Inverting the Pose Forecasting Pipeline with SPF2: Sequential Pointcloud Forecasting for Sequential Pose Forecasting Supplementary Materials

Xinshuo Weng[1], **Jianren Wang**[1], **Sergey Levine**[2], **Kris Kitani**[1], **Nicholas Rhinehart**[2]
[1]Robotics Institute, Carnegie Mellon University
{xinshuow, jianrenw, kkitani}@cs.cmu.edu
[2]Berkeley Artificial Intelligence Research Lab, University of California, Berkeley
{svlevine, nrhinehart}@eecs.berkeley.edu

1 Overview

As it is difficult to include every detail of our work in the main paper, we provide further details here about our new pipeline SPF^2 , the proposed SPF task, the first approach SPFNet, proposed evaluation procedure for the pipeline, and additional experiments to support our main paper.

2 New Perception and Trajectory Forecasting Pipeline: SPF^2

2.1 Fine-tuning of the Object Detector

As generated scene point clouds from SPFNet are not exactly same as the ground truth scene point clouds, using 3D detector pre-trained on the ground truth scene point clouds in our SPF^2 pipeline can lead to inferior performance. To adapt the 3D detector to work with our generated scene point clouds and achieve higher performance, we fine-tune the 3D detector using the scene point clouds generated by our SPFNet along with the 3D bounding box annotations.

3 Our Proposed Task: Sequential Pointcloud Forecasting (SPF)

3.1 Scene Point Cloud v.s. Object Point Cloud

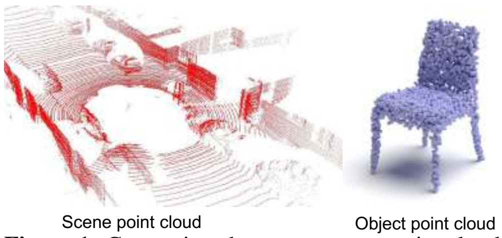

Figure 1: Comparison between a scene point cloud ($100k$ points) and an object point cloud ($1k$ points).

We have defined the scene point cloud in Sec 3.1 of the main paper. Here, we visually compare the scene point cloud with the significantly different object point cloud in prior work. As an example, we show a scene point cloud captured by a LiDAR sensor from the KITTI dataset in Fig. 1 (left) and an object point cloud in ShapeNet [1] in Fig. 1 (right). Obviously, scene point cloud contains much more information than the object point cloud, *e.g.*, many instances of foreground objects such as cars, pedestrians, and background objects such as trees, fences, building, traffic sign that will affect the motion of foreground objects. As a result, it is much more challenging to model and generate scene point clouds than generate the object point cloud. Additionally, it is usually more difficult to process a scene point cloud with about $100k$ points than an object point cloud with about $1k$ points.

3.2 Relation to Object Trajectory Forecasting

In Sec. 2 of the main paper, we have described the differences between prior object trajectory forecasting task and our SPF. Here, we additionally visualize the differences in Fig. 2. Intuitively, the distinction lies in that, object trajectory forecasting aims to forecast future locations of a few points, where each point represents an object of interest, while our proposed SPF task aims to predict future locations of all points in the scene point clouds, including points belong to the foreground objects and scene background.

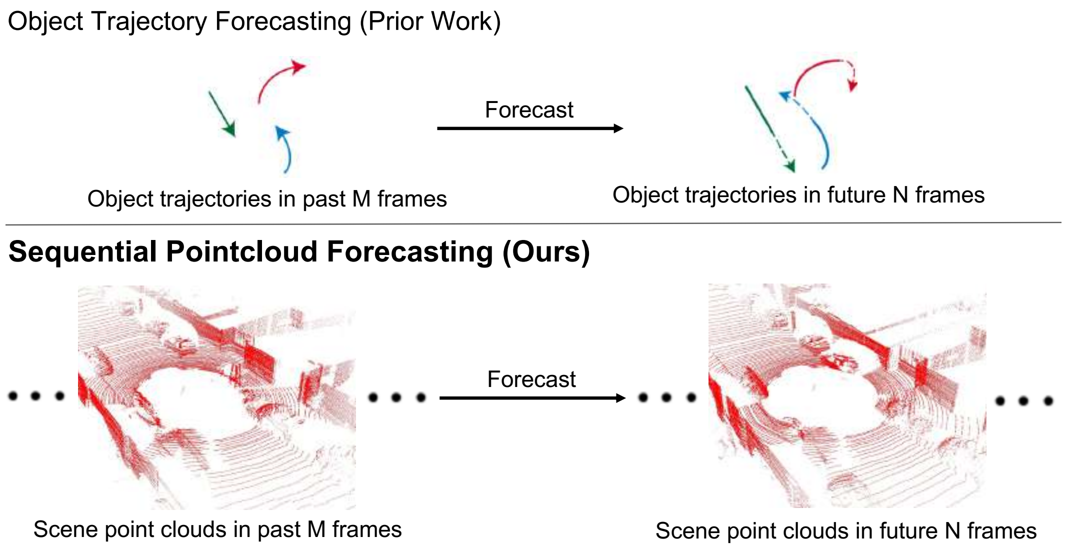
Object Trajectory Forecasting (Prior Work)
Forecast
Object trajectories in past M frames
Object trajectories in future N frames
Sequential Pointcloud Forecasting (Ours)
Forecast
Scene point clouds in past M frames
Scene point clouds in future N frames

