DeepPCD: Enabling AutoCompletion of Indoor Point Clouds with Deep Learning

PINGPING CAI, University of South Carolina, USA
SANJIB SUR, University of South Carolina, USA

3D Point Cloud Data (PCD) is an efficient machine representation for surrounding environments and has been used in many applications. But the measured PCD is often incomplete and sparse due to the sensor occlusion and poor lighting conditions. To automatically reconstruct complete PCD from the incomplete ones, we propose *DeepPCD*, a deep-learning-based system that reconstructs both geometric and color information for large indoor environments. For geometric reconstruction, *DeepPCD* uses a novel patch based technique that splits the PCD into multiple parts, approximates, extends, and independently reconstructs the parts by 3D planes, and then merges and refines them. For color reconstruction, *DeepPCD* uses a conditional Generative Adversarial Network to infer the missing color of the geometrically reconstructed PCD by using the color feature extracted from incomplete color PCD. We experimentally evaluate *DeepPCD* with several real PCD collected from large, diverse indoor environments and explore the feasibility of PCD autocompletion in various ubiquitous sensing applications.

CCS Concepts: • **Human-centered computing → Ubiquitous and mobile computing systems and tools**; • **Computing methodologies → Machine learning approaches**.

Additional Key Words and Phrases: Point Cloud Data, Graph Convolutions, Vision Transformer, Generative Adversarial Networks

ACM Reference Format:
Pingping Cai and Sanjib Sur. 2022. DeepPCD: Enabling AutoCompletion of Indoor Point Clouds with Deep Learning. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6, 2, Article 43 (June 2022), 29 pages. https://doi.org/10.1145/3534611

1 INTRODUCTION

Understanding and interpreting the surrounding 3D environments is a challenging machine perception problem [1, 2], and such perception enables many ubiquitous sensing applications in robotics, drones, autonomous driving, and virtual or augmented reality (VR/AR) [3–6]. To help machines understand 3D objects, shapes, and environments, researchers have developed multiple data structures for machine representation, *e.g.*, 3D Voxels [7], Meshes [8], and Point Clouds [9]. Among them, Point Cloud Data (PCD) is one of the efficient and popular representations and has been used in many research and commercial applications: Mobile robot simultaneous localization and mapping (SLAM) [10]; Object tracking for AR applications [11]; Real-time mapping of floors and surfaces during construction [12]; Vehicle detection in autonomous driving [5]; Object detection and classification in Structural Engineering [13]; Digital elevation models construction in archaeological applications [14]; *etc.*

A PCD is a set of points in 3D Cartesian coordinates embedding the depth and color information of the objects and environment [9]. It can effectively represent 3D objects with simple data structure compared to Meshes, and it can represent finer geometric structures with accurate point positions compared to Voxels. To obtain a PCD, we

Authors' addresses: Pingping Cai, pcai@email.sc.edu, University of South Carolina, USA; Sanjib Sur, sur@cse.sc.edu, University of South Carolina, USA.

2474-9567/2022/6-ART43 $15.00
https://doi.org/10.1145/3534611

use optical devices, such as RGB-D cameras and LiDAR sensors, to scan an environment from different vantage points manually or automatically [15]. However, we face three major challenges in scanning and obtaining a complete, high-quality PCD from an indoor environment: (1) It requires a lot of time and effort from humans or machines for scanning, especially in an environment with large scales; (2) It requires precise planning of the scan trajectories so that the device can cover all environmental surfaces and features; and (3) It requires powerful, long-range camera and depth sensors. Even by solving these challenges, we may not obtain a complete, high-quality PCD due to the sensor occlusion and poor indoor lighting conditions. The resultant PCD could be sparse and incomplete, missing a lot of important geometric and color information about the environment, and using it in 3D sensing applications will result in significant performance degradation [16]. Thus, reconstructing complete, high-quality PCD from incomplete, sparse, or low-quality ones is of vital importance.

Recently, a few research works proposed to use machine learning models to reconstruct high-quality PCD from low-quality inputs [17–20]. Although these methods are effective, they are designed for and tested on the PCD of small 3D objects, such as tables, cars, bikes, *etc.*, where each PCD comprises thousands of points only [21–23]. But for many vision-based ubiquitous sensing applications, we need high-quality PCD of large indoor environments to predict accurate locations and poses, facilitate device navigation, and understand scenes. Besides, the existing methods focus on global shapes and features of small objects during reconstruction. But PCD reconstruction of a large environment requires an emphasis on both the local and global structures, and local geometric information could get buried under the global features during feature extraction and learning. Thus, training existing networks with indoor PCD as input not only would be cumbersome but also may often fail to converge. (See Fig. 6 for an example where existing models fail to reconstruct the shape of walls).

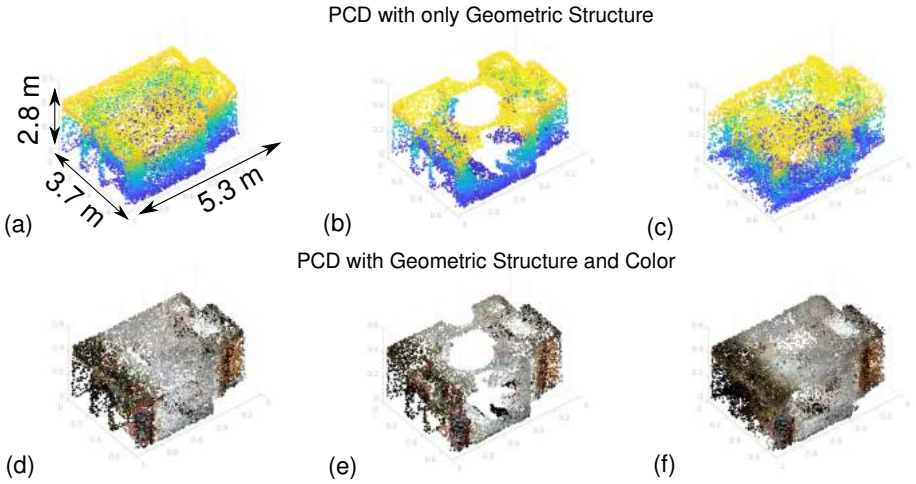

Fig. 1. (a) Ground truth PCD with only geometric structure. (b) Its incomplete structure. (c) Output of *DeepPCD* with only geometric structure. (d) Ground truth PCD with geometric structure and color. (e) Its incomplete structure and color. (f) Output of *DeepPCD* with structure and color.

We propose DeepPCD, a deep-learning-based system that solves the above challenges for high-quality PCD reconstruction of large indoor environments. Rather than asking the user to spend significant time scanning the entire environment in detail, *DeepPCD* lets the user freely scan within a short amount of time and obtain a coarse-grained, low-quality PCD with many missing parts and features. Then, by processing this PCD through learning based reconstruction models, *DeepPCD* outputs a fine-grained, high-quality, and complete PCD. Fig. 1 shows an example of ground truth PCD, its incomplete/low-quality version, and *DeepPCD*'s output. For mobile

and ubiquitous applications that rely on PCD for vision-related tasks, the quality of the PCD determines the applications' performance. Successfully solving this problem can bring significant improvements for these ubiquitous applications. Our system is designed as a preprocessing module for these applications and can be deployed easily without additional software modifications.

To design the reconstruction models, *DeepPCD* leverages three key ideas: (1) Although PCD is a collection of unordered set of points, indoor buildings mostly consist of simple geometric structures, such as straight walls, smooth floors, *etc.*; so, many points could be combined and approximated as a 3D plane. (2) A large environment could be split into multiple parts, and each part could be predicted independently and merged iteratively to reconstruct the full environment; this ensures that not only the reconstructed PCD preserves the accurate local and global structures but also the model converges during training. (3) Walls, floors, and many large objects from similar environments will likely share similar colors; so it may be feasible to infer the missing colors of one PCD from learning color from several examples of ground truth PCD. Although these three ideas are intuitive, it is challenging to synthesize them coherently into a deep learning framework. *DeepPCD* proposes a novel design that outputs high-quality PCD incorporating all three ideas.

At a high-level, *DeepPCD* first reconstructs the geometric structure of the incomplete PCD and then reconstructs its color; so, it trains two models in sequence, where the output from the geometric reconstruction is used as the input to the color reconstruction. *For geometric structure reconstruction*, *DeepPCD* divides the entire PCD into small patches and uses PointNet++ [24], a graph convolution network, as a backbone to extract the local patch features. This division approach is directly following intuition (2) so that *DeepPCD* can extract important local patch-level features with high accuracy. Then, these patch features are fed into a customized 2D Vision Transformer [25], that extracts the global shape features. *DeepPCD* then designs a Plane Point Generator to populate initial 3D points for each patch, and guided by the local patch features, it predicts the point displacements, shifts each point to the target position, and compensates the missing portions using a Multi-Layer Perceptron (MLP). The Plane Point Generator follows the intuition (1) to approximate multiple unordered set of points into 3D planes to facilitate the reconstruction process. Finally, *DeepPCD* merges these reconstructed parts together, and guided by the global features, it uses another MLP to reconstruct a structurally complete PCD. *For color reconstruction*, *DeepPCD* uses a conditional Generative Adversarial Network (cGAN) to infer the color of structurally complete PCD. The choice of cGAN is motivated by intuition (3) where the network could look for similar-looking indoor shapes in training datasets to generate the missing color of shapes at run-time. To this end, *DeepPCD* first extracts the conditional features from the incomplete color PCD using a PointNet++ backbone. Next, given a PCD without color, *DeepPCD* uses an MLP based encoder to encode the geometric features and merges them with the conditional features. Finally, an MLP based decoder infers the full color of the structurally complete PCD.

We implement and evaluate *DeepPCD* with two datasets: (1) Real dataset collected by us using an AR smartphone, and (2) Open-sourced *Stanford Large-Scale 3D Indoor Spaces* (S3DIS) dataset. During training, we feed pairs of incomplete and ground truth PCD into *DeepPCD*, and during testing, we evaluate the efficacy of *DeepPCD* in reconstructing both the geometric and color information of unseen, incomplete PCD. To simulate the incompleteness, we randomly generate holes and distortions in the PCD. Besides, we evaluate the benefit from *DeepPCD* in 3 ubiquitous applications: Localization, navigation, and object detection. Our real dataset consists of 3000 PCD samples from 50 different indoor environments, which are used to train our machine learning models and benchmark the effectiveness of the design components. We find that *DeepPCD* reconstructs the PCD with Chamfer Distance (ChD)[1] ranging from 0.00019 to 0.04189 (smaller the better) across all datasets, and the output closely resembles the ground truth. In contrast, the base-line methods have difficulties in reconstructing correct shapes and the missing portion of PCD. Besides, for the color reconstruction, the reconstructed PCD has an average similarity score ranging from 0.64 to 0.96 (1 is a perfect match) *w.r.t.* the ground truth PCD, and the

[1]ChD is the average squared distance among two closest points between two point sets: A metric to determine PCD quality [18, 20, 23, 26, 27]

reconstructed color PCD is almost similar to the ground truth for many cases. For indoor applications, *DeepPCD* can predict the locations of a device based on the reconstructed PCD with less than 0.01 m median error and track a device with less than 0.04 m median error. In contrast, incomplete PCD typically has median localization and tracking errors of 0.05 m and 0.62 m, respectively. *DeepPCD* can also accurately retrieve objects with a failure rate of less than 15% only.

In summary, we make the following contributions: (1) We design a customized deep learning framework for automatic PCD shape completion of large indoor environments by re-thinking existing models. To the best of our knowledge, *DeepPCD* is the first learning based system that facilitates the reconstruction of a large indoor PCD. (2) We design another customized learning framework to automatically assign colors to the PCD and use them to improve the performance of ubiquitous sensing applications. Our results demonstrate that *DeepPCD* is generalizable in many diverse indoor environments across multiple buildings. To catalyze the PCD based research, we have open-sourced the measured dataset and *DeepPCD* implementation through our project repository [28].

2 BACKGROUND AND CHALLENGES

2.1 Point Cloud Data

A PCD is represented by a set of points in the 3D Cartesian coordinates $[x, y, z]$, which captures the environment's geometrical shape. Each point is also associated with 3 intensity values, $[r, g, b]$, representing their color information. Such data representation stores information of 3D environments directly and can be processed or manipulated by various applications [9]. A key challenge is to obtain PCD with high quality and resolution, and quality and resolution directly affect the performance of many vision-based sensing applications. For example, if the measured PCD is incomplete during movement, the PCD based localization and tracking algorithms [29] could not extract enough distinct features to estimate the current pose or track the device during navigation. Typically, we can improve the PCD quality and resolution by scanning the environment for a longer duration. However, this method not only requires more time and effort but also does not guarantee that PCD will be complete or has high quality due to the camera occlusion, poor lighting conditions, and limited measurement range. Besides, to process and extract information from PCD using deep learning models, we face the following two challenges: (1) Since PCD contains an unordered set of points, the model should be robust to any sequence of input points; such properties are unavailable with traditional convolution based models. (2) The model should be able to extract geometric information among a cluster of points efficiently.

2.2 Challenges for Indoor Point Cloud Completion

A key challenge for indoor PCD completion is to efficiently extract geometric information from the incomplete PCD. Although past works have explored reconstructing high-quality, complete PCD, they mostly work with small 3D objects with thousands of points. For example, VoxelNet [7] converts the PCD to grids and then uses a 3D Convolution Neural Network (CNN) as the feature extractor to improve the object's shape. However, the computation and memory cost for voxel-based models grows cubically with the number of input points [30]. Besides, voxelization can cause a loss of information, even for PCD of small objects. These challenges will amplify with the indoor environments as they not only have large dimensions, requiring millions of points for PCD representation, but also have widely different structures, disparate lighting conditions, and perspective mismatch. Researchers have also proposed graph convolution networks to directly extract good data locality and regularity from the PCD. For example, PointNet++ [24] designs a feature extractor with a graph based convolution kernel and a Farthest Point Sampling (FPS) strategy that samples a subset of points farthest away from each other. Point-GNN [31] extends this method and efficiently encodes the point cloud with a nearest-neighbor graph to extract local relationships among points. It achieves a good reconstruction for PCD of small objects by focusing on

the finer-grained, local features. But the reconstruction of large environments needs emphasis on both the local and global features. Besides, PCD completion is an ill-posed problem [32], *i.e.*, multiple possible reconstruction outputs could fit one incomplete input. So, without prior knowledge and correct guidance, graph convolutions alone may not be sufficient for PCD completion.

