbf{q}}$ is directed towards $P^{\mathbf{q}}$ in an optimal solution). Therefore, the training objective for the optimal $\mathbf{n}_m^{\mathbf{q}}$ is by minimizing the term $|proj_{m,i}^{\mathbf{q}} - \mu_m^{\mathbf{q}}|$. This term is weighed differently with $\omega_{m,i}^{\mathbf{q}}$, which is dynamically assigned using Gaussian kernel depending on the $l2$ distance from $\mathbf{o}_m^{\mathbf{q}}$ to $\mathbf{p}_i^{\mathbf{q}}$. This allows the surface normal estimation to focus more on adjacent points than far points to adapt to point clouds with local density imbalance.

The best approximation of the implicit nearest distance, $t_m^{\mathbf{q}}$, is the projection distance of the nearest point in $P^{\mathbf{q}}$ from $\mathbf{o}_m^{\mathbf{q}}$ due to limited information in the point set:

$$L_{ms}^{\mathbf{q}} = \frac{1}{M} \cdot \sum_{m=1}^{M} |t_m^{\mathbf{q}} - proj_{m,\hat{i}}^{\mathbf{q}}|, \tag{3.6}$$

where $\hat{i} = \arg\min_i \|\mathbf{o}_m^{\mathbf{q}} \mathbf{p}_i^{\mathbf{q}}\|_2$ is the index of the point $\mathbf{p}_i^{\mathbf{q}}$ that minimizes the Euclidean distance from the origin $\mathbf{o}_m^{\mathbf{q}}$. Cumulative query ray depth, $\tilde{t}^{\mathbf{q}} = \sum_{m=1}^{M} t_m^{\mathbf{q}}$, is the sum of all $t_m^{\mathbf{q}}$ through multiple marching steps (Alg.1 and Fig. 3.2). $\mathbf{o}_m^{\mathbf{q}}$ is updated to $\mathbf{d}^{\mathbf{q}} \cdot \tilde{t}^{\mathbf{q}}$ at every step when $\tilde{t}^{\mathbf{q}}$ is accumulated with $t_m^{\mathbf{q}}$.

3.1.2 Neural implicit surface definition with MLP_ϵ

The original sphere tracing [15] does not guarantee precise query ray depth ($\tilde{t}^{\mathbf{q}} \neq \hat{t}^{\mathbf{q}}$) as illustrated in Fig. 3.1. Therefore, we have another distance function, MLP_ϵ, to add a small offset $\epsilon^{\mathbf{q}}$ to get $\hat{t}^{\mathbf{q}} = \tilde{t}^{\mathbf{q}} + \epsilon^{\mathbf{q}}$ (Alg. 1; Fig. 3.2). It also reduces computation redundancy by skipping infinitesimal marching steps and allowing a fixed number of marching steps within a batch for GPU utilization. The difference in the implicit surface definition is that the directional unit vector is given to MLP_ϵ as an input, and the function outputs the distance $\epsilon^{\mathbf{q}}$ only. It enables querying in an arbitrary direction $\mathbf{d}^{\mathbf{q}}$ from an arbitrary $\mathbf{o}_m^{\mathbf{q}}$, which is essential for our user-controlled upsampling. Unlike SDF, it is impossible to know if a ray has penetrated the surface using UDF. Therefore, the $\epsilon^{\mathbf{q}}$ has a constraint to have a non-negative value following [15], which is represented by the loss function:

$$L_\epsilon^{\mathbf{q}} = max(0, -\epsilon^{\mathbf{q}}). \tag{3.7}$$

Instead of L_ϵ, one may suggest the ReLu activation, which outputs $ReLu(\epsilon^{\mathbf{q}}) = max(0, \epsilon^{\mathbf{q}})$, at the last layer to obtain non-negative epsilon values naturally. However, this may introduce the dying ReLu problem [32], where the network does not update when the $\frac{\partial L}{\partial a^{\mathbf{q}}} = 0$, where $a^{\mathbf{q}} = ReLu(\epsilon^{\mathbf{q}})$. For example, given that,

$$L_{MAE} = \frac{\sum_{\mathbf{q}}^{Q} |\hat{t}^{\mathbf{q}} - t^{\mathbf{q}}|}{|Q|}, \tag{3.8}$$
$$L_{RMSE} = \sqrt{\frac{\sum_{\mathbf{q}}^{Q} (\hat{t}^{\mathbf{q}} - t^{\mathbf{q}})^2}{|Q|}},$$

we have,

$$\begin{aligned}\frac{\partial L_{MAE}}{\partial a^{\mathbf{q}'}} &= \frac{1}{|Q|}\cdot\sum_{\mathbf{q}}^{Q}\frac{\partial}{\partial a^{\mathbf{q}'}}|\hat{t}^{\mathbf{q}}-t^{\mathbf{q}}| \\ &= \frac{1}{|Q|}\cdot\sum_{\mathbf{q}}^{Q}\frac{\partial}{\partial a^{\mathbf{q}'}}|\tilde{t}^{\mathbf{q}}+a^{\mathbf{q}}-t^{\mathbf{q}}| \\ &= \frac{1}{|Q|}\cdot\sum_{\mathbf{q}}^{Q}\frac{(\tilde{t}^{\mathbf{q}}+a^{\mathbf{q}}-t^{\mathbf{q}})\cdot\frac{\partial}{\partial a^{\mathbf{q}'}}(\tilde{t}^{\mathbf{q}}+a^{\mathbf{q}}-t^{\mathbf{q}})}{|\tilde{t}^{\mathbf{q}}+a^{\mathbf{q}}-t^{\mathbf{q}}|} \\ &= \frac{1}{|Q|}\cdot\frac{(\tilde{t}^{\mathbf{q}'}+a^{\mathbf{q}'}-t^{\mathbf{q}'})\cdot a^{\mathbf{q}'}}{|\tilde{t}^{\mathbf{q}'}+a^{\mathbf{q}'}-t^{\mathbf{q}'}|},\end{aligned} \tag{3.9}$$

and

$$\begin{aligned}\frac{\partial L_{RMSE}}{\partial a^{\mathbf{q}'}} &= \frac{1}{2\cdot\sqrt{\frac{1}{|Q|}\sum_{\mathbf{q}}^{Q}(\hat{t}^{\mathbf{q}}-t^{\mathbf{q}})^2}}\cdot\frac{1}{|Q|}\cdot\sum_{\mathbf{q}}^{Q}\frac{\partial}{\partial a^{\mathbf{q}'}}(\hat{t}^{\mathbf{q}}-t^{\mathbf{q}})^2 \\ &= \frac{1}{|Q|\cdot 2\cdot\sqrt{\frac{1}{|Q|}\sum_{\mathbf{q}}^{Q}(\hat{t}^{\mathbf{q}}-t^{\mathbf{q}})^2}}\cdot\sum_{\mathbf{q}}^{Q}\frac{\partial}{\partial a^{\mathbf{q}'}}(\tilde{t}^{\mathbf{q}}+a^{\mathbf{q}}-t^{\mathbf{q}})^2 \\ &= \frac{1}{|Q|\cdot 2\cdot\sqrt{\frac{1}{|Q|}\sum_{\mathbf{q}}^{Q}(\hat{t}^{\mathbf{q}}-t^{\mathbf{q}})^2}}\cdot\sum_{\mathbf{q}}^{Q}2(\tilde{t}^{\mathbf{q}}+a^{\mathbf{q}}-t^{\mathbf{q}})\frac{\partial}{\partial a^{\mathbf{q}'}}(\tilde{t}^{\mathbf{q}}+a^{\mathbf{q}}-t^{\mathbf{q}}) \\ &= \frac{1}{|Q|\cdot 2\cdot\sqrt{\frac{1}{|Q|}\sum_{\mathbf{q}}^{Q}(\hat{t}^{\mathbf{q}}-t^{\mathbf{q}})^2}}\cdot 2(\tilde{t}^{\mathbf{q}'}+a^{\mathbf{q}'}-t^{\mathbf{q}'})a^{\mathbf{q}'}.\end{aligned} \tag{3.10}$$

Both partial derivatives are 0 if $a^{\mathbf{q}'}=0$, and this causes the chain rule to set all gradients of the parameters in MLP_ϵ as 0 for an arbitrary $\mathbf{q}'$, and the back-propagation is impossible. To avoid the dying ReLU problem [32], this loss function penalizes the depth overestimation of the last epsilon value instead of placing the ReLU activation at the network's end. An alternative to the proposed loss function is $L_\epsilon = min(0,\epsilon^{\mathbf{q}})^2$, which is differentiable for all $\epsilon^{\mathbf{q}}$. However, Equation 3.7 is chosen to penalize negative $\epsilon^{\mathbf{q}}$ values equally to strictly follow the sphere tracing algorithm [15], where the ray cannot penetrate the surface. Sub-gradients are used when $\epsilon^{\mathbf{q}}=0$, where $\frac{\partial L_\epsilon^{\mathbf{q}}}{\partial \epsilon^{\mathbf{q}}}$ is not differentiable.

3.1.3 Training objective

Given the target depth $t^{\mathbf{q}} \in \mathbb{R}^3$ of a training query ray, $\mathbf{q}$, the training objective of $f(\cdot)$ is by minimizing the following:

$$L_{total} = L_{MAE} + L_{RMSE} + \omega_{ms} \cdot L_{ms} + \omega_{tan} \cdot L_{tan} + L_{\epsilon},$$

where $0 \leq \omega_{ms} \leq 1$ and $0 \leq \omega_{tan} \leq 1$. The losses calculated per query ray, $L^{\mathbf{q}}_{tan}$, $L^{\mathbf{q}}_{ms}$ and $L^{\mathbf{q}}_{\epsilon}$ are averaged (e.g. $L_{tan} = \sum_{\mathbf{q}}^{Q} L^{\mathbf{q}}_{tan}/|Q|$) to obtain the final loss.