Figure 2: (Top) Object Trajectory Forecasting: given object trajectories in the past M frames, the goal is to predict object trajectories in the future N frames. Each colored arrow denotes an object's trajectory (from a top-down view). Solid parts of the arrows denote past trajectories and dotted parts denote future trajectories. **(Bottom) Sequential Pointcloud Forecasting**: given 3D scene point clouds in the past M frames, the goal is to predict a sequence of future scene point clouds.

3.3 Relation to Scene Flow

Although scene flow [2, 3] is not a forecasting task, scene flow task aims to estimate the 3D motion of each point in the scene point cloud, which is also related to our proposed SPF. Here, we clarify the differences between scene flow and our SPF to avoid any confusion. Specifically, the differences are three-fold: (1) training scene flow methods requires ground truth annotation of point-wise motion, which is expensive to obtain. In contrast, our proposed SPF task only requires the scene point clouds (*e.g.*, captured by a LiDAR sensor) as the ground truth for training and does not need to involve human annotation; (2) scene flow aims to learn point-wise motion between only two frames while our SPF aims to learn scene dynamics from a sequence of point clouds (*e.g.*, 30 frames). As a result, SPF is more challenging but can provide more useful information by modeling a history of past frames; (3) scene flow aims to learn motion in the current frame while SPF is a forecasting task and aims to predict points' locations in the future (*e.g.*, upto three seconds).

4 Approach: SPFNet

4.1 Preliminaries about the 2D Range Map Representation

We briefly summarize the process of transforming a scene point cloud to a 2D range map here but refer the readers to [4] for details. Intuitively, range map transformation leverages the ray-casting nature of the LiDAR point cloud and projects the 3D points onto a 2D spherical plane. Specifically, for a 3D point $P = (x, y, z)$ in the scene point cloud with z represents the height, its coordinate in the 2D spherical plane $Q = (\theta, \phi)$ is defined as the following:

$$\theta = \operatorname{atan2}(y, x), \qquad \phi = \arcsin(z/\sqrt{x^2 + y^2 + z^2}), \tag{1}$$

where θ is the azimuth angle and ϕ is the elevation angle from the origin. Then, to convert the continuous 2D spherical plane into a discrete 2D range map with resolution of $H \times W$, we discretize the 2D spherical plane with a fixed number of bins (H bins along elevation axis, W bins along azimuth axis) in a pre-defined range, *e.g.*, $\phi \in [-30°, 10°]$ and $\theta \in [0°, 360°]$. The range is chosen as most points captured by a standard LiDAR sensor (*e.g.*, Velodyne-64) fall into it. As a result, each bin (*i.e.*, an pixel in the range map) will cover the range of $(\frac{40°}{H}, \frac{360°}{W})$ in the spherical plane, and we can easily find the nearest bin for each point $Q = (\theta, \phi)$ as its pixel location. In order to recover the 3D coordinate of a point from its 2D spherical coordinate, knowing the azimuth and elevation angle is not enough as they only provide the direction of the point to the origin in 3D space. Therefore, we use the distance $d = \sqrt{x^2 + y^2 + z^2}$ as the pixel value for each projected point in the 2D range map.