3 SYSTEM DESIGN

3.1 Overview

DeepPCD aims to reconstruct complete, high-quality PCD from incomplete ones to improve the overall performance for many PCD-based ubiquitous sensing applications, such as device localization, robot navigation, and object tagging in 3D scenes for VR/AR. Since *DeepPCD* can directly output a high-quality PCD with the same data structures and formats as the original PCD, all these applications could run without additional software modifications. But it will improve the applications' overall performance by substituting the original PCD with the reconstructed ones. Fig. 2 shows the complete system pipeline. It consists of two main components: A geometric structure reconstruction network and a color reconstruction network. The components run in steps where the geometric network first reconstructs the full structure of the incomplete PCD, and then the color network assigs the colors to the points. *DeepPCD* is designed in such a two-step framework because we cannot pre-assign the color to each point before knowing their final geometric position. Besides, a model shows the difficulty in convergence when we attempt to merge the geometric and color reconstruction into one big network to learn the position and color at the same time. We now describe these design components in detail.

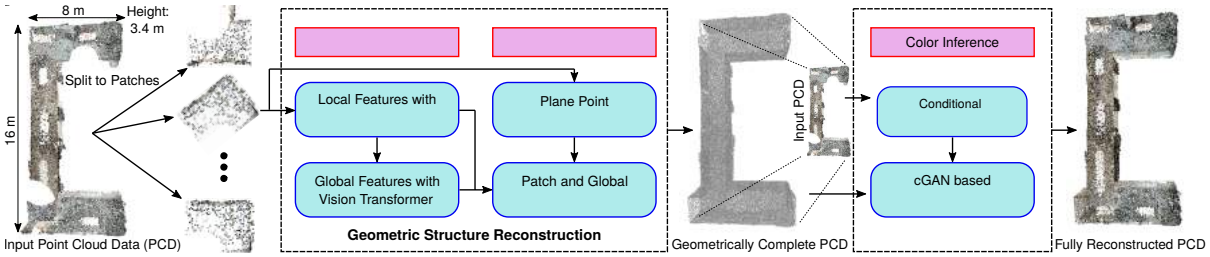

Fig. 2. System overview of *DeepPCD*.

3.2 Geometric Structure Reconstruction

The core purpose of the geometric structure reconstruction network is to reconstruct high-quality structural contents from incomplete, low-quality input PCD. Since an indoor PCD contains many points, it is challenging to efficiently extract features from the large, incomplete PCD and reconstruct it with arbitrary input point order. One solution could be to split the PCD into small parts, use the existing voxel-based or point-based models for reconstructing each part, and geometrically merge them to output the whole PCD. However, this method does not preserve the global geometrical information, and the final output may look distorted at the junction area of each patch. To overcome these challenges and better preserve both local and global geometric information, we propose a novel patch based reconstruction network that splits the input PCD into several patches, adaptively extracts the patch features by considering both their local and global structures, and merges and refines them. Once the PCD is split into small patches, *DeepPCD* passes them into two network blocks: (1) A feature extraction block that extracts the patch-level features as well as the global-level features, and (2) A global reconstruction block that extracts plane information for generating the initial points for small patches, predicts the true point locations using the local and global features, and merges them to produce the final output PCD. Fig. 3 shows the overall geometric structure reconstruction network.

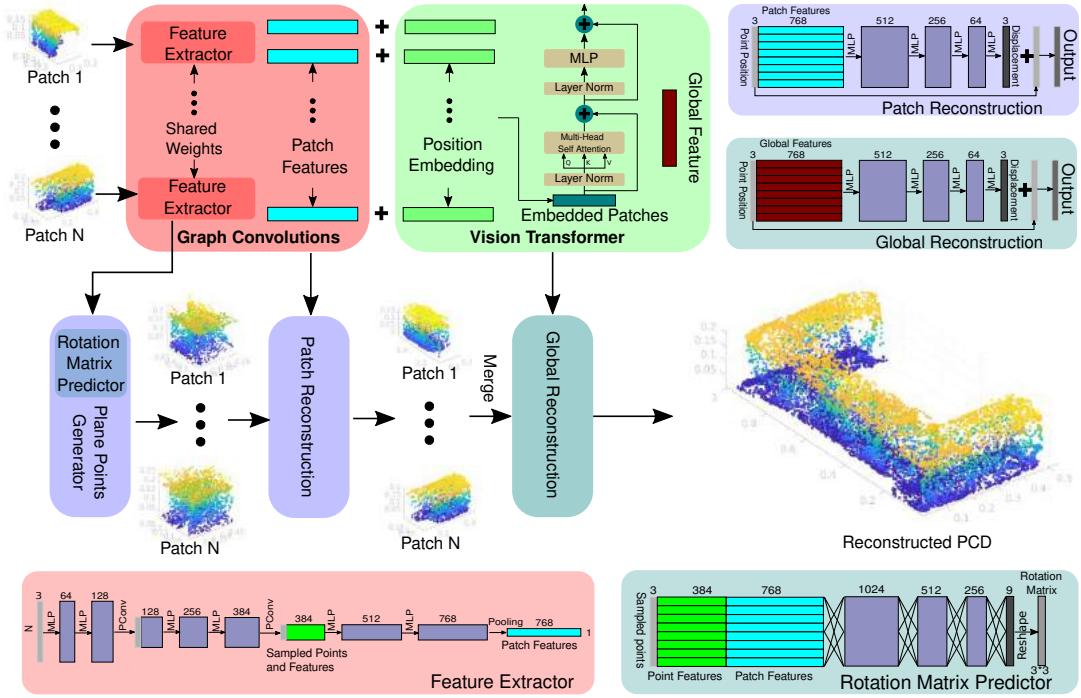

Fig. 3. Geometric structure reconstruction network of the *DeepPCD* system.

3.2.1 Feature Extraction. To extract the patch and global features, *DeepPCD* uses a customized Graph Convolution Network, derived from PointNet++ [24], and a customized Vision Transformer (ViT) network [25].

Local Features Using Graph Convolution Network: A commonly used feature extractor for many Computer Vision related tasks is the convolution network [33]. However, it is difficult to extract features from a 3D PCD using traditional convolution because a point may exist in any 3D location sparsely [34]. Besides, traditional convolutions, such as 3D CNN [35], do not meet two key requirements for PCD feature extraction: (1) The extractor needs to be point permutation invariant, *i.e.*, it should be able to extract features independent of the input order of points, and (2) The extractor should be able to learn not only the individual point features but also the geometrical relationship between points.

To meet these requirements, *DeepPCD* proposes to use a customized Graph Convolution Network as the local feature extractor, built atop the existing PointNet++ model [24]. Given an input PCD with N patches, a straightforward approach could be building N different local feature extractors. Although independently extracting the patch features could be sufficient for individual patch reconstruction, they may not help for capturing the global relationship between patches. To capture these global relationships, *DeepPCD* proposes to share the weights of graph convolution between the patch feature extractors (see Fig. 3). This ensures that not only the network learns the local features per independent patches and is robust to any number of patches, but also all patch features are tuned at the global scale. But sharing the extractors' weights is challenging because points within different patches reside in separate relative coordinate systems; so, their absolute values are not tuned to the global scale. For example, the 1^{st} patch may span the coordinates $(0, 0, 0)$ to $(0.25, 0.25, 1)$ while the N^{th} patch may span the coordinates $(0.75, 0.75, 0)$ to $(1, 1, 1)$. This coordinate gap would confuse the feature extractors and increase the difficulty of training the shared-weight extractor. Thus, *DeepPCD* designs a point

re-scaling technique by scaling each patch coordinate value to $[0, 1]$. *First*, these scaled patches are fed into the shared-weight feature extractors to extract the local patch features. *Then*, these local patch features are used in the reconstruction module to guide and reconstruct local patches (Section 3.2.2). *Finally*, after reconstructing all local patches, they are scaled back to their true coordinates. The output of this module consists of local patch features tuned at the global scale. Table 1 summarizes the local features extraction network parameters.

Table 1. Local feature extraction network parameters. MLP: Multi-Layer Perceptron; PConv: Point Convolution with Farthest Point Sampling; ReLU: Rectifier Linear Unit. F + 3: F is the feature dimension, and 3 is the (x,y,z) coordinate value of the point.

Layer	MLP1	MLP2	PConv1	MLP3	MLP4	PConv2	MLP5	MLP6	MaxPool
Number of Points	N	N	N/4	N/4	N/4	N/8	N/8	N/8	1
Input Channels	3	64	128	128+3	256	384	384+3	512	768
Output Channels	64	128	128	256	384	384	512	768	768
Activation Function	ReLU	ReLU		ReLU	ReLU		ReLU	ReLU	

Global Features Using Vision Transformer: While the above local features encode each small patch, simply merging them may not be suitable for a whole PCD, spanning multiple patches. This is because the local features are still more focused on finer shape reconstruction at a local scale and may be unable to express the global shape of the full PCD. To this end, *DeepPCD* designs a customized 2D ViT for extracting global features from all patches to refine the PCD reconstruction. The purpose of 2D ViT is to combine the patch-level local features and learn the relationship between them to form a PCD-level global features. This global feature could then be used to guide the patch merging process later, ensuring that not only each patch will be placed at the correct location but also the final output will be a smoothly merged PCD. Traditionally, ViTs are designed for 2D computer vision tasks, such as 2D image classification and segmentation [25], where it splits the image into a fixed number of patches and uses a shared convolution kernel to extract the local features. Then, a relative positional embedding based attention mechanism helps ViT to extract the overall global features. With this attention, ViT captures better global features and consequently outperforms traditional CNN [25].

DeepPCD customizes and extends the ViT framework to work with the 3D patches for the PCD. The traditional ViT framework uses a 2D positional embedding with a fixed number of patches. However, PCD from different indoor environments have different sizes and shapes and can have different number of patches. So, the existing fixed positional embedding strategy does not work. To this end, *DeepPCD* proposes a novel positional embedding strategy, inspired by the 1D Sinusoidal positional embedding in speech data transformer [36], that adaptively fits any number of patches inside the ViT framework. The 1D Sinusoidal positional embedding is given by [36]: $PE_{x,2i} = sin(x/10^{8i/d})$ and $PE_{x,2i+1} = cos(x/10^{8i/d})$, where x is the 1D position, and $i \in [0, d/2]$ is the index of each embedding feature with a maximum dimension of d. We extend this positional embedding strategy to make our network aware of the relative position of each patch, independent of the number of patches. We follow the same criteria in [36] to ensure that (1) each patch has a unique encoding for its 2D position, and (2) the embedding difference between any two positions is consistent with their patch distance. Specifically, we enable a 2D Sinusoidal positional embedding with the following:

$$\begin{aligned} PE_{x,y,4i} &= sin(x/10^{16i/d}) \times sin(y/10^{16i/d}); \quad PE_{x,y,4i+1} = sin(x/10^{16i/d}) \times cos(y/10^{16i/d}); \\ PE_{x,y,4i+2} &= cos(x/10^{16i/d}) \times sin(y/10^{16i/d}); \quad PE_{x,y,4i+3} = cos(x/10^{16i/d}) \times cos(y/10^{16i/d}) \end{aligned} \tag{1}$$

where x, y is the 2D position, and $i \in [0, d/4]$ is the index of each embedding feature with a maximum dimension of d. These functions ensure that each dimension of the positional embedding corresponds to a sinusoid, similar to [36], and the wavelengths form a geometric progression from 2π to $10^4 \times 2\pi$. Different from [36], we use 4 elements as basic encoding tuples because we have 2D positions. So, Equation (1) can encode patches at any 2D

position, and any fixed offset $PE_{x+k,y}$ can be expressed as a linear function of $PE_{x,y}$. It enables *DeepPCD* to adapt to a variable number of input patches, still ensuring their features are spatially located at the correct global scale.

Table 2. Global feature extraction network parameters. MLP: Multi-Layer Perceptron; GeLU: Gaussian Error Linear Unit.

Module	Multi-Head Attention				Normalization	MLP	
Layer	Query	Key	Value	Context	Layer Norm	MLP1	MLP2
Input Channels	768	768	768	768	768	768	1024
Output Channels	768	768	768	768	768	1024	768
Activation Function						GeLU	

After adding 2D positional embedding with patch features, we feed them to a transformer encoder consisting of normalization, attention, and MLP layers. This encoder network is the same as the original ViT and is designed to extract the global features from the local patch features. Specifically, our transformer encoder uses one head self attention module, which consists of 3 fully connected layers to generate *Query*, *Key*, and *Value*, respectively. For each patch, its *Query* vector is multiplied with the *Key* and *Value* vectors of all patches to calculate the attention scores and context output, which in turn, fed into a fully connected layer to generate the attention output. This output is concatenated with the original embedded patch feature and fed into an MLP layer to generate the final output. The output of the ViT module is the global feature, which is fed into the reconstruction block to guide the final reconstruction process. Table 2 summarizes the global features extraction network parameters.

3.2.2 PCD Structure Reconstruction. Once *DeepPCD* extracts the patch and global features, it employs the structure reconstruction block leveraging a plane point generator and point displacement prediction networks.

Table 3. Rotation matrix prediction network parameters. MLP: Multi-Layer Perceptron; ReLU: Rectifier Linear Unit.