3.2 Query ray and patch generation

The origin $\mathbf{o} \in O$ of query rays is sampled using surface normal estimation for each k nearest neighbourhood (k-NN; $k = 16$), $\mathcal{N} \in \mathbb{R}^{k \times 3}$, of downsampled points in the input point cloud:

$$\begin{aligned} \mu_{\mathcal{N}} \in \mathbb{R}^3 &= \left[\frac{\sum_s^{\mathcal{N}} s_x}{|\mathcal{N}|} \; \frac{\sum_s^{\mathcal{N}} s_y}{|\mathcal{N}|} \; \frac{\sum_s^{\mathcal{N}} s_z}{|\mathcal{N}|} \right]^T, \\ \mathbf{o} &= \mu_{\mathcal{N}} \pm ([\lambda_x \; \lambda_y \; \lambda_z]^T)^{\circ \frac{1}{2}}, \\ &= \left[\frac{\sum_s^{\mathcal{N}} s_x \pm |\mathcal{N}|\sqrt{\lambda_x}}{|\mathcal{N}|} \quad \frac{\sum_s^{\mathcal{N}} s_y \pm |\mathcal{N}|\sqrt{\lambda_y}}{|\mathcal{N}|} \quad \frac{\sum_s^{\mathcal{N}} s_z \pm |\mathcal{N}|\sqrt{\lambda_z}}{|\mathcal{N}|} \right]^T, \end{aligned} \tag{3.11}$$

where $\mu_{\mathcal{N}} \in \mathbb{R}^3$ is the mean point of $\mathcal{N} \in \mathbb{R}^{k \times 3}$ and $[\lambda_x \; \lambda_y \; \lambda_z]^T$ is the vector of eigenvalues of the covariance matrix $(\mathcal{N} - \mu_{\mathcal{N}} \cdot \mathbb{1}_k^T)^T \cdot (\mathcal{N} - \mu_{\mathcal{N}} \cdot \mathbb{1}_k^T)$. Because elements are squared or multiplied by each other for covariance calculations, Hadamard square root, $[\cdot]^{\circ \frac{1}{2}}$, is applied to the eigenvalues to calculate the standard deviation calculation of each principal component. This scaling operation places $\mathbf{o}$ at an adequate distance from the surface. The $\pm$ sign is randomly selected with a binomial distribution.

The generation of query rays and patches is based on arbitrary query points, which can be known points in existing point clouds or a novel point from rule-based methods explained in the following sections. The query ray origin corresponding to each query point is determined by searching the nearest $\mathbf{o}$. The query ray direction $\mathbf{d}^{\mathbf{q}}$ is defined as the unit vector from a query

ray origin to a query point. Given a query point, k-NN patches ($k = 16$), $P^{\mathbf{q}}$, is sampled from S followed by relative positioning. Random rotations are applied to patches to augment the data. Depths are normalized to the $(0, 1]$ range by scaling, where the maximum depth between $\mathbf{o}$ and $P^{\mathbf{q}}$ is set as 1.

3.2.1 Known query point sampling for supervised and self-supervised learning

Since the target depths are required to train the proposed network, the known query points and the corresponding depths in the dense ground truth are used for supervised learning. Self-supervised learning is achieved by replacing the query points of dense ground truth point clouds with sparse input point clouds.

3.2.2 Novel query point generation for upsampling

We propose a mid-point algorithm similar to that of Grad-PU [16]. In particular, for every point $\mathbf{s} \in S$, the method finds six nearest points, calculates the midpoint from $\mathbf{s}$ to each of these neighbours and takes each as a query point. We employ an additional constraint where the angles between the vectors from $\mathbf{s}$ to two adjacent midpoints in the neighbourhood, $\mathcal{N}^{\mathbf{s}}_{syn}$, should not be less than $\pi/6$ to maintain a hexagonal shape for uniformity, following the observation by Li et al. [24]. To accomplish this, we give indices to neighbouring points of s from the nearest to the farthest, $S' = \{\mathbf{s}'_1, ..., \mathbf{s}'_i, ...\}$, $\mathbf{s}\,\mathbf{s}'_i$. At each iteration, the midpoint is dropped if any of $\{\angle \mathbf{s}'_i\,\mathbf{s}\,\mathbf{s}'_j \mid j \leq i\}$ is smaller than $\pi/6$. The loop halts until either six midpoints are found or all points in the neighbourhood are checked:

$$\mathcal{N}^{\mathbf{s}}_{syn} = \bigcup_{i=1}^{n} \{\mathbf{s}'_i |\ \forall j \leq i,\ \angle \mathbf{s}'_i\,\mathbf{s}\,\mathbf{s}'_j\ > \frac{\pi}{6}\}, \tag{3.12}$$

$$\tilde{Q}_{syn} = \bigcup_{\mathbf{s}}^{S} \{\frac{s + s'}{2} |\ \forall s' \in \mathcal{N}^{\mathbf{s}}_{syn}\},$$

where n is the smallest number that satisfies $|\mathcal{N}^{\mathbf{s}}_{syn}| = 6$. Duplicate midpoints are omitted, and Farthest Point Sampling (FPS) [39] is used to select query points, $\tilde{Q}_{syn}$, for a specific upsampling rate, r.

We adapt our rule-based query generation method to real-scanned point clouds by altering the k-NN operations by selecting points, $S'' = \{\mathbf{s}''_1, ..., \mathbf{s}''_i, ..., \mathbf{s}''_{n=8}\}$, with the eight smallest angles, $\angle \mathbf{s}\boldsymbol{\alpha}\mathbf{s}''_i$, where $\boldsymbol{\alpha}$ is the sensor location. We define the neighbourhood of s as

$$\mathcal{N}^{\mathbf{s}}_{real} = \{\mathbf{s}'' | \ \forall \mathbf{s}'' \in S'', \ d_{near} < \|\mathbf{s}\,\mathbf{s}''\|_2^2 < d_{far}\}, \tag{3.13}$$

where d_{near} and d_{far} are, respectively the second nearest and the second farthest distances from $\mathbf{s}$ to S'' to reject outliers. This adaptation is more robust to the real-scanned data since LiDAR point clouds are more uniformly distributed in spherical coordinates than in Cartesian coordinates. Additionally, query points are generated on lines with a length greater than the median of all magnitudes of qualifying vectors in S to upsample in regions with a lower density. In particular,

$$\begin{aligned} V &= \bigcup_{\mathbf{s}}^{S} \{\mathbf{s}\,\mathbf{s}'' \mid \forall \mathbf{s}'' \in \mathcal{N}^{\mathbf{s}}_{real}\}, \\ \hat{V} &= \{\hat{\mathbf{v}} | \ \forall \hat{\mathbf{v}} \in V, \ \|\hat{\mathbf{v}}\|_2^2 > median(\{\|\mathbf{v}\|_2^2 \mid \forall \mathbf{v} \in V\})\}, \\ \tilde{Q} &= \bigcup_{n=1}^{N} \{\mathbf{s}^{\hat{\mathbf{v}}} + t^{\hat{\mathbf{v}}} \circ \mathbf{d}^{\hat{\mathbf{v}}} | \ \forall \hat{\mathbf{v}} \in \hat{V}, \ t^{\hat{\mathbf{v}}} = \frac{n}{N+1} \cdot \|\hat{\mathbf{v}}\|_2^2\}. \end{aligned} \tag{3.14}$$

where $\tilde{Q}$ is the set of query points, V is the set of vectors from s to $s'' \in \mathcal{N}^{\mathbf{s}}_{real}$, and $\hat{V}$ is the set of vectors with the magnitude greater than the median magnitude of vectors in V. Instead of selecting a single mid-point, N interpolated query points on $\hat{v} \in \hat{V}$ are generated, where $N = \lceil |\hat{V}|/(|S| \cdot r) \rceil$. Same as the query point sampling for uniformly distributed synthetic point clouds, FPS [39] is applied for a specific r. Outliers in query points and upsampled point clouds concerning their local neighbourhoods in input point clouds are omitted. More specifically, 10 nearest points, $\mathcal{N}^{\tilde{\mathbf{q}}}$, of $\tilde{\mathbf{q}} \in \tilde{Q}$ from S are selected. $\tilde{\mathbf{q}}$ is rejected if:

$$(\tilde{\mathbf{q}}_x > x^{\mathcal{N}^{\tilde{\mathbf{q}}}}_{0.9} \vee \tilde{\mathbf{q}}_x < x^{\mathcal{N}^{\tilde{\mathbf{q}}}}_{0.1}) \vee (\tilde{\mathbf{q}}_y > y^{\mathcal{N}^{\tilde{\mathbf{q}}}}_{0.9} \vee \tilde{\mathbf{q}}_y < y^{\mathcal{N}^{\tilde{\mathbf{q}}}}_{0.1}) \vee (\tilde{\mathbf{q}}_z > z^{\mathcal{N}^{\tilde{\mathbf{q}}}}_{0.9} \vee \tilde{\mathbf{q}}_z < z^{\mathcal{N}^{\tilde{\mathbf{q}}}}_{0.1}), \tag{3.15}$$

where $\tilde{\mathbf{q}}_x, \tilde{\mathbf{q}}_y, \tilde{\mathbf{q}}_z$ are 3D coordinates of $\tilde{\mathbf{q}}$ and $[x^{\mathcal{N}^{\tilde{\mathbf{q}}}}_{0.1}, < x^{\mathcal{N}^{\tilde{\mathbf{q}}}}_{0.9}]$ is the 10% to 90% inter-quantile range of x dimension in $\mathcal{N}^{\tilde{\mathbf{q}}}$. The multivariate ranges define the

ROI of our upsampling objective. Such a practice is more robust for outlier detection than the multivariate location estimator based on the mean, which is sensitive towards outliers [43].

3.3 Network architecture

Our architecture consists of feature encoding and ray marching modules (Figure 3.2). The feature encoding module transforms the 3D coordinates in a patch, $P^{\mathbf{q}}$, into a higher dimension space that includes the neighbourhood information. The features from the feature encoding module are accumulated in the ray marching module to define the UDF of $P^{\mathbf{q}}$.

3.3.1 Feature encoding module

We adopt the Point Transformer's (PT) self-attention module [59] to encode the points within patch $P^{\mathbf{q}}$. First, the points in $P^{\mathbf{q}}$ are fed into a multi-layer perceptron, MLP_F, shared throughout the entire network, to extract $c = 32$ features from the (x, y, z) coordinates of $\mathbf{p}_i^{\mathbf{q}}$, which are inputs to three different layers for key, query, and value in the Point Transformer. The relative coordinates between points are used for positional encoding as in the Point Transformer [59]. The output transformed features, $F_P^{\mathbf{q}}$, have inter-point spatial relationship information with other samples in the patch, making it advantageous for representation learning of neural implicit surfaces. To obtain $F_m^{\mathbf{q}}$ in Alg. 1 and Fig. 3.2, the objective is to calculate the cross-attention (CA) of $\mathbf{o}_m^{\mathbf{q}}$ from $F_P^{\mathbf{q}}$ using the ablated architecture of the Point Transformer's attention module [59]. $F_m^{\mathbf{q}}$ is fed into three different linear layers to obtain the key and the value, while $\mathbf{o}_m^{\mathbf{q}}$ is directly used for the query. The relative 3D coordinates of the local neighbourhood, $P^{\mathbf{q}}$, centred at $\mathbf{o}_m^{\mathbf{q}}$, are input to a separate linear layer for position encoding.

3.3.2 Ray marching module

The ray marching module contains MLP_I and MLP_ϵ that define the neural implicit surface. Depending on the MLP that follows the cross-attention mod-

ule, two types of 3D coordinates are retrieved given the encoded features $F_m^{\mathbf{q}}$ (Fig. 3.2): 1) MLP_I produces the approximate nearest point, $(x_m^{\mathbf{q}}, y_m^{\mathbf{q}}, z_m^{\mathbf{q}})$, on the implicit surface from $\mathbf{o}_m^{\mathbf{q}}$, or 2) MLP_ϵ produces the final $\epsilon^{\mathbf{q}}$ value from $\mathbf{o}_m^{\mathbf{q}}$ to the point where the query ray hits the surface. Both MLPs consist of three fully connected layers with a size of 32, followed by the ReLU activation. The output layers are of size of 3 and 1 are added to the end of MLP_I and MLP_ϵ, respectively.

Chapter 4

Experiments

4.1 Experimentation setups

Following the benchmarks from existing studies, the PU-GAN [24] and PU1K [40] datasets are used for quantitative and qualitative experimentations. The PU-GAN [24] dataset has 120 and 27 training and testing mesh models, respectively. Since no point clouds are provided, the Poisson disk sampling method [57] is used to obtain the input point cloud and the ground truth samples using the program provided by He et al. [16]. PU1K has a larger testing dataset with 127 point clouds. We train our models only on the PU-GAN dataset for all experiments to analyze the domain independence. The training dataset has simple, medium and complex sub-datasets, each with 40 models.

Moreover, we evaluate our method on the 3D reconstruction task for ITS [9], with a simulated real-scanned dataset using the Vista simulator's [1] occlusion removal on densely accumulated point clouds. KITTI-360 [26] (41 point clouds) and private data collected on Alberta highways (233 point clouds) are used for comprehensive assessments in different road environments. The input, low-resolution real-scanned simulation data, follows the configuration settings of the KITTI-360's HDL-64E sensor [26], [47]. Experiments are in the far range with sparse point density ($> 15m$ and $> 30m$ from the sensor for urban and highway environments). Input far-range point clouds have 9741.80 and 15592.06 points on average for urban and highway environments, respectively. We also simulate $2\times$ super-resolution in the horizontal and vertical axes of the range image, similar to that of the state-of-the-art Velodyne Alpha Prime

VLS-128 sensor [21]. The reconstruction ground truths are defined by omitting points in the accumulated point clouds outside of the ROI, as in 3.2.2, followed by FPS for 4× upsampled points. Outliers in the super-resolution point clouds are omitted to mitigate unfair over-penalization. Snippets of the KITTI-360 [26] scenes are used to assess the application on real-scanned data qualitatively. More visual results can be found in the Appendices.

Six models are trained: *Supervised*, $Self_{Test}$, $Self_{Train}$, $Self_{Simple}$, $Self_{Medium}$ and $Self_{Complex}$. *Supervised* is the only supervised model where the query point set is extracted from the dense ground truth point clouds. $Self_{Test}$ and $Self_{Train}$ are self-supervised models, with the input point clouds from the training and test datasets, respectively. $Self_{Simple}$, $Self_{Medium}$ and $Self_{Complex}$ are self-supervised models like the previous two but use a single point cloud with 2,048 training points, randomly sampled from the single, medium and complex training datasets, respectively. The weights of the loss functions are set as $\omega_{ms}, \omega_{tan} = 0.1$ for supervised training, and $\omega_{ms}, \omega_{tan} = 0.5$ for self-supervised learning. Self-supervised learning tends to overfit to the final depth by using input points for both query and patch generation. Thus, the weights on the arbitrary implicit surface learning using L_{tan} and L_{ms} compared to the precise final depth estimation using L_{MSE}, L_{RMSE} and L_{ϵ} are relatively higher for self-supervised training than for supervised training. Different numbers of epochs are also used because of the overfitting problem (*Supervised*: 100 / $Self_{Test}$, $Self_{Train}$: 15 / $Self_{Simple}$, $Self_{Medium}$: 30 and $Self_{Complex}$: 30). The batch size is set as $\lfloor |Q|/64 \rfloor$. The initial learning rate is set as 0.005 and decayed with a rate of 0.99 every epoch with the Adam optimizer [19]. The models are implemented using the PyTorch framework [36] and run on an NVIDIA RTX 3080 GPU Light Hash Rate (LHR) with 10GB of memory, except for the supervised model, which was trained on the NVIDIA RTX A6000 GPU with 48GB to accommodate the larger dataset. The size of query ray origins per sample is $|O| = 128$ to have diverse angles between the tangent plane and the query vector. The choice of the inference $|O|$ is reasoned by the k-NN size of 16 for the query ray origin generation (e.g. $128 = 2048/16$). Thus, a query ray origin covers a similar neighbourhood area generated in the inference stage.

Table 4.1: Quantitative results on the PU-GAN test dataset. All metric units are multiplied by a factor of 10^{-3}. Italic labels indicate a self-supervised method. The best three results are coloured red (first), orange (second) and green (third).

Methods	$r = 4$			$r = 16$		
	CD ↓	HD ↓	P2F ↓	CD ↓	HD ↓	P2F ↓
PU-Net[56]	0.493	4.508	4.315	0.510	6.739	5.442
MPU[49]	0.305	4.463	2.882	0.187	6.243	3.183
PU-GAN[24]	0.296	5.722	2.812	0.229	7.653	3.304
PU-GCN[40]	0.291	2.986	2.471	0.158	3.774	2.592
SAPCU[60]	0.465	10.572	3.421	0.510	6.739	5.442
Grad-PU[16]	0.268	**2.601**	1.990	0.126	**2.628**	2.233
Ours						
Supervised	**0.260**	2.736	**1.707**	**0.121**	2.654	**2.093**
$Self_{train}$	0.274	2.797	2.721	0.155	2.970	3.384
$Self_{test}$	0.293	2.864	2.756	0.141	2.694	3.355
$Self_{simple}$	0.290	6.030	3.109	0.161	5.335	3.757
$Self_{medium}$	0.301	3.260	3.093	0.160	3.200	3.680
$Self_{complex}$	0.325	3.400	3.427	0.181	3.304	4.355

The number of marching steps is set to 6, determined based on an ablation study.

4.2 Applications to Synthetic Datasets

4.2.1 Results on PU1K and PU-GAN datasets

Since PU1K's point cloud inputs are fixed, all the results of the compared methods are taken from the existing papers. Table 4.2 shows that our approach with supervised learning, even with training using the smaller PU-GAN dataset, performs at the state-of-the-art level along with PU-Transformer [42] and Grad-PU [16]. The Chamfer distance result, the best of all compared methods and tied with Grad-PU, is achieved by training without any loss functions for reducing the metric itself or its similarly designed alternatives. Our Hausdorff and P2F distances are ranked in the third and second places, respectively, with small margins compared to state-of-the-art methods. The training objective is mainly on depth prediction, and the naive rule-based query generation method determines the uniform distribution in the output.

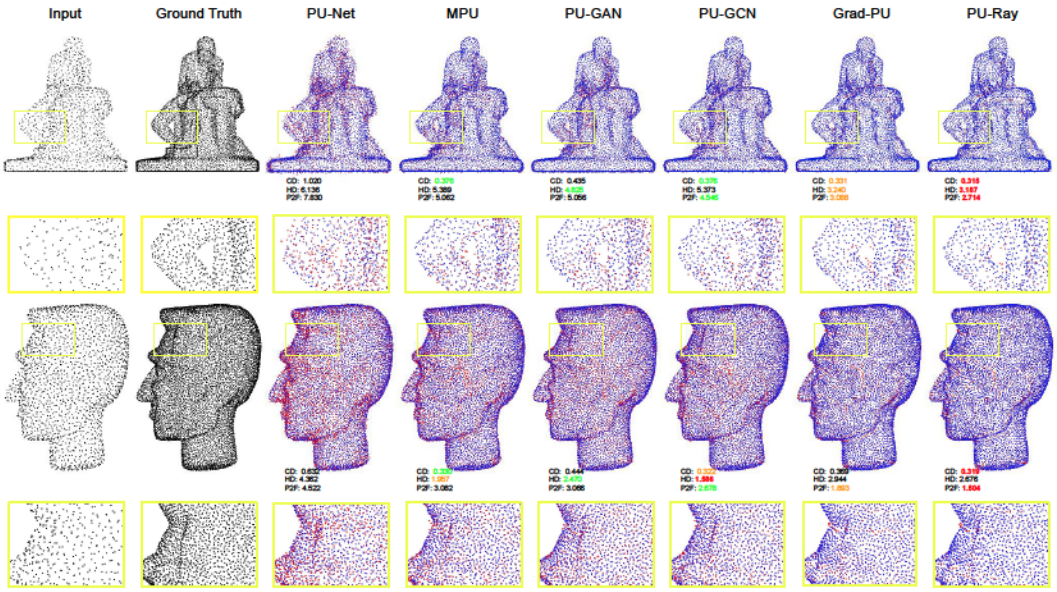

Figure 4.1: Qualitative comparisons between state-of-the-art methods and PU-Ray for 4× upsampling. Point colours are scaled from smaller (blue) to greater (red) P2F distances. The best three results are coloured red (first), orange (second) and green (third).