Layer	MLP1	MLP2	MLP3	MLP4
Number of Points	N	N	N	N
Input Channels	1155	1024	512	256
Output Channels	1024	512	256	3×3
Activation Function	ReLU	ReLU	ReLU	

Plane Points Generator: Since the PCD completion task is an ill-posed problem [32], prior knowledge could help guide *DeepPCD* for better outputs. To this end, we introduce the plane prior and design a plane point generator to populate good initial points and facilitate the reconstruction process. In Cartesian coordinate space (x, y, z), a plane can be expressed as: $a * (x - x_0) + b * (y - y_0) + c * (z - z_0) = 0$, where $[x_0, y_0, z_0]$ is the coordinate of a point on the plane, and $[a, b, c]$ is the normal vector of that point. Ideally, given a point and its normal, we can generate any number of points on this plane and use them as our initial 3D points for reconstruction, but the method will fail for shapes with multiple planes, such as the junction between a wall and ground. To this end, we first generate predefined grid points on the XY plane and then rotate and transform them to construct different 3D planes. For rotation, we use a 3×3 rotation matrix and multiply it with the point coordinates. To automatically learn such rotation matrix, we randomly sample N points inside an incomplete patch, extract their features, and use an MLP network that takes these sampled points' features and local patch features as input and outputs the predicted rotation matrix. The rotation matrix is then multiplied with the predefined grids to populate the rotated 3D plane points, which will be used for the final reconstruction. Table 3 shows the detailed network structure for predicting the rotation matrix.

Patch and Global Reconstruction: The generated plane points for each patch should be moved into their true position to complete the input patches. A strawman approach could be feeding these points into a network

Table 4. Local and global point displacement prediction network parameters. MLP: Multi-Layer Perceptron; LNorm: Layer Normalization; ReLU: Rectifier Linear Unit; Tanh: Hyper-tangent activation function.

Layer	MLP1	MLP2	LNorm	MLP3	MLP4
Number of Points	N	N	N	N	N
Input Channels	768+3	512	256	256	64
Output Channels	512	256	256	64	3
Activation Function		ReLU			Tanh

and letting themselves regress to the right coordinates. However, without additional local and global shape features to guide the reconstruction process, the network cannot learn the position transformation. So, instead of using a direct regression, we design a local point displacement prediction network, similar to [27], that merges the local shape feature with the point coordinates and learns the accurate transformation to identify where those points should move. So, we duplicate the patch feature vector and concatenate them with initial plane points populated by the plane point generator. Then, we feed them into an MLP with a Hyper-tangent (Tanh) activation function to predict the point displacement for each point. Finally, we add the original point position with predicted displacement to generate their final coordinates. Ideally, if we can correctly complete all patches, then a simple merging strategy would be enough to reconstruct the final output. However, as each small patch does not contain the global information, the reconstructed patches still need to be refined by a global module. So, we also use a global point displacement network, with the same architecture as the local displacement, to merge all local patches, refine them, and produce the final structurally complete PCD. Note that both patch and global point displacement prediction networks take 768+3 dimensional features as input, where 768 is the feature dimension, and 3 is the (x,y,z) coordinates of the points. Also, the activation function for displacement prediction is Tanh since the displacement could be in the range [-1,+1] in any direction. Table 4 summarizes the detailed network structure for both the local and global point displacement prediction networks.

3.2.3 Structure Loss Function. All the network blocks rely on their loss functions to tune the convolution/MLP weights appropriately and train themselves. We use the ***Chamfer Distance*** (ChD) between the ground truth and reconstructed PCD as the loss function [26]. Chamfer Distance measures the average squared L2-norm distance among two point sets and is defined as:

$$\mathbf{L}_{ChD}(S_1, S_2) = \frac{1}{N_1} \sum_{p_1 \in S_1} \min_{p_2 \in S_2} ||p_1 - p_2||_2^2 + \frac{1}{N_2} \sum_{p_2 \in S_2} \min_{p_1 \in S_1} ||p_2 - p_1||_2^2 \tag{2}$$

where S_1, S_2 are the two point sets, N_1, N_2 are the number of points in two point sets, and p_1, p_2 are the Cartesian coordinates of the points. Although minimizing ChD loss ensures that the location difference of the points between the ground truth and reconstructed PCD are minimized, it alone does not guarantee to preserve the accurate geometric structure of each patch or the global structure of the reconstructed PCD. To preserve better local and global structures, we propose an additional ***Patch loss*** based on the ChD of the reconstructed patches:

$$\mathbf{L}_{Patch} = \frac{1}{N_p} \sum_{i} \mathbf{L}_{ChD}(P_i, G_i) \tag{3}$$

where N_p is the number of patches, P_i is the reconstructed patch, and G_i is its ground truth patch. Furthermore, to constrain the predicted rotation matrix and ensure that it follows the fundamental property (*i.e.*, $R^T \times R = I$, where R and I are the Rotation and Identity Matrices, respectively), we introduce the ***Rotation loss*** as:

$$\mathbf{L}_{Rot} = ||I - R^T \times R||_2^2 \tag{4}$$

The combined loss function of the geometric structure reconstruction network is then determined as:

$$\mathbf{L}_{Geometry} = \lambda_{ChD} \cdot \mathbf{L}_{ChD} + \lambda_P \cdot \mathbf{L}_{Patch} + \lambda_R \cdot \mathbf{L}_{Rot} \tag{5}$$

where, λ_{ChD}, λ_P, and λ_R are the network hyper-parameters that balance the weight of the global ChD, Patch, and Rotation losses, respectively. They represent the networks' focus on patch, rotation, and global structure adjustments during the reconstruction. Our goal is to find the best set of values for these parameters, and determining the exact values is tricky and difficult. But intuitively, the value for λ_{ChD} should be the largest since it is responsible for accurate reconstruction of global 3D shapes. We will discuss the hyper-parameters tuning in more detail in Section 4. *This network, with its optimized loss function, enable* DeepPCD *to fill up the missing portion of the incomplete PCD and reconstruct an accurate structurally complete PCD.*

3.3 Color Reconstruction

For many vision-based ubiquitous applications, such as device localization, robot navigation, *etc.*, geometric structure information is sufficient to enable their functionalities. However, other applications, such as 3D object tagging in VR/AR, would require both the geometric structure and the color information of the objects and environments. Thus, *DeepPCD also aims to reconstruct the color of the incomplete PCD leveraging the output from the geometric structure reconstruction and using the color reconstruction network.*

3.3.1 Color Reconstruction Network Design. Our network design is intuitive: Walls, floors, and many large objects across different environments in a building or set of buildings will likely share similar colors. So, we could infer the missing colors of the PCD based on the partial color and geometric structures. To this end, *DeepPCD* leverages a cGAN framework, similar to the existing methods [37–39]; but different from their works, it uses the incomplete color PCD as additional information to guide the color reconstruction. We found that cGAN fits the color reconstruction requirements better since it can look for similar shapes in training datasets and generate colors at run-time by extracting color features from incomplete color PCD and then use them as guidance (conditional feature) for predicting the color of geometric reconstructed PCD. At a high-level, *DeepPCD* trains a cGAN framework by leveraging hundreds of incomplete color PCD and the corresponding ground truth (completely colored PCD). cGAN uses a *Generator* **G** to learn the association between the incomplete color to the ground truth and uses a *Discriminator* **D** that trains **G** to learn better association at each epoch [40]. During the run-time, when cGAN has been trained appropriately, **G** can estimate the PCD colors without the ground truth.

GAN Fundamentals: The traditional GAN is designed to learn the regularities and patterns in the input data and then generate new datasets that resemble the input [41]. GAN uses two models for training: A generative model **G** that learns the input data distribution and tries to generate new datasets, and a discriminative model **D** that classifies whether the generated dataset is real or fake. The training procedure for GAN is a zero-sum, two-player, adversarial game [41], where **G** tries to maximize the probability of **D** classifying the generated dataset wrongly, and **D** tries to minimize this probability. The conditional version of the GAN (cGAN), is trained by feeding conditional data to both **G** and **D**, so that the generated datasets are restricted to only a certain domain [40]. Therefore, in *DeepPCD*, we propose a cGAN based model, where the ground truth datasets are only restricted to the indoor PCD, and the generated datasets are conditioned on the completely colored PCD.

Adapting cGAN for 3D PCD: Directly applying the cGAN, however, is infeasible since it is designed for 2D images, but the datasets in *DeepPCD* are 3D PCD. Thus, we customize the traditional cGAN [40] and design appropriate **G** and **D** for indoor PCD. Fig. 4 shows the overall structure of the color reconstruction network.

For the *Generator* **G** design, we first use PointNet++ [24] as the backbone to extract 1D conditional features from the incomplete color PCD. Simultaneously, we design a feature *encoder-decoder* based network to predict the color of each point in the structurally complete PCD. Note that the feature encoder and decoder should meet the following requirements: (1) The networks should not change the number of points in the input PCD, and (2) They should be flexible to process different numbers of points to accommodate various indoor environments. To meet these requirements, we design an MLP based encoder and decoder which can extract high level features from

Table 5. Generator network parameters for PointNet++ feature extractor. MLP: Multi-Layer Perceptron; PConv: Point Convolution with Farthest Point Sampling; ReLU: Rectifier Linear Unit; N1 = N × Sampling rate; N2 = N1 × Sampling rate; Act. Func.: Activation Function.

Layer	MLP1	MLP2	PConv1	MLP3	MLP4	PConv2	MLP5	MaxPool
Number of Points	N	N	N1	N1	N1	N2	N2	1
In Channels	6	64	128	131	256	384	387	512
Out Channels	64	128	128	256	384	384	512	512
Act. Func.	ReLU	ReLU		ReLU	ReLU		ReLU	

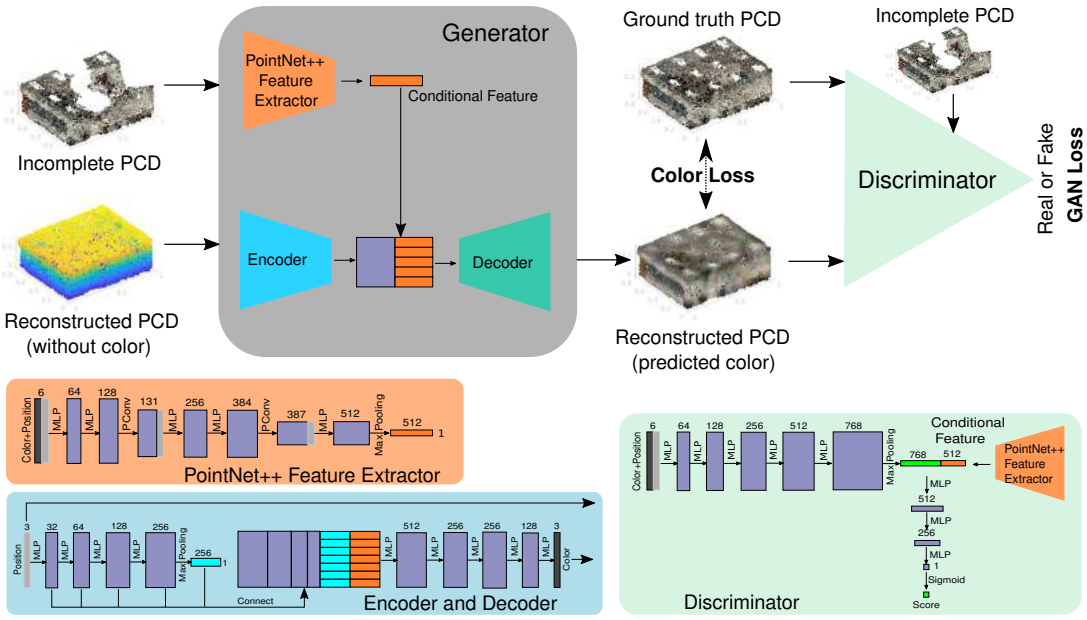

Fig. 4. Color reconstruction network of the *DeepPCD* system.

variable input sizes. But preserving the low level features is also important to reconstruct the color of individual points in the PCD. So, we employ a densely connected architecture and concatenate the encoder output with the intermediate outputs of previous MLP layers and max pooling layer. Finally, after merging the conditional feature with encoded features, we use an MLP based decoder to predict the color of each point.

For the *Discriminator* **D** design, we first use an MLP based network with global max pooling to extract the 1D global information from the generated color PCD and the ground truth PCD. Then, we concatenate it with the conditional feature generated by the PointNet++ and feed them into another MLP network. To output the classification score (*i.e.*, whether the generated PCD is real or fake), **D** uses a Sigmoid activation function. Finally, **G** leverages the score from **D** for adjusting its weights and predicting the accurate color PCD. Tables 5, 6, and 7 show the detailed **G** and **D** parameters for *DeepPCD*'s color reconstruction network.

3.3.2 Color Loss Function. Similar to the structure reconstruction, the color reconstruction network also relies on loss functions to train itself. Recall that **G** tries to generate colors that are as close as the real ones, while **D** tries to distinguish generated colors from the real ones. So, we use a combination of the traditional cGAN loss, $\mathbf{L}_{cGAN}(\mathbf{G}, \mathbf{D})$ [40], and point-to-point L1-norm color loss, $\mathbf{L}_C(\mathbf{G})$ [38, 39]. Color loss helps **G** in predicting a point color by minimizing point-to-point mean absolute error, while cGAN loss maintains the global adversarial game.

Table 6. Generator network parameters for the encoder and decoder.

Layer	MLP1	MLP2	MLP3	MLP4	MaxPool	MLP5	MLP6	MLP7	MLP8	MLP9
Number of Points	N	N	N	N	1	N	N	N	N	N
Input Channels	3	32	64	128	256	1248	512	256	256	128
Output Channels	32	64	128	256	256	512	256	256	128	3
Activation Function	ReLU	ReLU	ReLU	ReLU		ReLU	ReLU	ReLU	ReLU	ReLU

Table 7. Discriminator network parameters.