This illustrates that uniform distribution in the output can be achieved using a simple method instead of a model with a vast number of network parameters to fit the ground truth. A similar trend of PU1K results is also observed in the PU-GAN dataset results (Table 4.1). Our supervised method achieves the lowest Chamfer and P2F distances and the second-best Hausdorff distances of all compared methods in 4× and 16× upsampling. The significant margins in the P2F differences between our method and the Grad-PU [16] illustrate the importance of accurate depth prediction of a query ray. Other fixed-rate methods [24], [40], [49], [56], and upsample 4× twice for 16× upsampling, add more noise to the output. On the other hand, PU-Ray introduces less noise using an arbitrary scaling rate, r. The visual quality of our results is organized in Fig. 4.1 compared to the existing methods. More specifically, we observe that smaller P2F distances occur on complex surfaces. The current models fail to determine whether the space in between (e.g., a triangle surrounded by a leg in the top row of Fig. 4.1) should be an implicit surface or a void. On the contrary, our method's query generation limits where the rays should be directed

Table 4.2: Quantitative results on the PU1K test dataset. All metric units are multiplied by a factor of 10^{-3}. Italic labels indicate a self-supervised method. The best three results are coloured red (first), orange (second) and green (third).

Methods	CD ↓	HD ↓	P2F ↓
PU-Net[56]	1.155	15.170	4.834
MPU[49]	0.935	13.327	3.511
PU-GACNet[14]	0.665	9.053	2.429
PU-GCN[40]	0.585	7.577	2.499
PU-Transformer[42]	0.451	3.843	**1.277**
Grad-PU[16]	**0.404**	**3.732**	1.474
Ours			
Supervised	**0.404**	4.025	1.341
$Self_{train}$	0.431	4.537	1.820
$Self_{test}$	0.427	4.705	1.857
$Self_{simple}$	0.576	12.963	2.114
$Self_{medium}$	0.444	5.181	2.024
$Self_{complex}$	0.452	5.213	2.160

so that they will likely hit the surface. Our superior depth prediction gives smaller numbers of P2F errors on a folded surface, as shown along the hairline of a statue in the bottom row of Fig. 4.1. The self-supervised models also show promising results compared to some existing methods PU-Net [56], MPU [49], PU-GACNet [14] and PUGCN [40]. Moreover, the models trained on a single point-cloud input demonstrate that our method does not require a large dataset to achieve a significant performance improvement, demonstrating its potential for zero-shot learning. All of our five self-supervised models generally outperform SAPCU [60]. Accurate ground-truth depths could provide better guidance for training, while SAPCU has to rely on manually defined implicit surface representations. Table 4.1 and Fig. 4.2 illustrate how SAPCU fails with the PU-GAN test dataset, demonstrating the domain dependency issue. At the same time, all of our self-supervised methods can produce upsampling results with acceptable qualities.

Table 4.3: Quantitative results on the noisy PU-GAN test dataset. All metric units are multiplied by a factor of 10^{-3}. Italic labels indicate a self-supervised method. The best three results are coloured red (first), orange (second) and green (third).

Methods	$\gamma = 0.01$			$\gamma = 0.02$		
	CD ↓	HD ↓	P2F ↓	CD ↓	HD ↓	P2F ↓
PU-Net[56]	0.606	6.291	9.748	1.010	10.449	16.156
MPU[49]	0.460	8.692	7.262	0.782	10.264	13.589
PU-GAN[24]	0.457	6.056	7.294	0.795	9.089	14.217
PU-GCN[40]	**0.436**	5.596	6.981	0.784	8.687	13.583
SAPCU[60]	0.728	16.707	10.933	**0.731**	16.829	10.933
Grad-PU[16]	0.448	**4.474**	**6.447**	0.767	7.194	11.417
Ours						
Supervised	0.454	4.745	6.872	0.749	**7.185**	11.897
$Self_{train}$	0.466	4.794	7.390	0.749	7.441	12.056
$Self_{test}$	0.471	4.664	7.221	0.735	7.353	11.700
$Self_{simple}$	0.482	5.815	7.190	0.785	10.830	**11.270**
$Self_{medium}$	0.477	4.714	7.253	0.738	7.262	11.435
$Self_{complex}$	0.488	4.761	7.466	0.745	7.275	11.526

Table 4.4: Quantitative results of supervised methods on the PU-GAN test dataset with smaller input sizes. All metric units are multiplied by a factor of 10^{-3}. The best three results are coloured red (first), orange (second) and green (third).

Methods	$\lvert S\rvert = 256$		$\lvert S\rvert = 512$		$\lvert S\rvert = 1024$	
	CD ↓	P2F ↓	CD ↓	P2F ↓	CD ↓	P2F ↓
PU-Net[56]	3.073	17.044	1.719	11.657	0.923	7.020
MPU[49]	2.259	9.466	1.186	7.432	0.610	4.604
PU-GAN[24]	**2.152**	9.606	1.133	7.207	**0.572**	4.430
PU-GCN[40]	2.222	8.653	1.124	6.494	0.577	4.007
Grad-PU[16]	2.266	8.583	**1.075**	5.495	0.577	3.336
Ours	2.293	**8.281**	1.105	**4.847**	0.574	**2.841**

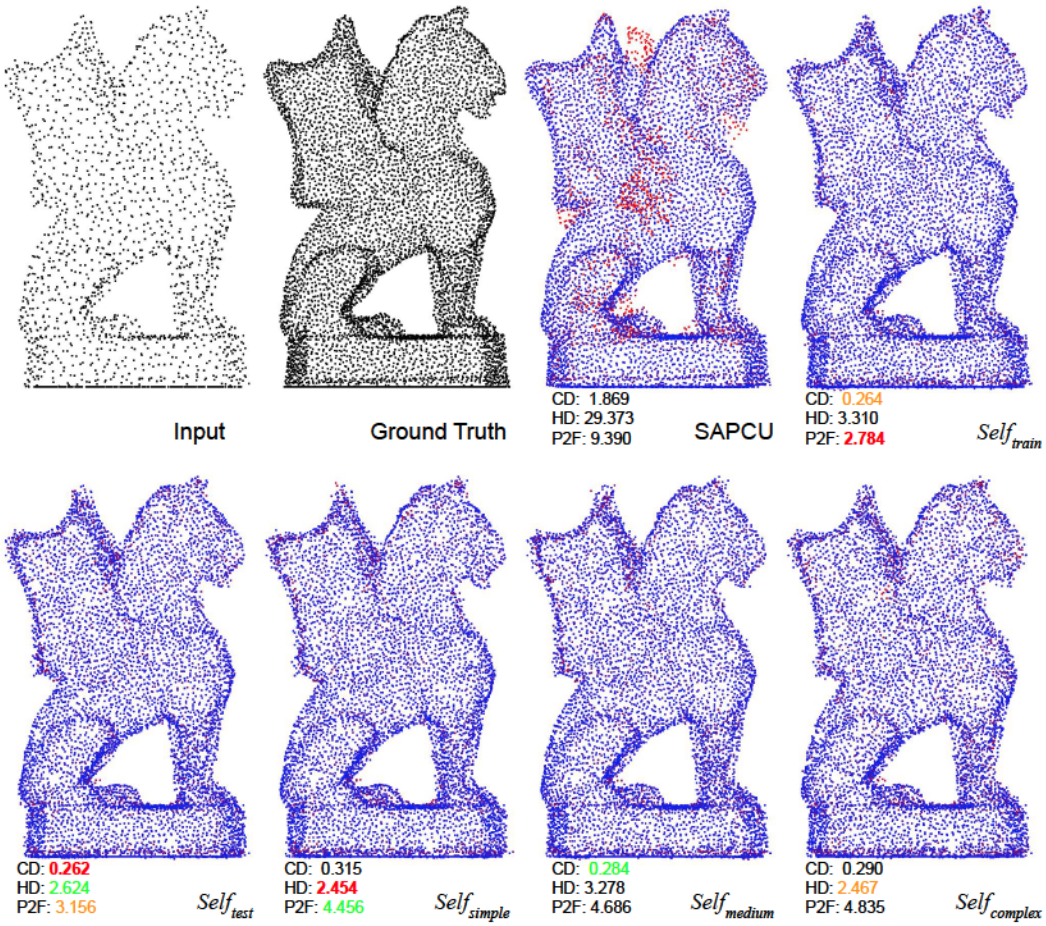

Figure 4.2: Qualitative comparisons between SAPCU [60] and our self-supervised models for 4× upsampling. Point colours are scaled from smaller (blue) to greater (red) P2F distances. The best three results are coloured red (first), orange (second) and green (third).

4.2.2 Robustness to challenging inputs

PU-Ray's robustness to noise was tested by applying the method to perturbed input with Gaussian noise multiplied by two different factors, $\gamma = 0.01$ and $\gamma = 0.02$ (Table 4.3, Fig. 4.3). While the three metric values are slightly behind Grad-PU [16] for $\gamma = 0.01$, the best performances are achieved by several of our models learning at $\gamma = 0.02$. An observation suggests that our self-supervised model's performance degradation is less sensitive to a higher noise level than other methods. Our method preserves the details, such as overhead power lines and tree branches, compared to the noisy output of MPU

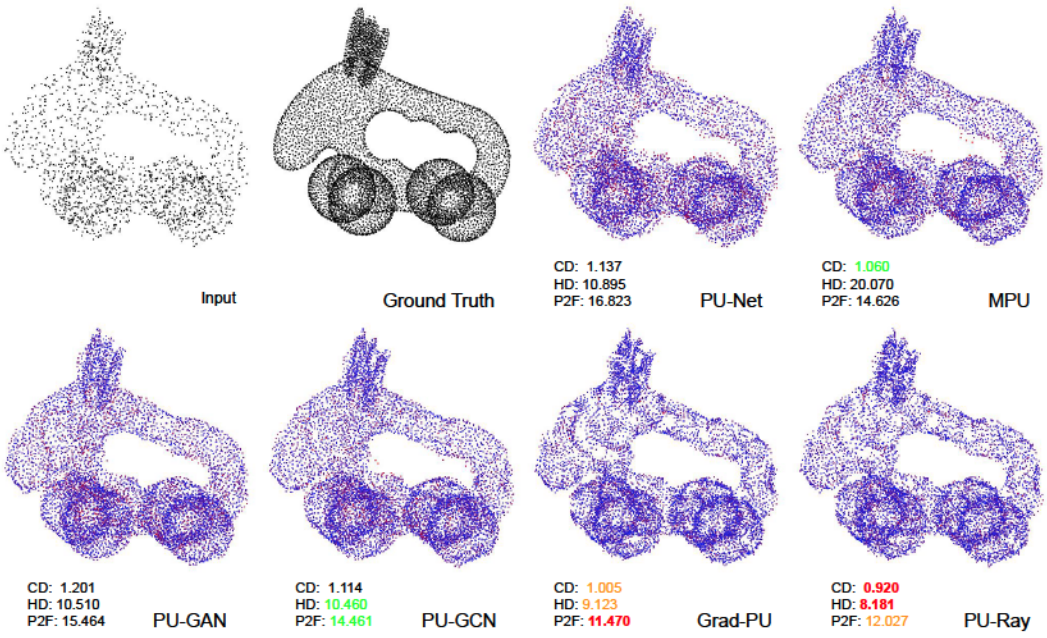

Figure 4.3: Comparisons between upsampling results of existing methods and PU-Ray on a noisy point cloud input with $\gamma = 0.02$ for $4\times$ upsampling. Our method is trained with supervised learning. The best three results are coloured red (first), orange (second) and green (third).

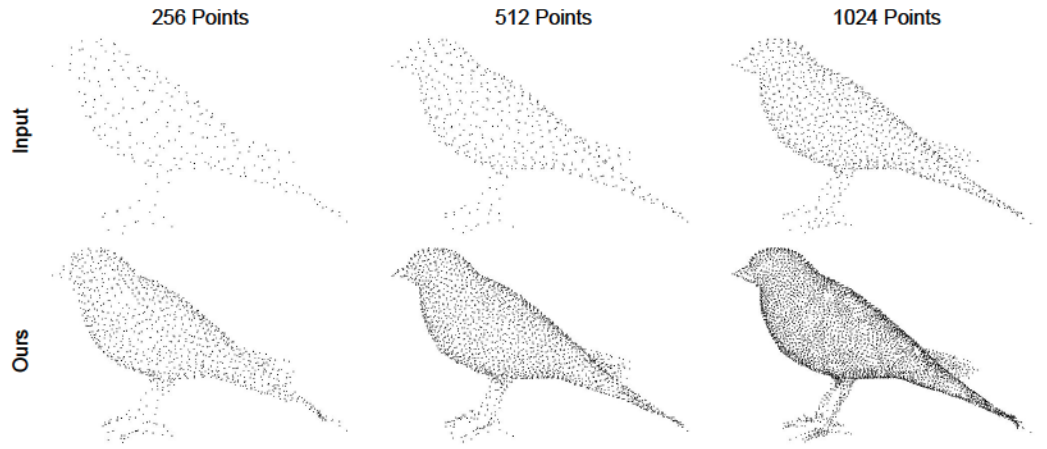

Figure 4.4: PU-Ray's $4\times$ upsampling outputs given different input sizes of 256, 512 and 1024.

[49] while having denser points than the super-resolution results (Fig. 4.3). Evaluation with smaller input sizes shows how well the method adapts to point clouds with a different point density from the training datasets. Fig.

4.4 illustrates that the shape is preserved regardless of the input sizes ranging from 4 to 2 times smaller than the training point clouds. Table 4.4 shows that our method consistently performs the best in the P2F distance metric and competitive results in the CD metric. It is noteworthy that CD is not a good evaluation metric when the compared point clouds have sparse point densities [52]. HD metric is disregarded for the same reason. The results show the method's potential in real-scanned applications where point densities vary in different regions (i.e. decreasing density with the increasing distance to the sensor [54]). It is noteworthy that CD is not a good evaluation metric when the compared point clouds have sparse point densities [52]. HD metric is disregarded for the same reason.

4.3 Applications to real-scanned data

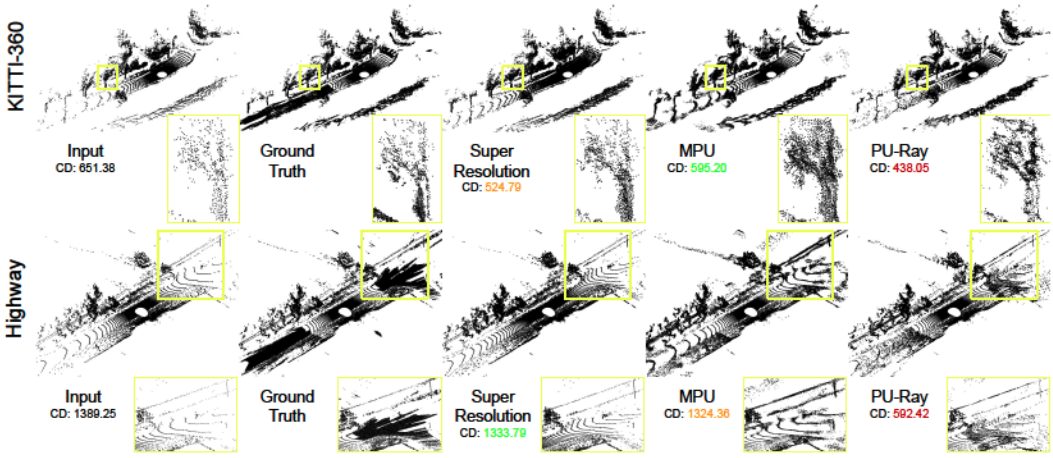

Figure 4.5: Visual comparisons between the input point clouds, super-resolution [1], MPU [49]
and PU-Ray (Supervised) outputs of KITTI-360 [26] and highway datasets.

We demonstrate the effectiveness of our method in the 3D reconstruction task, which is an emphasized application of point cloud reinforcement for intelligent transportation systems (ITS) [9]. Quantitative (Table 4.5) and qualitative results (Figure 4.5) show that the main strength of PU-Ray is demonstrated when it is applied to real-world scenes in both urban (KITTI-

Table 4.5: Reconstruction results on real-scanned data. The best three results are coloured red (1^{st}), orange (2^{nd}) and green (3^{rd}).

Methods	KITTI-360 ($> 15m$)		Highway ($> 30m$)	
	Avg. r	CD ↓	Avg. r	CD ↓
Input (low-res) [1]	-	725.68	-	1295.29
Super-res[1]	3.25×	541.59	3.50×	897.97
PU-Net[56]	4×	807.99	4×	1156.78
MPU[49]	4×	729.01	4×	1146.67
PU-GAN[24]	4×	776.20	4×	1180.13
PU-GCN[40]	4×	732.28	4×	1164.67
Grad-PU[16]	4×	753.16	4×	1307.14
Ours				
Supervised	4×	**523.82**	4×	**788.99**
$Self_{train}$	4×	555.88	4×	822.78
$Self_{test}$	4×	560.10	4×	826.58
$Self_{simple}$	4×	595.18	4×	838.91
$Self_{medium}$	4×	558.65	4×	826.15
$Self_{complex}$	4×	567.13	4×	831.25

360 [26]) and highway environments. The consistently lower CDs suggest the robustness of PU-Ray compared to range-image super-resolution results in 3D reconstruction (Table 4.5). The upsampling behaviour of super-resolution in the 2D space fails to generate points with a uniform distribution in 3D, resulting in clustering (e.g. ring artifacts) (Fig. 4.5). Occlusions by complex surfaces (e.g., trees) may have affected super-resolution to have a lower upsampling rate than $r = 4$ in the far range, and instead, more points are added in the near range. Therefore, range-image super-resolution is not suitable for reconstruction, as it does not have the control of point generations in ROI. Our observation suggests that previous upsampling methods [16], [24], [40], [49], [56] do not overcome the domain dependency problem by generating points coherently with the input points (Fig. 4.5). Compared to Grad-PU's [16] mid-point algorithm, our rule-based query generation allows points in the ROIs where there are wide gaps between input points, which is better suited for uneven distributions commonly observed in real-scanned data. Our method outperforms in preserving details, such as overhead power lines and tree branches, compared to the noisy output of MPU [49] while having denser

Table 4.6: Comparison study of computation costs of upsampling on large-scale real-scanned datasets. A superscript indicates the GPU device used (1080: GeForce GTX 1080 Ti, 2080: GeForce RTX 2080 Ti and 3080LHR: GeForce RTX 3080 LHR).