Layer	MLP1	MLP2	MLP3	MLP4	MLP5	MaxPool	MLP6	MLP7	MLP8	Sigmoid
Number of Points	N	N	N	N	N	N	1	1	1	1
Input Channels	6	64	128	256	512	768	1280	512	256	1
Output Channels	64	128	256	512	768	768	512	256	1	1
Activation Function	ReLU	ReLU		ReLU	ReLU		ReLU	ReLU	ReLU	

The traditional cGAN loss and the L1-norm color loss are defined as:

$$\mathbf{L}_{cGAN}(\mathbf{G},\mathbf{D}) = \mathbb{E}_x[log\ (\mathbf{D}(x|y))] + \mathbb{E}_z[log\ (1-\mathbf{D}(\mathbf{G}(z|y)))]; \qquad \mathbf{L}_C(\mathbf{G}) = \mathbb{E}_z[||C_{Real} - \mathbf{G}(z|y)||_1] \tag{6}$$

where x is the predicted color PCD, y is the conditional extra information, and z is the input PCD without color. Note that in the original cGAN, z is defined as the noise to generate non-deterministic outputs. However, in *DeepPCD*, we generate the deterministic color output using the L1-norm loss. Here, C_{Real} is the ground truth color PCD. Similar to [38], we use a combination of the cGAN loss and L1 color loss, which is defined as:

$$\mathbf{L}_{Color} = \mathbf{L}_{cGAN}(\mathbf{G},\mathbf{D}) + \lambda_C \cdot \mathbf{L}_C(\mathbf{G}) \tag{7}$$

The hyper-parameter λ_C is used to balance the cGAN loss and L1 color loss. Intuitively, for our color reconstruction, both $\mathbf{L}_{cGAN}$ and $\mathbf{L}_C$ are equally important; so, the losses should have equal weights. *This network, with its optimized loss function, enables DeepPCD to reconstruct the color PCD similar to the ground truth.*

4 IMPLEMENTATION

4.1 Hardware Platform and Datasets

To implement and evaluate *DeepPCD*, we use two datasets. The first one is collected by us using an AR-capable smartphone, ASUS Zenfone AR [42], with an RGB camera and depth sensor. The smartphone is equipped with a 22.7 megapixel (MP) RGB camera and a 0.3 MP depth sensor with 77° field of view. The camera and depth sensor have sampling rates of 30 fps and 1.8 fps, respectively. The maximum range of the depth sensor is 6 m only; so, to scan a large environment, we walk around by holding the phone with different poses. We collect PCD of 25 large, diverse indoor environments across 3 buildings using a SLAM app, RTABMap [43], running in real-time on the smartphone, and then, extract the PCD offline to a host PC.

To collect the ground truth PCD without holes and incompleteness, we scan the environment in detail across all ceilings, floors, walls, and objects. These detailed scans enable us to not only capture the high-quality PCD but also train, test, and benchmark all our design components. The indoor environments include general-purpose hallways, office spaces, lobby, *etc.* Each collected PCD contains over 3 million points with detailed geometric structure and color information, and it takes an average 15 min for scanning. After collecting each raw, complete PCD, we notice that the ground truth PCD could still contain many redundant parts and floating points caused by the RGB and depth sensor pollution. Since the noisy data could affect the network training and applications, we pre-process the PCD. To this end, we use a MeshLab application [44] that allows us to remove unnecessary parts manually as well as crop the PCD and tag meaningful objects in it.

The second dataset is from an open-sourced *Stanford Large-Scale 3D Indoor Spaces* (S3DIS) [45], containing PCD of 6 different big floors with many rooms. Compared to our dataset, which contains mainly large open

areas, such as hallways and corridors, S3DIS contains mainly small areas, such as classrooms and offices. We also select 25 distinct environments from S3DIS. *Combining these two datasets allows us to evaluate the versatility of DeepPCD in reconstructing complete PCD of various indoor environments with diverse structures.*

To expedite the training/testing time, we downsample the PCD to contain approximately 20,000 points using a Poisson-disk sampling [46]. Note that due to the MLP based network design, *DeepPCD*'s trained models are not constrained by the number of input points and can be used without any network architecture change for a larger PCD. Before feeding PCD into the *DeepPCD* networks, we scale them to [0,1] for both RGB and location attributes and split them into small patches. This scaling step helps to speed up the model training and convergence time. The splitting process is very similar to splitting a 2D image, but instead of splitting the PCD into 3D volumetric patches, we split it across the length and breadth. This is because most indoor environments have large length and width and small (mostly fixed) heights. We split the PCD into 5×5 grid patches across the length and breadth and store the position of every small nonempty patch. But the total number of PCD is still too small for robust training; so, we augment the datasets synthetically. *First*, we randomly crop a complete PCD to generate 5 different PCD. *Second*, we introduce incompleteness by randomly generating small holes inside the cropped PCD, that mimics PCD incompletion of real data collection: We select several different random seed points, and for each seed, we remove 3% of the total points closest to the seed. *Finally*, we generate 10 different incomplete PCD for every cropped PCD, and each of them has different regions of incompleteness. In summary, our collected and S3DIS datasets contain a total 3000 distinct PCD samples. For each dataset, we use 1200 samples for training *DeepPCD*, and the rest of the samples are used for testing and benchmarking all our design components and applications.

4.2 Network Training

We explore different network settings to ensure *DeepPCD* converges to the optimal parameters. For the structure reconstruction, we first set the total epoch to be 500 and use "Adam" as the optimization function with a learning rate of 3×10^{-4}. Then, we design a smart learning scheduler to speed up the training process: Specifically, we evaluate the network performance every 20 epochs, and if we observe the validation error is on a plateau, we automatically decrease the learning rate by a scale of 0.1. It allows the network to converge to the optimal settings faster. We also explore different combinations of hyper-parameters by training the network multiple times and find that the geometric structure reconstruction performs much better when the ratio between λ_{ChD} and λ_P is 2×, and the ratio between λ_{ChD} and λ_R is close to 1000×, *e.g.*, $(\lambda_{ChD}, \lambda_P, \lambda_R) = (1, 0.5, 0.001)$. This fits our intuition because *DeepPCD* cares more about the final global reconstruction results than the local patch reconstruction, and the rotation loss should be much smaller than both of them to prevent the network from learning incorrectly. Across all datasets, our network converges successfully within 500 epochs.

For the color reconstruction, we set the total epoch to be 500 and use "Adam" as the optimization function. Note that, for cGAN, we need to train two models, the *Generator* **G** and the *Discriminator* **D**, simultaneously. To this end, we follow the traditional GAN training procedure, and in each epoch, we first fix **G** and only train **D** with a Binary Cross Entropy loss [41]. Then, we fix **D** and train **G** with both the Binary Cross Entropy loss and the Color loss. After trying different combinations of learning rates, we set the optimal rate of **G** and **D** as 2×10^{-4} and 1×10^{-4}, respectively. For hyper-parameter settings, we also explore different combinations and find that the color reconstruction network performs much better with equal weightage on cGAN and color loss, *i.e.*, $\lambda_C = 1$.

The networks are implemented in PyTorch with Python 3.8 and CUDA support [47] in a server with Nvidia's RTX A6000 [48]. For structure reconstruction, the network takes ~0.5 days to complete the training, and for color reconstruction the network takes ~2 days to complete the training.

5 PERFORMANCE EVALUATION

We evaluate *DeepPCD* using 2 metrics commonly adopted to compare the point cloud quality and additional 2 metrics for evaluating applications based on the PCD. Furthermore, to illustrate the efficacy of *DeepPCD*,

we compare its performance with 2 traditional 3D-vision based algorithms: Plane Fitting [49] and Hole Filling [50], as well as 2 advanced deep-learning-based algorithms: FoldingNet [23] and PCN [18]. Also, to evaluate *DeepPCD*'s robustness, we perform multiple ablation studies and estimate the metrics under different network settings and environments. We also explore 3 ubiquitious applications based on the indoor PCD and compare their performances with and without *DeepPCD*.

► **Chamfer Distance (ChD)**: The objective measure of the differences between the point locations inside the reconstructed PCD *w.r.t.* the ground truth PCD [18, 20, 23, 26, 27]. It is estimated as the average squared distance among two closest point pairs between two point sets: Smaller the ChD between the ground truth and reconstructed PCD, better the reconstructions. The unit of ChD is $meters^2$, and its scale goes from 0 and ∞.

► **Structural Similarity Index Measure (SSIM)**: The objective measure of distortion of structural information between two colored images [51]. SSIM is designed for 2D images; so, we follow [52] to map the 3D PCD into voxel-images by voxelizing the reconstructed and ground truth color PCD and then projecting them into 2D images on 3 isometric planes. Finally, we calculate the average SSIM between these projected 2D images. The SSIM scale goes from 0 to 1, where 1 means a perfect match between pixels of ground truth and reconstruction.

► **Localization and Tracking Errors**: The absolute error between the ground truth and predicted locations obtained from the ground truth and reconstructed PCD, respectively.

► **Object Tagging Accuracy**: The objective measure to find the correct number of objects in the reconstructed PCD with reference to the ground truth PCD.

Evaluation Summary: (1) For the structure reconstruction, *DeepPCD* achieves a median ChD of 0.00045 and outperforms all the base-line algorithms, both traditional 3D vision-based and advanced deep learning based, by significant margins. The results are consistent across diverse environments from two different datasets, and most reconstructed PCD is visually similar to the ground truth. (2) For the color reconstruction, *DeepPCD* achieves a median SSIM of 0.89 *w.r.t.* the ground truth across all test cases, and the reconstructed colors match well with the ground truth colors in many cases. (3) For the indoor localization and tracking, *DeepPCD* reduces the average errors by 0.04 m and 0.17 m, respectively, compared to the incomplete PCD based approaches, and the results are similar to the ground truth PCD based approaches. Furthermore, *DeepPCD* is able to reduce the error in object tagging from 26.5% to 16.4% across all test environments.

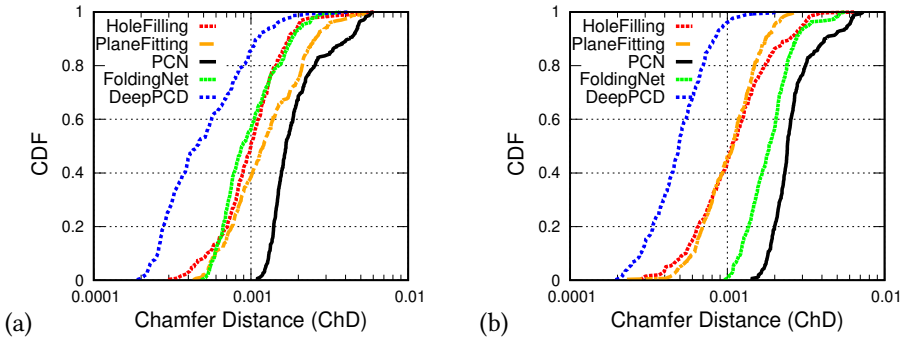

Fig. 5. ChD distribution for the PCN, FoldingNet, Plane Fitting, Hole Filling, and *DeepPCD* for 300 test samples in each dataset: (a) Ours; (b) S3DIS.

5.1 Geometric Structure Reconstruction

We start by evaluating the performance of *DeepPCD*'s geometric structure reconstruction across S3DIS and our datasets. For each dataset, we have 300 test samples and we estimate the ChD for the reconstructed PCD *w.r.t.*

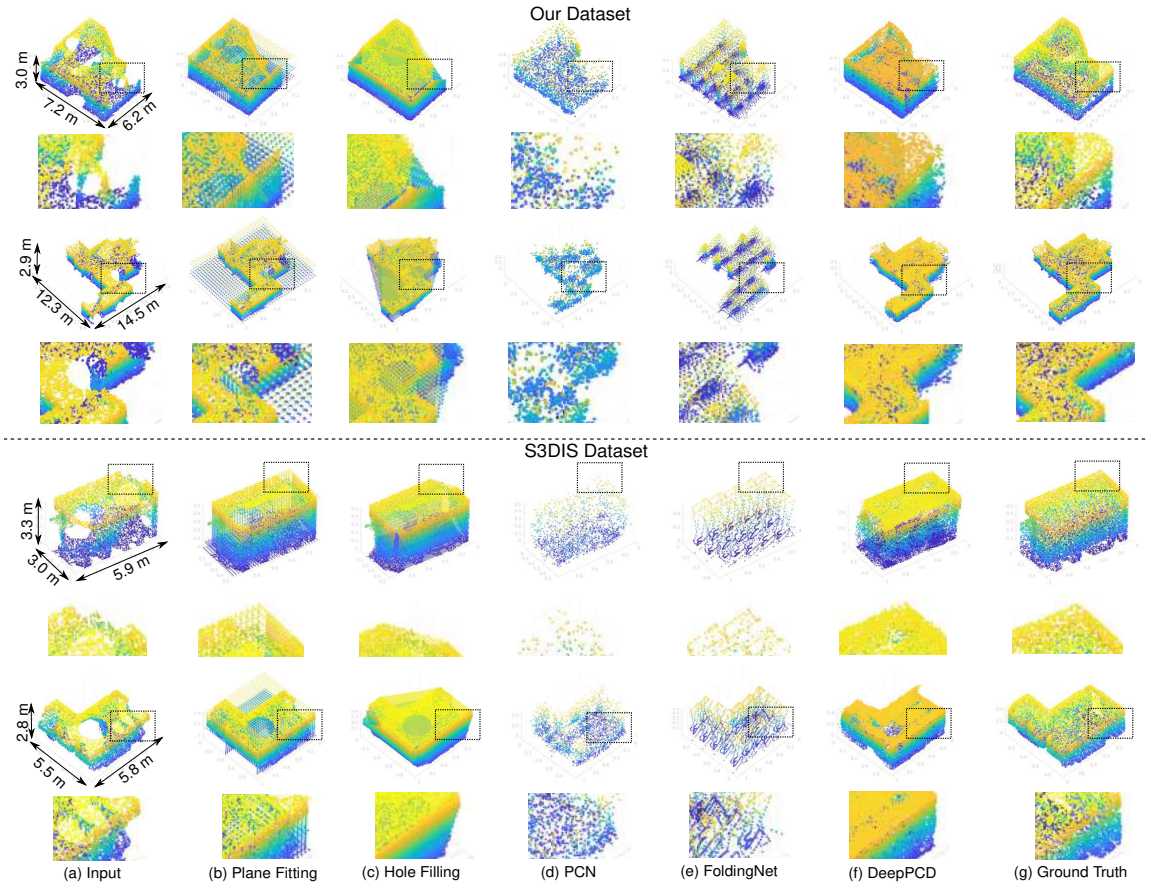

Fig. 6. Geometric structure reconstruction results.

the ground truth. We also compare *DeepPCD* with four existing methods: Plane Fitting [49], Hole Filling [50], PCN [18], and FoldingNet [23]. The Plane Fitting method first detects plane by RANSAC algorithm [53] and once a plane is found, points are assigned by uniformly filling in gaps in the plane. The Hole Filling algorithm first connects the neighboring points to form triangles in the 3D scene, and then, the larger triangles are converted to polygons, which represent the holes in the input scene. After that, we can generate synthetic points to fill the holes. For both these methods, we use their open-sourced code to test their performances. Both the deep learning methods, PCN and FoldingNet, are originally designed for PCD reconstruction of small objects. For a fair comparison, we use their open-sourced code, split each of our large PCD into small patches, train their models with default hyper-parameters, and reconstruct the large PCD by merging the predicted small PCD.