Methods	Size ($kb \downarrow$)	Train ($sec/epoch \downarrow$)	Inference ($sec/sample \downarrow$)	
			KITTI-360	Highway
Grad-PU^{1080}[16]	401.8	262.00	**3.70**	10.69
Ours^{1080}	**326.6**	**64.62**	6.02	**9.66**
PU-Net^{2080}[56]	796.7	71.41	5.87	12.38
MPU^{2080}[49]	682.6	149.03	6.03	12.49
PU-GAN^{2080}[24]	1053.8	98.20	6.74	13.36
PU-GCN^{2080}[40]	918.4	62.31	6.24	12.76
Ours^{2080}	**326.6**	**58.15**	**5.33**	**8.22**
Ours^{3080LHR}	**326.6**	**35.33**	**3.13**	**4.97**

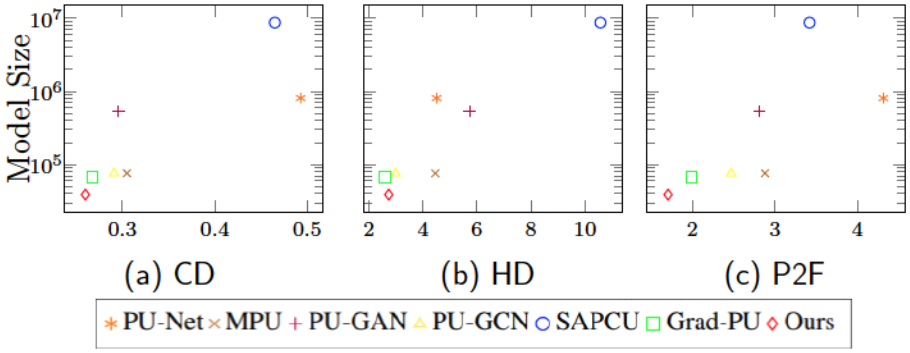

Figure 4.6: Number of parameters versus upsampling performances of PU-Ray and compared models.

points than that of the super-resolution results (Fig. 4.5). Even a patch-based method, MPU [49], could not generate implicit surfaces when data from unseen domains are provided. Illustrations in Fig. 1.2 and Appendices demonstrate upsampling on smaller snippets of the KITTI-360 [26] dataset show that our method can upsample not only at the infrastructure level but also at the object level, such as pedestrians and cars.

4.4 Computation Efficiency Assessment

Our PU-Ray method achieves the smallest model size (3.9k parameters) among all the compared point cloud upsampling methods, which is less than 60% of

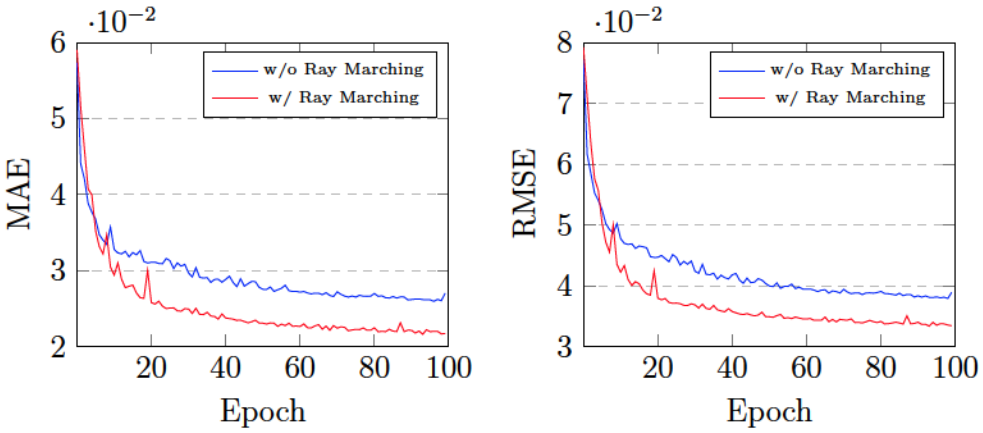

Figure 4.7: Learning curve graphs of the ray marching module on validation depth MAE and RMSE with and without the sphere tracing algorithm.

Grad-PU (6.7k parameters) holding the title before this study. Fig. 4.6 shows how our memory efficiency and performances stand compared to the compared methods. Our method is consistently placed at the bottom left, indicating that it achieves state-of-the-art performance without using up a substantial computational overhead. Considering the case of self-supervised learning, choosing our method can significantly reduce the computation requirements as SAPCU [60] requires 81.2k parameters to run its encoding and decoding processes compared to 38,916 parameters in our method. We experiment with our supervised model trained with a reduced batch size to adapt to a GPU for fair comparisons of the training computation costs between different methods. Our method generally shows faster training and inference speeds, except for the inference speed of Grad-PU [16] on the urban (KITTI-360) dataset (Table 4.6). Our method also shows the smallest gap between inference speeds in urban and highway environments. Processing on large-scale point clouds is performed on GPUs with memory sizes of $10 \sim 11$ gigabytes.

4.5 Ablation studies

In ablation studies, we justify the choice of designs for our method. Fig. 4.7 shows the validation mean absolute errors (MAE) and root mean squared errors (RMSE) of the depth predictions during training of the ray marching

module with and without the spherical tracing algorithm [15]. The consistent results of MAE and RMSE suggest that the ray marching positively affects representation learning of the neural implicit surface through multiple iterations of UDF definition.

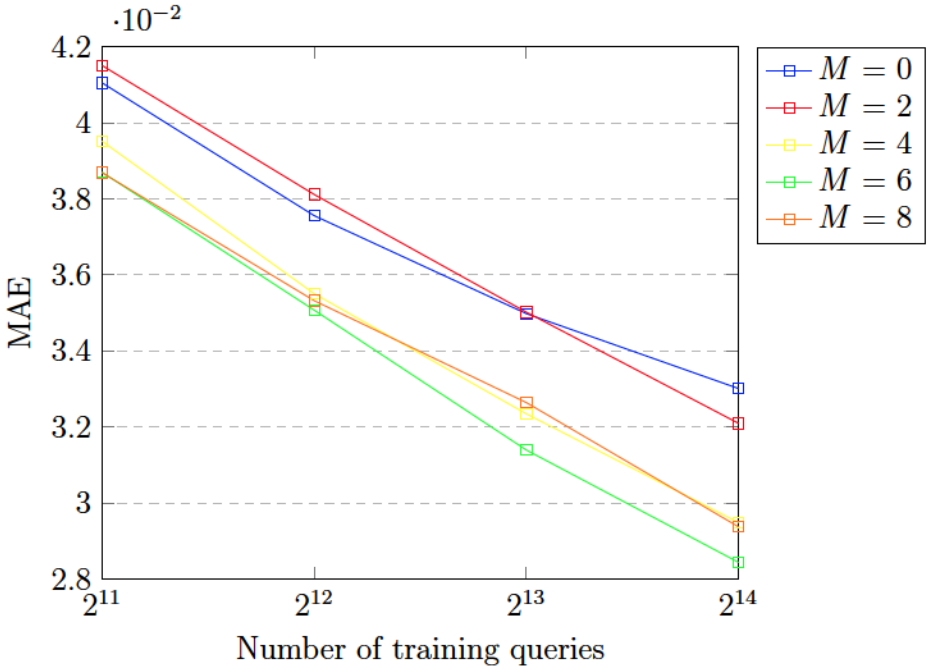

Figure 4.8: The ablation study shows the effect of the number of ray marching steps, M, on depth prediction performance. The depths are normalized.

The effect of the number of ray marching steps on the depth prediction performance is summarized in Fig. 4.8 with $M \in \{0, 2, 4, 6, 8\}$. The mean absolute error (MAE) of the depth prediction is recorded for each model trained with randomly selected query sets with sizes of 2,048, 4,096, 8,192 and 16,384. The numbers are averaged from the results of 4 trials with different random seeds for stability. As per the sphere tracing algorithm [15], the general trend shows that more marching steps result in better performance until six iterations throughout different training set sizes. However, the performance gain is minimal to negative from 6 to 8 marching steps, leading to our method's design choice, which is $M = 6$.

Ablated loss functions show how each term plays a role in the depth prediction (Table 4.7). Theoretically, the depth prediction should be consistent after

Table 4.7: Ablation study on the loss functions. Depth MAEs are reported. The best three results are coloured red (1^{st}), orange (2^{nd}) and green (3^{rd}).

	# of Inference Macrhing Steps		
Loss Functions	4	6	8
w/ L_{MSE} & L_{RMSE} only	0.18241	0.02101	0.08246
w/o L_{ϵ}	0.02705	0.02153	0.02345
w/o L_{tan}	0.02607	0.02005	0.02114
w/o L_{ms}	0.02704	0.01970	0.02049
Ours	**0.02526**	**0.01966**	**0.02025**

Table 4.8: Ablation study on the query generation method.

	CD ↓	HD ↓	P2F ↓
Query Point Generation Method	10^{-3}	10^{-3}	10^{-3}
Simple mid-point	0.271	2.679	1.662
w/ hex-neighbourhood constraint	0.260	2.736	1.707

a few iterations following the sphere tracing algorithm [15]. Also, a better UDF representation of the neural implicit surface is assumed to yield a precise depth prediction regardless of the number of inference marching steps. Therefore, a loss function should be rejected if the resulting model does not show such behaviour. Our novel ϵ estimation shows noticeable precision improvement to the original algorithm. Although removing one of L_{tan} and L_{ms} may not show significant degradation of the final performance, they are essential for following the original algorithm and generating implicit surface representation, as the results do not show consistency in different marching steps. The hexagonal neighbourhood constraint for query generation affects the performance gain in the Chamfer distance with the trade-offs in the Hausdorff and P2F distances Fig. 4.8.

Chapter 5

Conclusion

5.1 Limitations

PU-Ray utilizes the flexibility offered by the Regions of Interest (ROIs) definition to enhance the upsampling process. However, despite this advantage, the quality of the upsampling results heavily depends on the effectiveness of query ray and patch generation techniques. These techniques play a crucial role in accurately capturing the details of the underlying surface geometry. While PU-Ray's mid-point algorithm has demonstrated success in certain scenarios, there are instances where its performance is compromised, particularly when dealing with surfaces of high complexity. In such cases, the algorithm may fail to accurately represent the intricate features of the surface, highlighting the need for further refinement and modifications to ensure consistent and reliable upsampling results across different surface types.

While k-NN operations are commonly used in upsampling methods [10], [14], [16], [25], [31], [40]–[42], [49], [56], [60], [61] due to their simplicity and effectiveness, they have inherent limitations, especially when applied to complex surfaces. The approximation provided by k-NN may not adequately capture the local surface characteristics, leading to inaccuracies in query ray and patch generation. Additionally, the computational complexity associated with k-NN operations makes them unsuitable for time-critical applications where efficiency is paramount. Given the limitations identified with both the mid-point algorithm and k-NN operations, future research efforts should prioritize the development of more precise and computationally efficient techniques

for generating local patches. By addressing these challenges, researchers can improve the overall performance and reliability of upsampling methods like PU-Ray, making them more suitable for a wide range of applications across various surface complexities.

5.2 Summary and Future Work

This paper introduces PU-Ray, a novel ray-based upsampling method designed to address the domain dependency problem prevalent in existing end-to-end point cloud upsampling techniques. PU-Ray leverages the sphere tracing algorithm on a neural implicit surface, as described in [15], to accurately predict query ray depths. Unlike previous methods, PU-Ray employs a rule-based scheme to generate an arbitrary number of query rays, ensuring a more uniform distribution across the point cloud. Additionally, PU-Ray utilizes accurate ground truth data for training query rays, enabling supervised and self-supervised learning approaches applicable to real-world Intelligent Transportation Systems (ITS) scenarios.

PU-Ray achieves state-of-the-art quantitative results on synthetic datasets and noisy inputs while maintaining a minimal number of parameters. Furthermore, extensive experiments conducted on novel datasets for 3D reconstruction demonstrate PU-Ray's robustness in urban and highway environments compared to alternative upsampling methods. The findings of this study suggest that upsampling techniques such as PU-Ray should be preferred over range-image super-resolution methods for accurate 3D reconstruction tasks.

Future research efforts could benefit from exploring the application of ray marching on point clouds to achieve precise depth predictions on real-scanned surfaces. However, it's essential to note that despite the promising results demonstrated by PU-Ray and similar methods, their inference speeds currently align with laboratory experimentation rates due to limited computational resources. Therefore, significant challenges remain in adapting these techniques for practical use with real-scanned data in industrial settings. Future investigations should focus on developing strategies to accelerate these processes and

meet the computational demands of industrial standards in the field of ITS.

References

[1] A. Amini, T.-H. Wang, I. Gilitschenski, *et al.*, "Vista 2.0: An open, data-driven simulator for multimodal sensing and policy learning for autonomous vehicles," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022.

[2] M. Berger, J. A. Levine, L. G. Nonato, G. Taubin, and C. T. Silva, "A benchmark for surface reconstruction," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 2, pp. 1–17, 2013.

[3] J. Bloomenthal and C. Bajaj, *Introduction to implicit surfaces*. Morgan Kaufmann, 1997, ch. Surface Tiling, pp. 126–165.

[4] J. Bloomenthal and C. Bajaj, *Introduction to implicit surfaces*. Morgan Kaufmann, 1997, ch. Ray Tracing Implicit Surfaces, pp. 166–195.

[5] C. Chen, A. Jin, Z. Wang, *et al.*, "Sgsr-net: Structure semantics guided lidar super-resolution network for indoor lidar slam," *IEEE Transactions on Multimedia*, 2023.

[6] J. Chibane, G. Pons-Moll, *et al.*, "Neural unsigned distance fields for implicit function learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 638–21 652, 2020.

[7] J. H. Clark, "Hierarchical geometric models for visible surface algorithms," *Communications of the ACM*, vol. 19, no. 10, pp. 547–554, 1976.

[8] Y. Cui, R. Chen, W. Chu, *et al.*, "Deep learning for image and point cloud fusion in autonomous driving: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 722–739, 2021.

[9] B. Fei, W. Yang, W.-M. Chen, *et al.*, "Comprehensive review of deep learning-based 3d point cloud completion processing and analysis," *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[10] W. Feng, J. Li, H. Cai, X. Luo, and J. Zhang, "Neural points: Point cloud representation with neural fields for arbitrary upsampling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, pp. 18 633–18 642.

[11] Y. Gao, X. Liu, J. Li, Z. Fang, X. Jiang, and K. M. S. Huq, "Lft-net: Local feature transformer network for point clouds analysis," *IEEE transactions on intelligent transportation systems*, vol. 24, no. 2, pp. 2158–2168, 2022.

[12] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu, "Pct: Point cloud transformer," *Computational Visual Media*, vol. 7, pp. 187–199, 2021.

[13] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 12, pp. 4338–4364, 2020.

[14] B. Han, X. Zhang, and S. Ren, "Pu-gacnet: Graph attention convolution network for point cloud upsampling," *Image and Vision Computing*, vol. 118, p. 104 371, 2022.

[15] J. C. Hart, "Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces," *The Visual Computer*, vol. 12, no. 10, pp. 527–545, 1996.

[16] Y. He, D. Tang, Y. Zhang, X. Xue, and Y. Fu, "Grad-pu: Arbitrary-scale point cloud upsampling via gradient descent with learned distance functions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2023, pp. 5354–5363.

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

[18] T. Huang, Q. Liu, X. Zhao, J. Chen, and Y. Liu, "Learnable chamfer distance for point cloud reconstruction," *Pattern Recognition Letters*, vol. 178, pp. 43–48, 2024.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[20] Y. Kwon, M. Sung, and S.-.-E. Yoon, "Implicit lidar network: Lidar super-resolution via interpolation weight prediction," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 8424–8430.

[21] J. Lambert, A. Carballo, A. M. Cano, *et al.*, "Performance analysis of 10 models of 3d lidars for automated driving," *IEEE Access*, vol. 8, pp. 131 699–131 722, 2020.

[22] B. Li, H. Meng, Y. Zhu, *et al.*, "Enhancing 3-d lidar point clouds with event-based camera," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–12, 2021.

[23] K. Li, X. Wang, Y. Xu, and J. Wang, "Density enhancement-based long-range pedestrian detection using 3-d range data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 5, pp. 1368–1380, 2015.

[24] R. Li, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "Pu-gan: A point cloud upsampling adversarial network," in *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[25] R. Li, X. Li, P.-A. Heng, and C.-W. Fu, "Point cloud upsampling via disentangled refinement," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 344–353.

[26] Y. Liao, J. Xie, and A. Geiger, "KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d," *Pattern Analysis and Machine Intelligence (PAMI)*, 2022.

[27] F. Lin, Y. Yue, S. Hou, *et al.*, "Hyperbolic chamfer distance for point cloud completion," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 14 595–14 606.

[28] F. Lin, Y. Yue, Z. Zhang, *et al.*, "Infocd: A contrastive chamfer distance loss for point cloud completion," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[29] H. Liu, K. Liao, C. Lin, Y. Zhao, and Y. Guo, "Pseudo-lidar point cloud interpolation based on 3d motion representation and spatial supervision," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6379–6389, 2021.

[30] S. Liu, Y. Zhang, S. Peng, B. Shi, M. Pollefeys, and Z. Cui, "Dist: Rendering deep implicit signed distance function with differentiable sphere tracing," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2019–2028.

[31] X. Liu, X. Liu, Y.-S. Liu, and Z. Han, "Spu-net: Self-supervised point cloud upsampling by coarse-to-fine reconstruction with self-projection optimization," *IEEE Transactions on Image Processing*, vol. 31, pp. 4213–4226, 2022.

[32] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, "Dying relu and initialization: Theory and numerical examples," *arXiv preprint arXiv:1903.06733*, 2019.

[33] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2015, pp. 922–928.

[34] K. McHenry and P. Bajcsy, "An overview of 3d data content, file formats and viewers," *National Center for Supercomputing Applications*, vol. 1205, p. 22, 2008.

[35] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 165–174.

[36] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035.

[37] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[38] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and multi-view cnns for object classification on 3d data," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5648–5656.

[39] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[40] G. Qian, A. Abualshour, G. Li, A. Thabet, and B. Ghanem, "Pu-gcn: Point cloud upsampling using graph convolutional networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021, pp. 11 683–11 692.

[41] Y. Qian, J. Hou, S. Kwong, and Y. He, "Pugeo-net: A geometry-centric network for 3d point cloud upsampling," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX*, Springer, 2020, pp. 752–769.

[42] S. Qiu, S. Anwar, and N. Barnes, "Pu-transformer: Point cloud upsampling transformer," in *Proceedings of the Asian Conference on Computer Vision*, 2022, pp. 2475–2493.

[43] P. J. Rousseeuw and A. M. Leroy, "Robust regression and outlier detection," in John wiley & sons, 1987, ch. Robust estimation of multivariate location and covariance matrices, including the detection of leverage points, pp. 248–273.

[44] A. Savkin, Y. Wang, S. Wirkert, N. Navab, and F. Tombari, "Lidar upsampling with sliced wasserstein distance," *IEEE Robotics and Automation Letters*, vol. 8, no. 1, pp. 392–399, 2022.

[45] T. Shan, J. Wang, F. Chen, P. Szenher, and B. Englot, "Simulation-based lidar super-resolution for ground vehicles," *Robotics and Autonomous Systems*, vol. 134, p. 103 647, 2020.

[46] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[47] "Velodyne's hdl-64e: A high definition lidar sensor for 3-d applications," 2009. [Online]. Available: `https://api.semanticscholar.org/CorpusID:26835230`.

[48] R. Venkatesh, T. Karmali, S. Sharma, *et al.*, "Deep implicit surface point prediction networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12 653–12 662.

[49] Y. Wang, S. Wu, H. Huang, D. Cohen-Or, and O. Sorkine-Hornung, "Patch-based progressive 3d point set upsampling," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.

[50] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Transactions on Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.

[51] T. Whitted, "An improved illumination model for shaded display," in *ACM Siggraph 2005 Courses*, 2005, 4–es.

[52] T. Wu, L. Pan, J. Zhang, T. WANG, Z. Liu, and D. Lin, "Balanced chamfer distance as a comprehensive metric for point cloud completion," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., 2021, pp. 29 088–29 100.

[53] Z. Wu, S. Song, A. Khosla, *et al.*, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.

[54] Y. Xiong, W.-C. Ma, J. Wang, and R. Urtasun, "Learning compact representations for lidar completion and generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2023, pp. 1074–1083.

[55] Q. Xu, Y. Zhong, and U. Neumann, "Behind the curtain: Learning occluded shapes for 3d object detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 2893–2901.