Fig. 5a shows the Cumulative Distribution Function (CDF) of the ChD across all test cases. For our dataset, the median and 90^{th} percentile ChDs of *DeepPCD* are only 0.0004 and 0.0010, respectively, which is tolerable in practice. In contrast, the median ChD for the PCN and FoldingNet are 0.0017 and 0.0009, respectively, and the median ChD for the Plane Fitting and Hole Filling are 0.0012 and 0.0010, respectively. Besides, PCN and FoldingNet are unable to reconstruct many PCD: The shapes are ambiguous, and the point distribution is significantly different than the ground truth (see Fig. 6 for an example). Although the Hole Filling and Plane Fitting can fill the holes in

incomplete PCDs, they lack an understanding of the global shapes and cannot distinguish holes from common shape borders. As a result, they always fill in unnecessary parts (Fig. 6).

For S3DIS dataset too, *DeepPCD* outperforms all base-line algorithms and reconstructs high-quality PCD: The median and 90^{th} percentile ChDs are 0.0005 and 0.0008, respectively (Fig. 5b). So, *DeepPCD* can successfully reconstruct the missing portion of incomplete inputs and is able to preserve the core geometric structure of large objects, such as the walls, floors, and ceilings, of the PCD in most cases. *These results show that DeepPCD's geometric structure reconstruction network is well generalizable for multiple environments, and it reconstructs PCD similar to the ground truth consistently.*

5.2 Color Reconstruction

Next, we evaluate *DeepPCD*'s color reconstruction network on the same test samples. We first statistically evaluate the performance of color reconstruction by estimating the SSIM. For each PCD, we map it into 6 perpendicular cubic faces (*e.g.*, front, back, left, right, *etc.*) to obtain the 2D projected images. Then, we compute the SSIM of these images *w.r.t.* the ground truth projections and average the 6 SSIM values. We estimate the SSIM of the reconstructed color PCD and compare it with the incomplete color PCD. Fig. 7(a-b) show the CDF plots of the SSIM values. For our collected dataset, the median and 90^{th} percentile SSIM for the incomplete PCD are only 0.83 and 0.91, respectively. In contrast, *DeepPCD* significantly improves the color quality of the PCD, and the median and 90^{th} percentile SSIM are 0.89 and 0.95, respectively. For S3DIS dataset also, *DeepPCD* performs very well (Fig. 7b), and the median and 90^{th} percentile SSIM are 0.93 and 0.98, respectively.

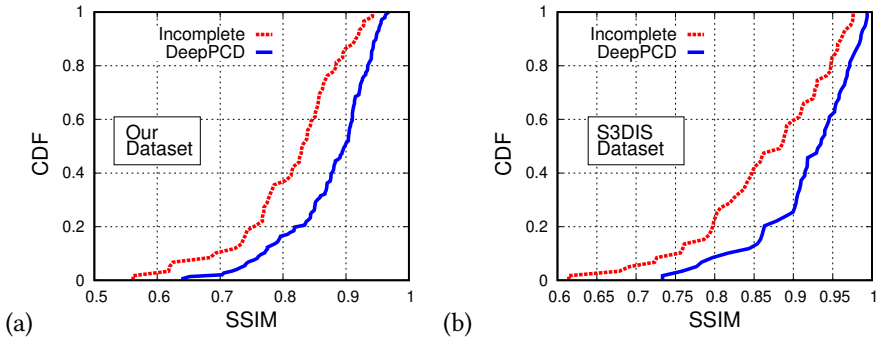

Fig. 7. SSIM distribution for the Incomplete and *DeepPCD* reconstructed PCD across 300 test samples in each dataset: (a) Our dataset; (b) S3DIS dataset.

Fig. 8 shows visual examples of color reconstructed PCD, and we observe that *DeepPCD* can infer similar color as the ground truth in most regions. But occasionally, it could reconstruct slightly worse colors in those regions that already have the color information in the incomplete PCD. This is because we rebuild the entire PCD color from scratch using an encoder-decoder structure. So, this process could introduce some distortion and loss of information of existing colors. In such cases, *DeepPCD* could identify those regions in the reconstructed PCD and replace the predicted colors with the true colors from the incomplete PCD. *These results demonstrate that DeepPCD reconstructs not only the structure but also the color of the PCD similar to the ground truth for multiple, diverse environments consistently.*

5.3 Ablation Study

To further analyze the effectiveness of different components of *DeepPCD*, we perform explicit ablation studies. Specifically, we evaluate the performance of *DeepPCD* for different number of points and different number of

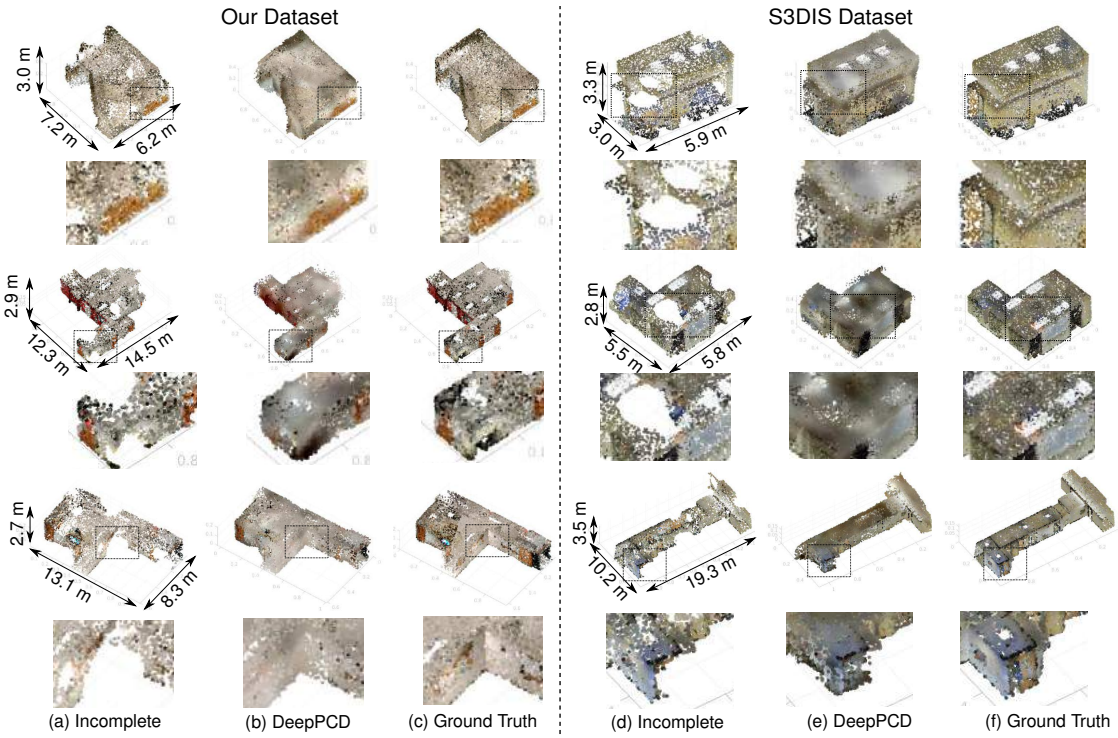

Fig. 8. Color reconstruction results.

patches for ViT, effectiveness of the Plane Point Generator in the geometric structure reconstruction, and effect of different percentages of incompleteness in PCD on both geometric and color reconstructions.

5.3.1 Performance Under Different Number of Points and Different Number of Patches. Recall that we downsample the PCD into a fixed number of points before training. Intuitively, with more points per cubic space of the PCD, the ChD would be smaller because the network have more references and features for reconstruction. On the other hand, the training time will increase with more points. So, to understand the effects of number of points on *DeepPCD*, we conduct ablation study by varying the number of points for each PCD from 5000 to 25000 at a step of 5000 points. Table 8 shows the result, and we see that the trend of ChD follows our intuition. Besides, with more points, the training time also increases from approximately 411 min to 600 min. Throughout the rest of the evaluations, we have used 20000 points for each PCD to balance between accuracy and training time.

Table 8. Performance of *DeepPCD* with different number of points.

Number of Points	5000	10000	15000	20000	25000
ChD (median)	6.65×10^{-4}	4.83×10^{-4}	4.15×10^{-4}	3.88×10^{-4}	3.77×10^{-4}
ChD (90^{th} %-ile)	17.16×10^{-4}	12.83×10^{-4}	11.4×10^{-4}	10.77×10^{-4}	10.16×10^{-4}
Training Time	411 min	445 min	495 min	543 min	600 min

Additionally, the number of patches may also affect the performance of *DeepPCD* since more number of smaller patches allows generating denser points with better local details. But when the patch grids is too dense, the number of input points in each grid will be smaller and the feature extractor may not extract sufficient structure

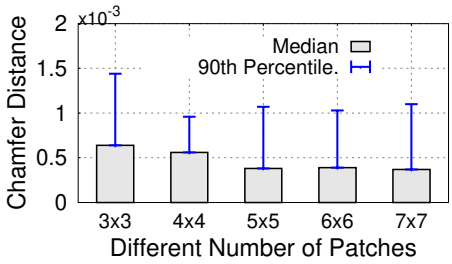

Fig. 9. Performance of *DeepPCD* with different number of patch grids.

information form it, which may cause a performance degradation. To understand the effect of grid numbers, we conduct another ablation study. Specifically, we train and test our network under different grid numbers from 3×3 to 7×7, and Fig. 9 shows the result. We see that when the number of patch grids increases from 3×3 to 5×5, the median ChD reduces steadily, but beyond 5×5, the benfit diminishes. Besides when the number of patches is too large, the training time increases too. So, throughout the rest of the evaluations, we have selected the number of patches to be 5×5 to balance the time and accuracy.

5.3.2 Effectiveness of Plane Point Generator. Recall that the Plane Point Generator populates the initial points on planes to facilitate the reconstruction process. To evaluate its effectiveness, we remove this generator and substitute it with 2D grid points on the X-O-Y plane, similar to [23]. Specifically, for each patch in the PCD, we first generate a 2D grid with 1024 points and then feed them directly into our patch and global reconstruction modules to generate the complete PCD. For a fair comparison, we train this ablated network (*i.e.*, without Plane Point Generator) using the same network parameter settings and same datasets as *DeepPCD*.

Table 9. Performance of *DeepPCD*'s geometric structure reconstruction with and without the Plane Point Generator.

DeepPCD	ChD (median)	ChD (90^{th} %-ile)
Without	4.3×10^{-4}	11.51×10^{-4}
With	3.88×10^{-4}	10.77×10^{-4}

Table 9 shows the median and 90^{th} percentile ChD with and without the Plane Point Generator. With the generator, the median error reduces from 4.3×10^{-4} to 3.88×10^{-4}, and the 90^{th} percentile error reduces from 11.51×10^{-4} to 10.77×10^{-4}. Fig. 10 shows visual examples of a PCD with and without the generator and compares the results with the incomplete and ground truth PCD. We observe that the points reconstructed without the generator are more likely to be unevenly distributed, and the walls and floors will be distorted. This is because without the Plane Point Generator the ChD loss alone cannot guarantee that the points will be located on a plane. Hence, without the good initial points, *DeepPCD* will have difficulty in converging to the optimal network parameters for the PCD reconstruction. So, the *Plane Point Generator populates well-distributed initial points to facilitate structure reconstruction accurately and consistently.*

5.3.3 Effect of Different Percentages of Incompleteness. We then evaluate *DeepPCD* under various types of incompleteness of the input PCD. We follow the method described in Section 4 to control the total number of points inside a PCD by generating small holes inside it to mimic the PCD incompletion of real data collection. Table 10 shows the results. The median and 90^{th} percentile ChD loss under 50% of incompleteness (*i.e.*, 50% of points removed from the ground truth PCD) could be up to 10.1×10^{-4} and 25.7×10^{-4}, respectively. But these losses reduce quickly when the percentage of incompleteness decreases, *i.e.*, when incomplete PCD includes more points. But a larger number of points do not always improve *DeepPCD*'s performance: There is hardly 1.6×10^{-4}

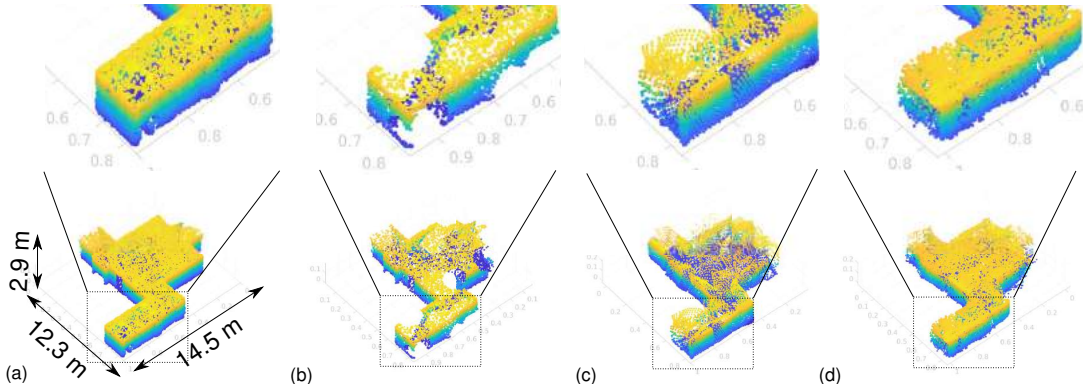

Fig. 10. (a–b) Ground truth and incomplete PCD. Geometric reconstructions: (c) without Plane Point Generator. (d) with.

Table 10. Performance of *DeepPCD* under different percentages of PCD incompleteness.