[56] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P. Heng, "Pu-net: Point cloud upsampling network," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2790–2799, 2018.

[57] C. Yuksel, "Sample elimination for generating poisson disk sample sets," in *Computer Graphics Forum*, Wiley Online Library, vol. 34, 2015, pp. 25–32.

[58] Y. Zhang, W. Zhao, B. Sun, Y. Zhang, and W. Wen, "Point cloud upsampling algorithm: A systematic review," *Algorithms*, vol. 15, no. 4, p. 124, 2022.

[59] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 16 259–16 268.

[60] W. Zhao, X. Liu, Z. Zhong, *et al.*, "Self-supervised arbitrary-scale point clouds upsampling via implicit neural representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 1999–2007.

[61] Y. Zhao, L. Hui, and J. Xie, "Sspu-net: Self-supervised point cloud upsampling via differentiable rendering," in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 2214–2223.

Appendix A

Qualitative Results on the KITTI-360 Dataset

We present more of our qualitative results on snippets of the KITTI-360 dataset [26]. Points are generated in the unknown regions, illustrating the importance of user-controlled upsampling.

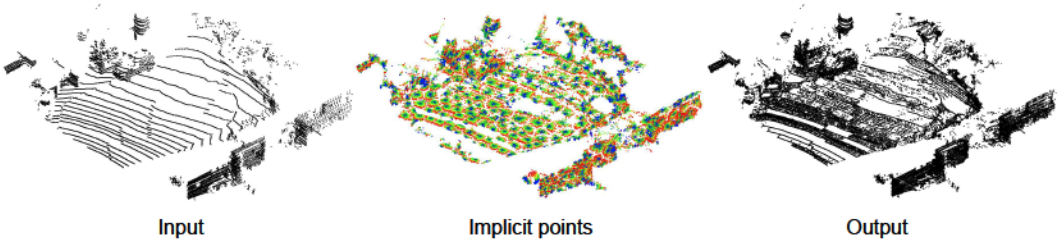

Figure A.1: Qualitative result on a KITTI-360 snippet.

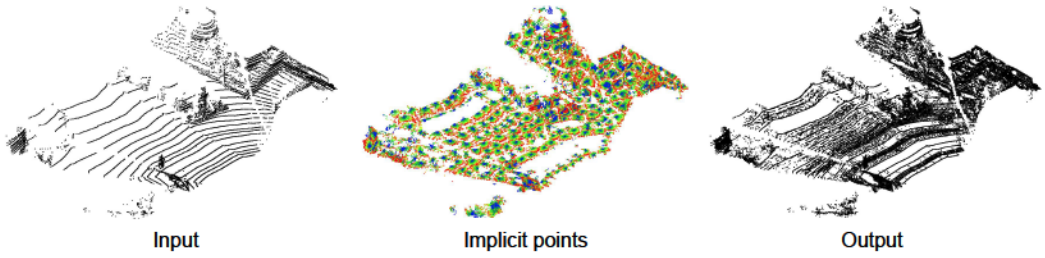

Figure A.2: Qualitative result on a KITTI-360 snippet.

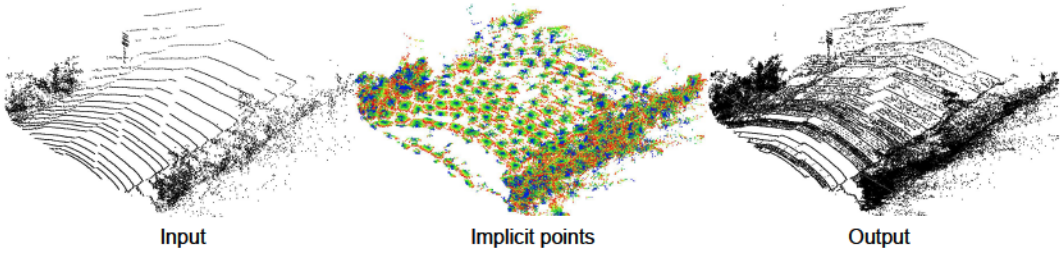

Figure A.3: Qualitative result on a KITTI-360 snippet.

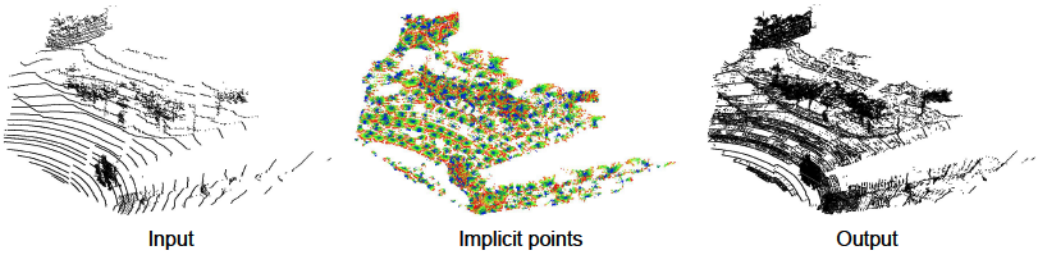

Figure A.4: Qualitative result on a KITTI-360 snippet.

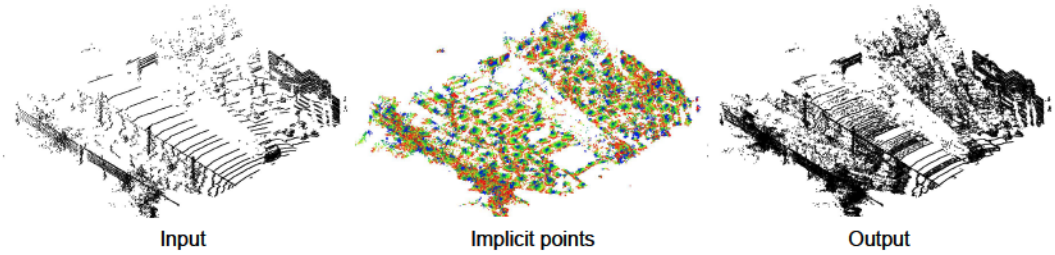

Figure A.5: Qualitative result on a KITTI-360 snippet.

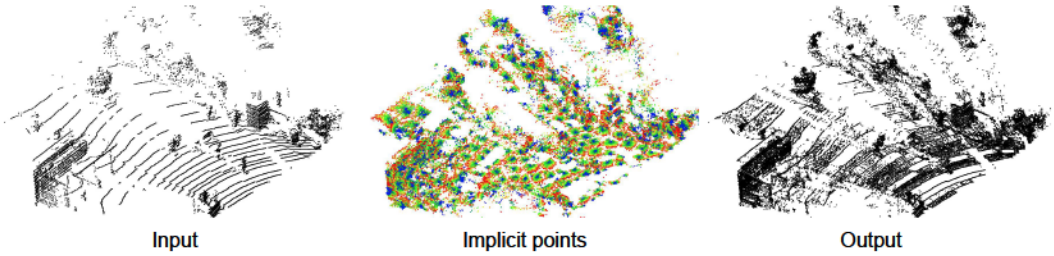

Figure A.6: Qualitative result on a KITTI-360 snippet.

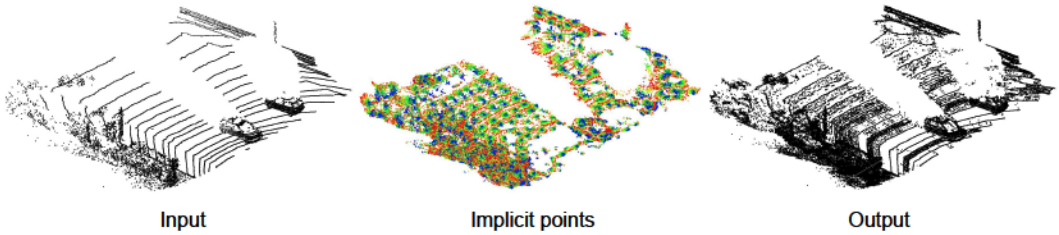

Figure A.7: Qualitative result on a KITTI-360 snippet.

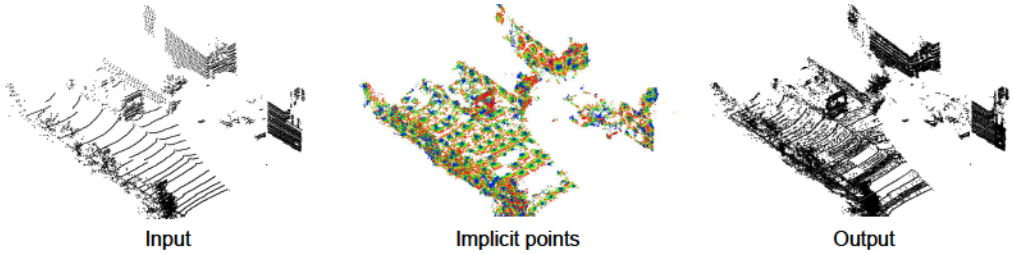

Figure A.8: Qualitative result on a KITTI-360 snippet.

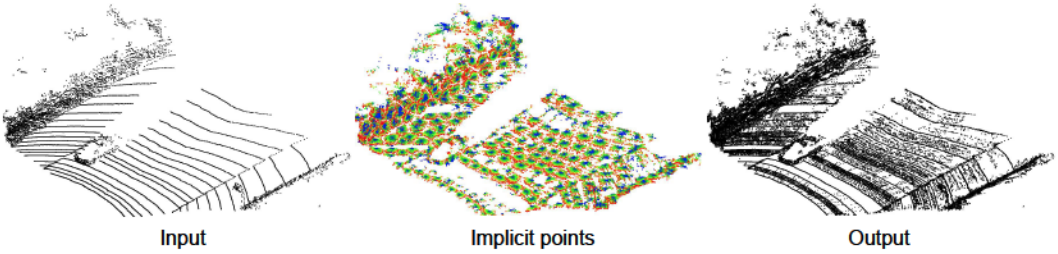

Figure A.9: Qualitative result on a KITTI-360 snippet.