Incompleteness	ChD (median)	ChD (90^{th} %-ile)	SSIM (median)	SSIM (90^{th} %-ile)
50%	10.1×10^{-4}	25.7×10^{-4}	0.81	0.89
30%	3.9×10^{-4}	10.8×10^{-4}	0.89	0.95
10%	2.6×10^{-4}	7.1×10^{-4}	0.93	0.97

ChD improvement for an additional 20% points. Similarly, for the SSIM in color PCD, we notice the same trend as ChD. Under 50% of incompleteness, the median and 90^{th} percentile SSIM is 0.81 and 0.89, respectively, and they improve quickly to 0.89 and 0.95, respectively, with additional 20% points. Fig. 11 shows example visual results under different percentages of incompleteness. We notice that the top portion of PCD in Fig. 11d is completely missing, and under such circumstances, *DeepPCD* could not reconstruct that region. This is intuitive since, without any structural clues of a missing region, *DeepPCD* would unlikely be able to reconstruct it. In such cases, *DeepPCD* could mark the region in the reconstructed PCD and guide the user to briefly re-scan the area. We leave this real-time guidance system design as future work.

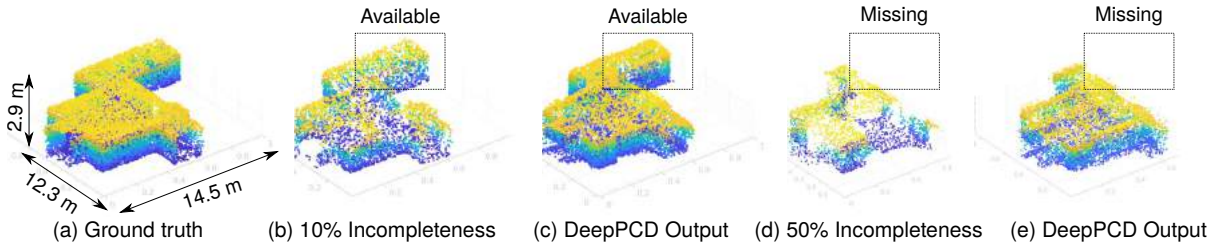

Fig. 11. (a) Ground truth PCD. Geometric reconstructions under different percentages of PCD incompleteness. (b) 10% incompleteness in PCD. (c) Reconstruction results of (b). (d) 50% incompleteness in PCD. (e) Reconstruction result of (d).

5.3.4 Evaluation on Real Incomplete PCDs. So far, we have seen that *DeepPCD* can reconstruct PCD with good structures and colors for synthetic incomplete PCD. To understand its performance in realistic conditions, we train the *DeepPCD* model with synthetic incomplete PCD, and test it on real incomplete PCD. To this end, we place the smartphone on an office cart to simulate a robot movement and measure the PCD for an average 4 min in various environments (ground truth PCD is measured with 15 min scan time). Fig. 12 shows the PCD and reconstruction results. Due to the limited scan time and range and arbitrary scan trajectory, the collected PCD is naturally incomplete. But *DeepPCD* can successfully reconstruct both the structure and color of the missing

regions. *These results show that DeepPCD adapts very well in different environments without requiring the user to spend a lot of time scanning and collecting the entire indoor PCD.*

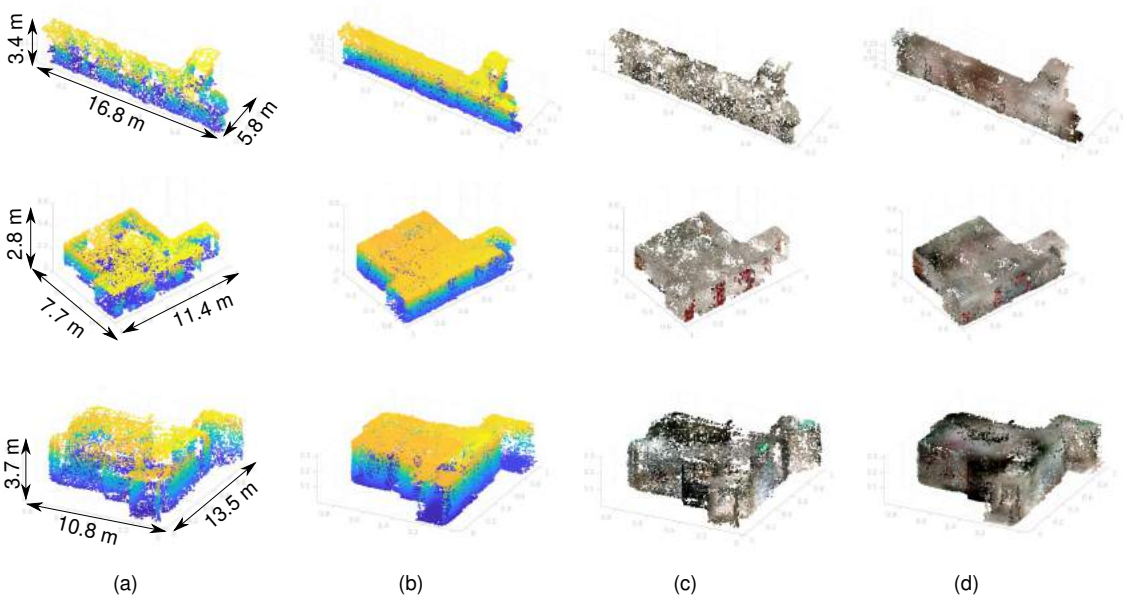

Fig. 12. Reconstruction results for real incomplete PCD: (a) Measured structure of incomplete PCD. (b) *DeepPCD*'s structure reconstruction. (c) Measured color of incomplete PCD. (d) *DeepPCD*'s color reconstruction.

Table 11. Inference time from *DeepPCD*.

Module	Model Size	Inference Time
Geometric	30.9 MB	0.028 s
Color	34.7 MB	0.18 s

5.3.5 Run-time Complexity of DeepPCD. For mobile and ubiquitous computing environments, the ability to quickly generate complete PCD is important and can improve the user experience. It is possible to build a GPU server where a user could upload the collected incomplete PCD and download the complete PCD quickly. So, to understand the run-time performance of *DeepPCD*, we evaluate its inference time on the RTX A6000 GPU server. Table 11 shows that the inference time for the geometric and color reconstruction networks are quite fast, on average, 0.028 s and 0.18 s, respectively. The size of trained models is also quite small, less than 35 MB; so, a mobile platform with a powerful GPU could download the model and run it locally. *So, DeepPCD is suitable for mobile platforms and can respond to a user in near real-time.*

5.4 Applications Result

So far, we have evaluated *DeepPCD*'s ability to reconstruct high-quality PCD. We now showcase the benefits of such reconstructions for three applications: (1) Device localization; (2) Device navigation; and (3) Object tagging and retrieval. Each application runs in different indoor environments, and we evaluate and compare their performance using incomplete PCD with 30% of incompleteness, reconstructed PCD, and ground truth PCD.

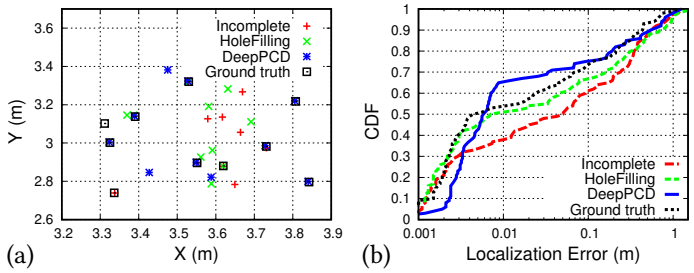

Fig. 13. (a) Examples of estimated locations. (b) CDF of localization errors in all test environments.

5.4.1 Device Localization. Accurate indoor localization enables many ubiquitous sensing applications and has been an active area of research in the past decade. Since GPS is typically inaccurate indoors, researchers have explored IMUs, such as accelerometers, gyroscopes, magnetometers, wireless signals, and 3D scenes, *etc.*, to estimate device position. Among them, visual data based device localization remains the most popular and widely used technique [10]. Since the device's localization accuracy depends on its ability to reconstruct an accurate 3D scene or point cloud [10], a high-quality PCD could bring significant improvement in localization performance. So, to test the efficacy of *DeepPCD*, we employ the existing vision-based localization technique using the Normal-Distributions Transform (NDT) [29]. In the NDT based localization, we can identify the location of a device based on the scene changes between itself and a reference location. Since we would like to evaluate the individual location estimation performance, we consider the center of the PCD to be the reference position, and its location and scene to be accurately known *a-priori*. Then, we randomly select 10 different locations, run the NDT based localization, and estimate the Euclidean distance, *i.e.*, error, between the predicted and ground truth locations. For the predicted location, we compare the localization accuracy among *DeepPCD*, Hole Filling, incomplete and ground truth PCDs.

Fig. 13a visually shows the localization result in a 2D plane for one environment. We observe that, compared to the incomplete PCD, the predicted locations using *DeepPCD*'s reconstructed PCD mostly match with the estimated location using the ground truth PCD. Fig. 13b shows the CDF of localization errors across all test cases. The median and 90^{th} percentile location errors using *DeepPCD*'s reconstructed PCD are 0.006 m and 0.54 m, respectively, which are very close to the ground truth PCD based localization. However, using the reconstructed PCD by Hole Filling, the median and 90^{th} percentile location errors increase to 0.007 m and 0.70 m, respectively. Next, Table 12 shows the localization error for different approaches under varying incompleteness level. Under all conditions, we observe that *DeepPCD*'s performance is relatively stable and close to the ground truth. These results indicate that *DeepPCD's reconstructed PCD is quite similar to ground truth and can achieve close to the ground truth localization performance.*

Table 12. Average localization errors under different percentages of PCD incompleteness.

Incompleteness	Incomplete	Hole Filling	*DeepPCD*	Ground Truth
50%	0.166	0.149	0.127	0.122
30%	0.194	0.193	0.137	0.122
10%	0.162	0.154	0.124	0.122

5.4.2 Device Navigation. We now evaluate the performance of *DeepPCD* for aiding vision-based device navigation in indoor environments. Different from the localization above, navigation involves continuous device movement and estimation based on the prior location in device's path. We leverage the existing NDT based tracking algorithm [29] and compare the performance of *DeepPCD w.r.t.* the incomplete and ground truth PCD. The tracking algorithm is similar to the previous localization, but the key difference is that instead of using a fixed, known reference location, it uses the previously estimated location as the reference. To this end, we first manually generate a

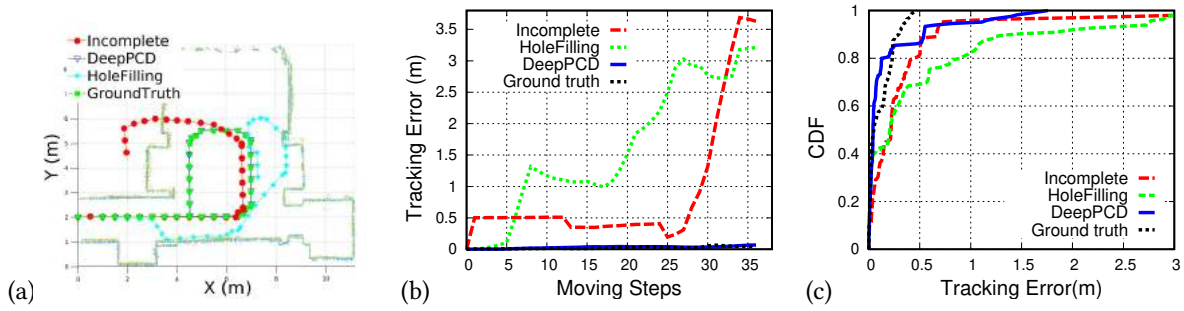

Fig. 14. (a) Tracking results in one test environment. (b) Tracking error (m) with moving steps. (c) CDF of tracking error.

device's moving path given a fixed field of view inside the indoor environment. Then, for each point in this path, we find the visible PCD of the device and treat it as the scanned PCD. Finally, we feed this continuously scanned PCD into the NDT tracking algorithm to estimate the moving path. Fig. 14(a-b) illustrate an example tracking result on a 2D plane and the accumulated error with movement steps. The tracking result with incomplete PCD is distorted, but *DeepPCD* can follow the ground truth closely. Furthermore, we examine the tracking performance across 10 different environments, and Fig. 14c shows the CDF plot of error using PCD from incomplete, ground truth, and *DeepPCD*. We see that the median and 90^{th} percentile tracking errors using incomplete PCD are 0.21 m and 0.66 m, respectively. In contrast, using the reconstructed PCD from *DeepPCD*, the median tracking error closely match with the ground PCD based tracking and decreases to 0.04 m only. Similarly, Table 13 also shows that *DeepPCD* has similar tracking performance as the ground truth PCD even under different incompleteness. *These results demonstrate that the tracking errors do not accumulate significantly when devices use DeepPCD's reconstructed PCD.*

Table 13. Average tracking errors under different percentages of PCD incompleteness.

Incompleteness	Incomplete	Hole Filling	*DeepPCD*	Ground Truth
50%	0.17	0.50	0.16	0.11
30%	0.34	0.52	0.17	0.11
10%	0.28	0.32	0.07	0.11

5.4.3 Object Tagging and Retrieval. Besides localization and navigation, applications such as AR/VR require the capability to tag and retrieve 3D objects. Unlike tracking and localization, which only use PCD shape and position information, the accuracy of object tagging and retrieval depends on the detailed geometric and color information of the input PCD. To test how effectively *DeepPCD* improves these applications' ability to tag and retrieve objects in 3D scenes, we run a template matching algorithm that matches objects in reconstructed PCD with the ground truth PCD. Specifically, we first manually tag meaningful objects, such as doors, windows, tables, chairs, *etc.*, inside each ground truth PCD and store their relative object bounding boxes. Next, during testing, we use these bounding box's coordinates as a reference and find the same regions within the reconstructed PCD. Then, we calculate the SSIM of reconstructed objects in these regions compared to ground truth objects. If the SSIM is larger than a threshold (0.9), then we assume that *DeepPCD* successfully reconstructs this object, and we treat it as one detected object. Similarly, we re-run this matching algorithm with the incomplete PCD. Fig. 15a shows one example of a reconstructed door in PCD. Compared to incomplete ones, the reconstructed object contains more geometric information. Furthermore, to count how many objects we can reconstruct compared to the incomplete PCD, we re-run the same process on the ground truth and incomplete PCD. Fig. 15b shows the failure rate in detecting objects with *DeepPCD* and incomplete PCD *w.r.t.* the ground truth. We detect 170 and

191 total objects in all the incomplete and reconstructed PCD, respectively, and we have 207 total objects in the ground truth. Thus, *DeepPCD* reduces the failure rate in detection from 17.3% to 7.3%.

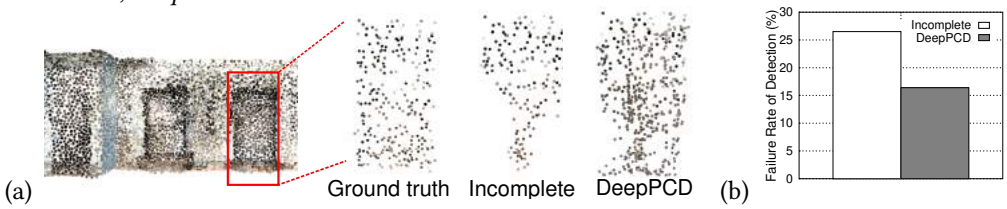

Fig. 15. (a) A door in the ground truth, incomplete, reconstructed PCD. (b) % of failures in detecting objects across all test environments.

In sum, these results demonstrate that DeepPCD is generalizable across diverse environments and improves the ubiquitous applications' performance by reconstructing PCD similar to the ground truth.

6 RELATED WORK

6.1 3D Shape Completion

Existing approaches for 3D shape completion can be divided into three categories: Geometry-based, alignment-based, and learning-based. Geometry-based approaches infer the missing regions directly from the observed regions by extracting geometric clues. For example, [54] introduces a formal method for discovering regular or repeated geometrical structures in 3D shapes for shape repair. Authors in [55] assume most objects in a target scene are symmetrical and present an algorithm to identify the symmetric structures in the incomplete shapes for plausible scene completions. However, such an assumption may not always hold true for complex objects in indoor environments. Alignment-based approaches, instead of generating the missing part directly, build large shape databases and fill the incomplete input by searching the most similar shape from the database [56, 57]. However, constructing such a database for real indoor environments consisting of many objects could be challenging. Learning-based approaches train a model to predict the complete shape from the partial input and have been shown to outperform both the geometry or alignment-based approaches [18, 21, 58].

Based on their network structures, we can further divide the learning-based approaches into two main categories: Convolution-based and Graph-based. Convolution-based method, such as [17], first voxelizes the PCD into 3D voxels and then uses 3D CNN to process it, learn features, and reconstruct better quality PCD. Although 3D CNN captures coarse geometric features between voxels, a lot of fine geometrical information could be lost during the voxelization process. Decreasing the size of voxels and increasing the number of total voxels could preserve more geometric information, but the process is more computational and memory intensive. Graph-based method, such as [18], solves these challenges by building a graph neural network and combining MLP to process and extract features directly from the PCD. But graph neural network relies on the edge graph to explore the relationship among points, which is built upon the Euclidean distance between points and cannot capture accurate geometric features like 3D CNN kernels. Recently, GRNet combines these two methods together and introduces 3D grids as intermediate representations to regularize unordered PCD [19]. To further improve the performance of shape reconstruction, several researchers use prior knowledge to guide the network. For example, authors in [59] proposed a shape prior learning method for 3D objects to improve the performance of shape reconstruction. Besides, authors in [58] addressed this task by introducing an extra single-view image to guide the point cloud completion. Although these methods can reconstruct good shapes, they are designed for reconstructing small objects and may not work well with PCD of large indoor environments [60]. In contrast, *DeepPCD* explores and uses the property of indoor environments and proposes a patch based method for PCD reconstruction of large indoor environments accurately and consistently.

6.2 Indoor PCD Completion

The completeness of the indoor scene is critical for many ubiquitous sensing applications. Recently, several works have started exploring the indoor scene completion task. For example, authors in [60] introduce a novel mesh-based completion approach where it takes an incomplete scan as input and predicts a complete mesh along with semantic labels. However, their method is designed for 3D meshes and can not be applied for indoor PCD. Authors in [16] propose a traditional method for PCD completion using a surface connectivity relation inference. Missing points are completed by estimating the connectivity relations between pairs of the surfaces and by filling individual planar surfaces. Although they show that their method is good at estimating the missing points in a small region, it is difficult to reconstruct the large missing area of PCD. Besides geometric structures, reconstructing color information is also important for PCD completion. Authors in [38] propose a point cloud colorization network based on a GAN architecture, which directly takes a PCD as input and outputs color information for each point. Authors in [39] extend the GAN and use a conditional GAN (cGAN) for 3D PCD colorization. They not only use a point Discriminator to directly judge the color of each point but also render the estimated colored PCD into a 2D image. Different from these existing approaches, *DeepPCD* aims to reconstruct the missing colors of incomplete PCD with measured incomplete colors as the prior information to guide the color reconstruction process.

7 DISCUSSION AND FUTURE WORKS

Although, *DeepPCD* can reconstruct the structure and color of a majority of the PCD, we observe for a few cases, the location and color of points could be distorted. We believe this could be due to the use of encoder-decoder architecture which highly compresses and decompresses the abstract features, and the use of disjoint geometric and color reconstruction networks. One approach to solve this problem is to jointly train both the networks with a closed-loop feedback between them so that the performance of each reconstruction could be better. Designing this large network is non-trivial due to the tuning requirement of a large number of parameters and hyper-parameters. We leave this study as future work. Besides, for the geometric reconstruction, we have used a static number of patches to split each PCD, but we believe it would be feasible to split the PCD adaptively depending on the level of homogeneity in an indoor environment. We will also explore this adaptive splitting technique in our future work. Currently, we have designed and implemented *DeepPCD* in a stand-alone system to mainly demonstrate its feasibility and effectiveness. The inference time is fast, on average, it takes 0.028 s and 0.18 s for geometric and color reconstructions, respectively. In the future, such inference can run directly on a GPU server, and a user could upload the incomplete PCD to the server and obtain the reconstruction results in near real-time. Finally, we also plan to customize *DeepPCD* by trimming different layers so that it can be potentially trained on mobile platforms directly with deep-learning frameworks, such as TensorFlow Lite [61] or PyTorch Mobile [62].

8 CONCLUSION

We propose *DeepPCD*, a deep-learning-based system that can reconstruct the missing geometric and color information of large indoor incomplete PCD. *DeepPCD* designs customized graph neural networks and vision transformer to extract the local and global features, and proposes a plane prior to improve the structural quality of the PCD. It further designs a customized cGAN based network to extract color information from incomplete PCD and uses them to predict the color of structurally complete PCD. Experimental results validate the effectiveness of *DeepPCD*, qualitatively and quantitatively, in two diverse datasets, spanning 50 indoor environments, and show its efficacy in improving the performance of vision-based ubiquitous sensing applications.

ACKNOWLEDGMENTS

We sincerely thank the reviewers and the editors for their comments and feedback. This work is partially supported by the NSF under grant CNS-1910853 and CAREER-2144505.

REFERENCES

[1] B.-S. Kim, P. Kohli, and S. Savarese, "3D Scene Understanding by Voxel-CRF," in *IEEE International Conference on Computer Vision (ICCV)*, 2013.

[2] A. Vincent, "A 3D Perception System for the Mobile Robot Hilare," in *IEEE International Conference on Robotics and Automation*, 1986.

[3] T. Vieville, E. Clergue, R. Enciso, and H. Mathieu, "Experimenting with 3D Vision on a Robotic Head," *Robotics and Autonomous Systems*, vol. 14, no. 1, 1995.

[4] H. Zhang, G. Wang, Z. Lei, and J.-N. Hwang, "Eye in the Sky: Drone-Based Object Tracking and 3D Localization," in *ACM International Conference on Multimedia*, 2019.

[5] Y. Zeng, Y. Hu, S. Liu, J. Ye, Y. Han, X. Li, and N. Sun, "RT3D: Real-Time 3-D Vehicle Detection in LiDAR Point Cloud for Autonomous Driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, 2018.

[6] K. C. Kwan and H. Fu, "Mobi3DSketch: 3D Sketching in Mobile AR," in *ACM CHI*, 2019.

[7] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," in *IEEE/CVF CVPR*, 2018.

[8] C. Gotsman, X. Gu, and A. Sheffer, "Fundamentals of Spherical Parameterization for 3D Meshes," in *ACM SIGGRAPH*, 2003.

[9] L. Linsen, "Point Cloud Representation," Karlsruhe Institute of Technology, Germany, Tech. Rep., 2001.

[10] X. Li, S. Du, G. Li, and H. Li, "Integrate Point-Cloud Segmentation with 3D LiDAR Scan-Matching for Mobile Robot Localization and Mapping," *MDPI Sensors*, vol. 20, no. 1, 2020.

[11] R. Radkowski, "Object Tracking with a Range Camera for Augmented Reality Assembly Assistance," *Journal of Computing and Information Science in Engineering*, vol. 16, no. 1, 2016.

[12] Q. Wang and M.-K. Kim, "Applications of 3D Point Cloud Data in the Construction Industry: A Fifteen-Year Review from 2004 to 2018," *Advanced Engineering Informatics*, vol. 39, 2019.

[13] S. B. Walsh, D. J. Borello, B. Guldur, and J. F. Hajjar, "Data Processing of Point Clouds for Object Detection for Structural Engineering Applications," *Computer-Aided Civil and Infrastructure Engineering*, vol. 28, no. 7, 2013.

[14] D. A. White, "LIDAR, Point Clouds, and Their Archaeological Applications," in *Mapping Archaeological Landscapes from Space*, 2013.

[15] M. Labbe and F. Michaud, "RTAB-Map as an Open-Source Lidar and Visual Simultaneous Localization and Mapping Library for Large-Scale and Long-Term Online Operation," *Journal of Field Robotics*, vol. 36, no. 2, 2019.

[16] Y. Xiao and Y. Taguchi and V. R. Kamat, "Coupling Point Cloud Completion and Surface Connectivity Relation Inference for 3D Modeling of Indoor Building Environments," *Journal of Computing in Civil Engineering*, vol. 32, no. 5, 2018.

[17] A. Dai, C. Ruizhongtai Q., and M. Niessner, "Shape Completion using 3D-Encoder-Predictor CNNs and Shape Synthesis," in *IEEE/CVF CVPR*, 2017.

[18] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert, "PCN: Point Completion Network," in *International Conference on 3D Vision (3DV)*, 2018.

[19] H. Xie, H. Yao, S. Zhou, J. Mao, S. Zhang, and W. Sun, "GRNet: Gridding Residual Network for Dense Point Cloud Completion," in *European Conference on Computer Vision: Computer Vision – ECCV*, 2020.

[20] M. Liu, L. Sheng, S. Yang, J. Shao, and S.-M. Hu, "Morphing and Sampling Network for Dense Point Cloud Completion," *AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, 2020.

[21] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," https://shapenet.org/, 2015.

[22] L. P. Tchapmi, V. Kosaraju, H. Rezatofighi, I. Reid, and S. Savarese, "TopNet: Structural Point Cloud Decoder," in *IEEE/CVF CVPR*, 2019.

[23] Y. Yang, C. Feng, Y. Shen, and D. Tian, "FoldingNet: Point Cloud Auto-Encoder via Deep Grid Deformation," in *IEEE/CVF CVPR*, 2018.

[24] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[25] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *International Conference on Learning Representations (ICLR)*, 2021.

[26] H. Fan, H. Su, and L. Guibas, "A Point Set Generation Network for 3D Object Reconstruction from a Single Image," in *IEEE/CVF CVPR*, 2017.

[27] P. Xiang, X. Wen, Y.-S. Liu, Y.-P. Cao, P. Wan, W. Zheng, and Z. Han, "SnowflakeNet: Point Cloud Completion by Snowflake Point Deconvolution with Skip-Transformer," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2021.

[28] S. Sur, "DeepPCD Project," 2022. [Online]. Available: https://syrex.cse.sc.edu/research/ubiquitous-sensing/deeppcd/

[29] H. Sobreira, C. M. Costa, I. Sousa, L. Rocha, J. Lima, P. C. Farias, P. Costa, and A. P. Moreira, "Map-matching algorithms for robot self-localization: A comparison between perfect match, iterative closest point and normal distributions transform," *J. Intell. Robotics Syst.*, vol. 93, 2019.

[30] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-Voxel CNN for Efficient 3D Deep Learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[31] W. Shi and R. Rajkumar, "Point-GNN: Graph Neural Network for 3D Object Detection in a Point Cloud," in *IEEE/CVF CVPR*, 2020.
[32] S. I. Kabanikhin, "Definitions and Examples of Inverse and Ill-posed Problems," *Journal of Inverse and Ill-posed Problems*, vol. 16, no. 4, 2008.
[33] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, 1998.
[34] Y. Liu, B. Fan, S. Xiang, and C. Pan, "Relation-Shape Convolutional Neural Network for Point Cloud Analysis," in *IEEE/CVF CVPR*, 2019.
[35] S. Ji, W. Xu, M. Yang, and K. Yu, "3D Convolutional Neural Networks for Human Action Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, 2013.
[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All You Need," in *Advances in Neural Information Processing Systems (NIPS)*, 2017.
[37] X. Cao and K. Nagao, "Point Cloud Colorization Based on Densely Annotated 3D Shape Dataset," in *MMM*, 2019.
[38] J. Liu, S. Dai, and X. Li, "PCCN: Point Cloud Colorization Network," in *IEEE International Conference on Image Processing (ICIP)*, 2019.
[39] T. Shinohara, H. Xiu, and M. Matsuoka, "Point2Color: 3D Point Cloud Colorization Using a Conditional Generative Network and Differentiable Rendering for Airborne LiDAR," in *IEEE/CVF CVPR Workshops*, 2021.
[40] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," 2014. [Online]. Available: https://arxiv.org/abs/1411.1784
[41] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems (NIPS)*, 2014.
[42] AsusTek Computer Inc., "Zenfone AR: Go Beyond Reality," 2021. [Online]. Available: https://www.asus.com/us/Phone/ZenFone-AR-ZS571KL/
[43] IntRoLab, "Real-Time Appearance-Based Mappingy," 2021. [Online]. Available: http://introlab.github.io/rtabmap/
[44] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: an Open-Source Mesh Processing Tool," in *Eurographics Italian Chapter Conference*, 2008.
[45] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese, "3D Semantic Parsing of Large-Scale Indoor Spaces," in *IEEE/CVF CVPR*, 2016.
[46] R. Bridson, "Fast Poisson Disk Sampling in Arbitrary Dimensions," in *ACM SIGGRAPH 2007 Sketches*, 2007.
[47] Open-Source, "PyTroch," 2021. [Online]. Available: https://pytorch.org/
[48] NVIDIA, "RTX A6000," 2021. [Online]. Available: https://www.nvidia.com/en-us/design-visualization/rtx-a6000/
[49] L. Li, F. Yang, H. Zhu, D. Li, Y. Li, and L. Tang, "An Improved RANSAC for 3D Point Cloud Plane Segmentation Based on Normal Distribution Transformation Cells," *Remote Sensing*, vol. 9, no. 5, 2017.
[50] Geodan, "Generate Synthetic Points to Fill Holes in Point Clouds," 2020. [Online]. Available: https://github.com/Geodan/fill-holes-pointcloud
[51] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, 2004.
[52] Q. Yang, H. Chen, Z. Ma, Y. Xu, R. Tang, and J. Sun, "Predicting the perceptual quality of point cloud: A 3d-to-2d projection-based exploration," *IEEE Transactions on Multimedia*, vol. 23, 2021.
[53] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, 1981.
[54] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. J. Guibas, "Discovering Structural Regularity in 3D Geometry," in *ACM SIGGRAPH*, 2008.
[55] P. Speciale, M. R. Oswald, A. Cohen, and M. Pollefeys, "A Symmetry Prior for Convex Variational 3D Reconstruction," in *European Conference on Computer Vision: Computer Vision – ECCV*, 2016.
[56] Y. Li, A. Dai, L. Guibas, and M. NieBner, "Database-Assisted Object Retrieval for Real-Time 3D Reconstruction," *Computer Graphics Forum*, 2015.
[57] V. G. Kim, W. Li, N. J. Mitra, S. Chaudhuri, S. DiVerdi, and T. Funkhouser, "Learning Part-Based Templates from Large Collections of 3D Shapes," *ACM Transactions on Graphics*, vol. 32, no. 70, 2013.
[58] X. Zhang, Y. Feng, S. Li, C. Zou, H. Wan, X. Zhao, Y. Guo, and Y. Gao, "View-Guided Point Cloud Completion," in *IEEE/CVF CVPR*, 2021.
[59] X. Wang, J. Ang, Marcelo H, and G. H. Lee, "Point Cloud Completion by Learning Shape Priors," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.
[60] A. Dai, D. Ritchie, M. Bokeloh, S. Reed, J. Sturm, and M. Niebner, "ScanComplete: Large-Scale Scene Completion and Semantic Segmentation for 3D Scans," in *IEEE/CVF CVPR*, 2018.
[61] Open-Source, "Deploy Machine Learning Models on Mobile and IoT Devices," 2022. [Online]. Available: https://www.tensorflow.org/lite
[62] Open-Source, "PyTorch Mobile," 2022. [Online]. Available: https://pytorch.org/mobile/home/

A APPENDIX

A.1 Additional Reconstruction Results

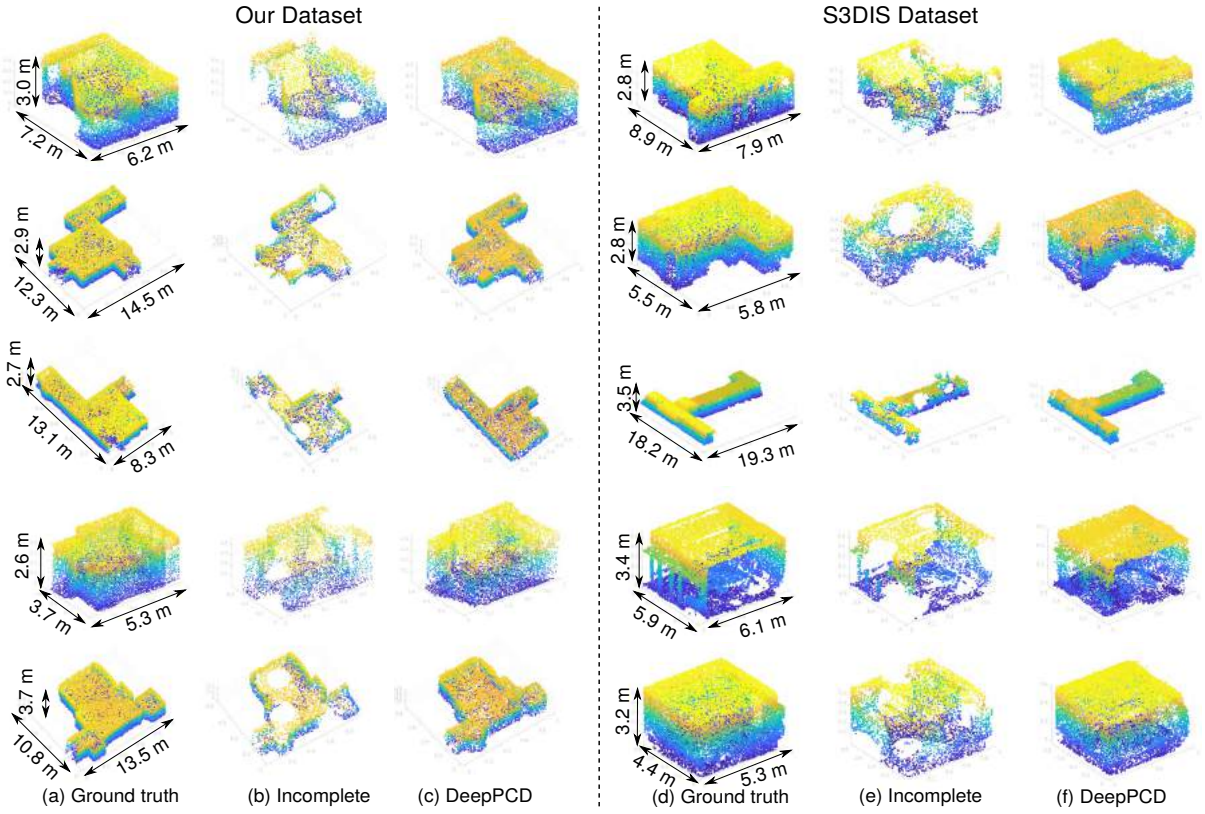

Fig. 16. Additional geometric structure reconstruction results.

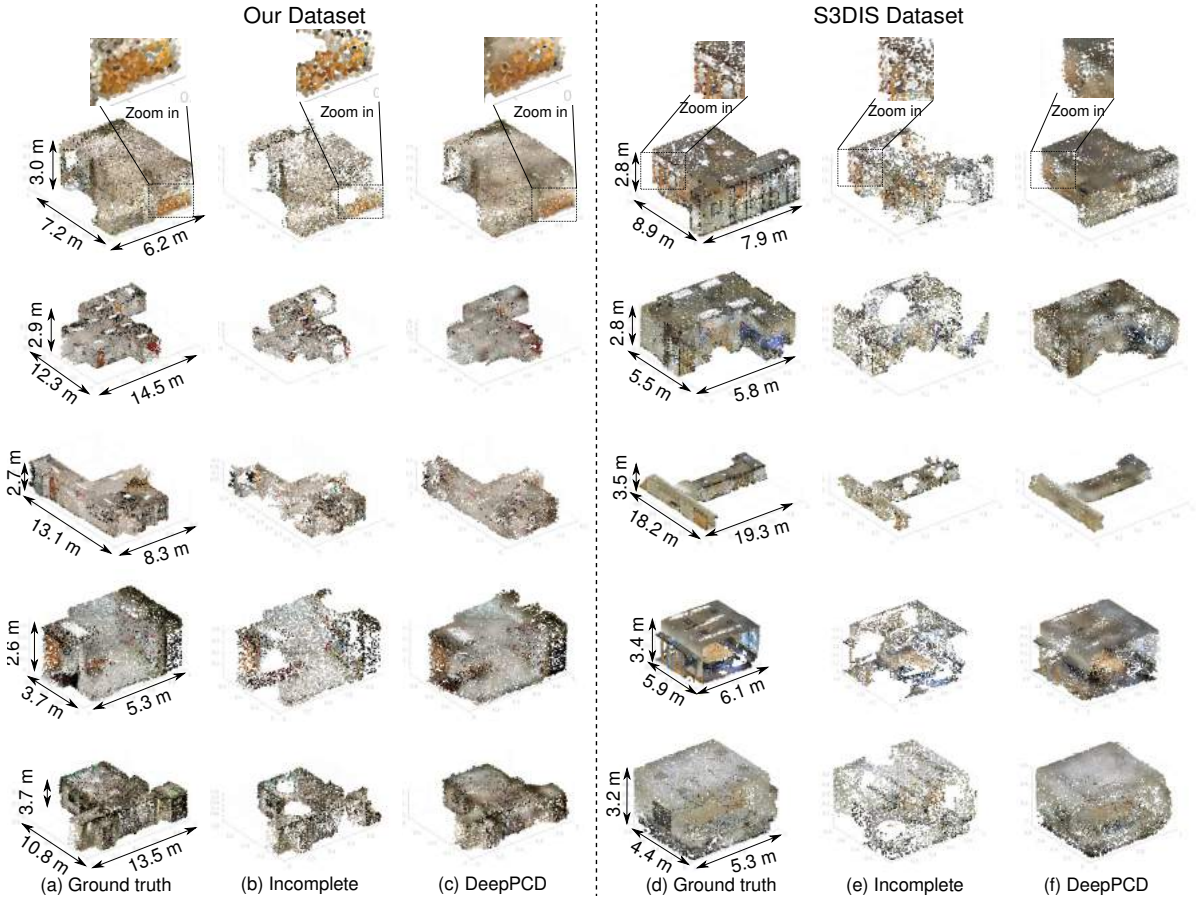

Fig. 17. Additional color reconstruction results.

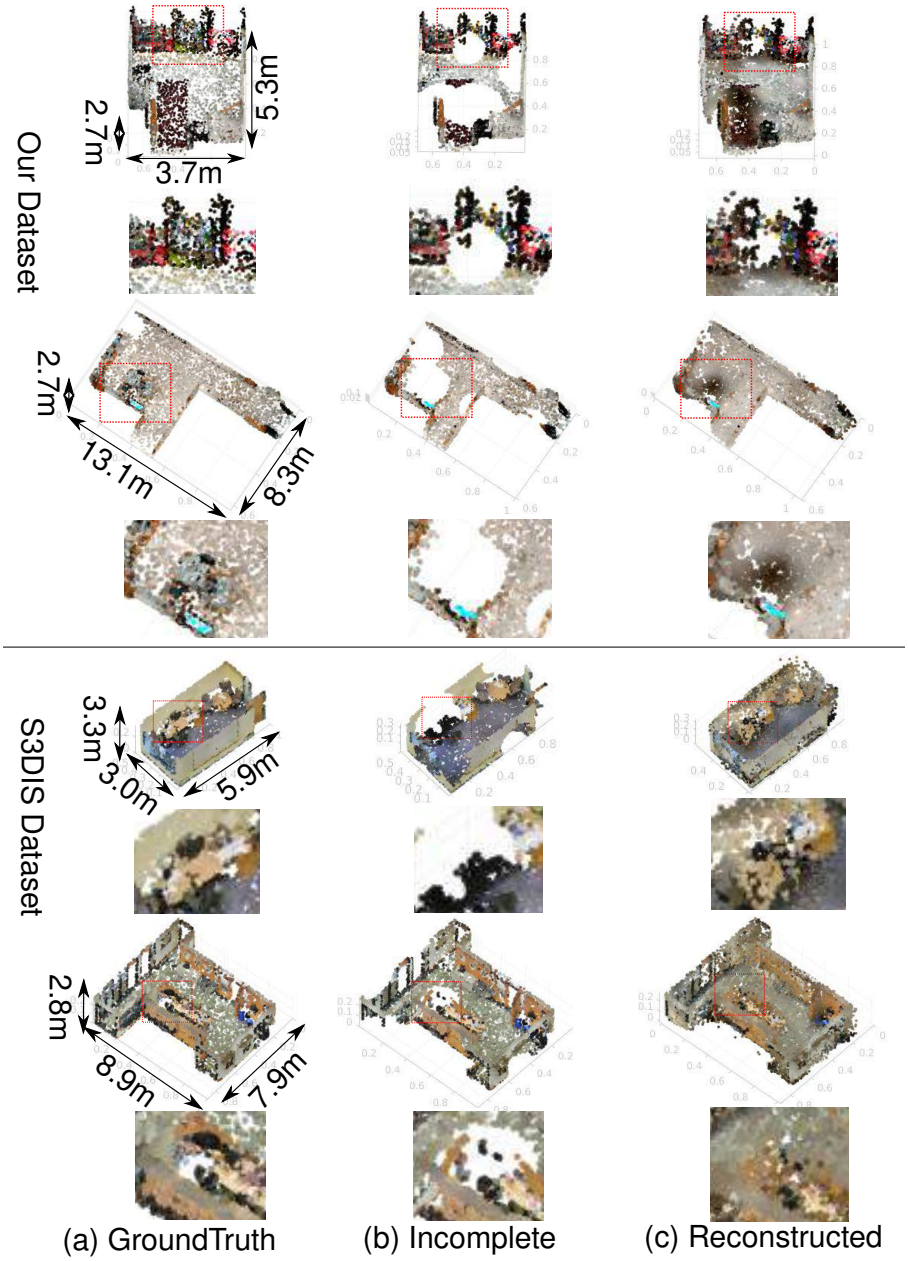

Fig. 18. Color reconstruction results of objects inside PCDs.