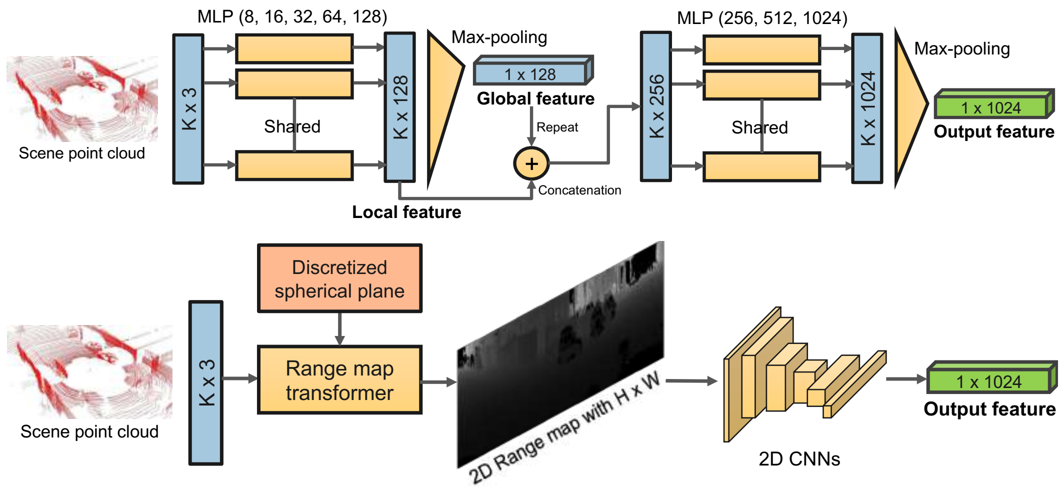

Figure 3: (Top): Point-based encoder. **(Bottom)**: Range map-based encoder. Note that we only show one frame of the scene point cloud to be processed by the encoder in the above figure but our full SPFNet uses encoder with shared weights to process a sequence of scene point clouds.

In the case where multiple 3D points are projected to the same bin in the range map[1], we pick the one with the largest distance d as the pixel value. This decision is made to preserve information in the faraway region of the point cloud, because there are often dense and redundant points in the nearby range but sparse points in the faraway range. For the bins that do not have any projected point, we simply fill their pixel value with zero and will mask them out during training and inference. As a result, the range map with a resolution of $H \times W$ still preserves the 3D information and can be easily converted back to a point cloud. Note that the range map is different from a depth map in that (1) two maps are not in a same 2D coordinate; (2) depth map needs the camera matrix in order to be converted to a point cloud while the range map can be directly converted to a point cloud; (3) depth map usually only covers the points in frontal view while the range map can be defined to have the horizontal field of view of $360°$ degrees.

4.2 Detailed Network Architecture of the Encoder

We have briefly described our two (point-based and range map based) encoders in Sec. 3.2 of the main paper. To help readers better understand our SPFNet, here we additionally illustrate the network architectures of our two encoders in Figure 3 and also provide more details of the neural network architectures below:

Point-based encoder. A scene point cloud with a shape of $K \times 3$ is fed into a shared five-layer multi-layer perceptron (MLP) (3⇒8⇒16⇒32⇒64⇒128) to obtain a local feature with a dimension of 128 for each point. Then a global feature with the dimension of 1×128 is obtained by applying a max-pooling operator to the local features of all points. We then fuse the local and global features by concatenation to obtain a feature with a dimension of 256 for each point. The fused features are then processed by a subsequent three-layer MLP (256⇒256⇒512⇒1024) and a max-pooling operator to obtain the output feature with a dimension of 1024 for each point. As K is very large (*e.g.*, $> 100k$) for every frame, maintaining high-dimensional features for every single point in the scene point cloud will reach the memory limit of the GPUs. As a result, we compress the features by a max-pooling operator to obtain the final output feature with a dimension of 1024 to represent the entire scene point cloud.

Range map-based encoder. An input scene point cloud with a shape of $K \times 3$ along with a discretized spherical plane is fed into the range map transformer (see Sec. 4.1) to obtain the 2D range map with resolution of $H \times W$ (we use 120 x 1024 in our experiments to predict about 122.88k points per frame, similar to the number of points in the scene point cloud captured by a standard Velodyne LiDAR sensor). Then, we use a standard 2D CNNs with eight layers (each layer has convolution + batch normalization + ReLU) to extract the output feature from the 2D range map. Specifically,

[1]Note that this situation happens very rarely in our experiments because we use a high-resolution range map so that there is no downsampling effect during the transformation of a point cloud to a 2D range map.

the eight layers in the 2D CNNs convert the input range map from a shape of $1 \times 120 \times 1024$, to $4 \times 60 \times 512$, to $8 \times 60 \times 256$, to $16 \times 60 \times 128$, to $32 \times 30 \times 64$, to $64 \times 15 \times 32$, to $128 \times 8 \times 16$, to $256 \times 4 \times 8$, to $512 \times 2 \times 4$. Then, a final convolution layer (without batch norm and ReLU) is applied to obtain the final output feature with a shape of 1024×1, to make our SPFNet compatible with both the point-based encoder and range map-based encoder simultaneously.

Note that both our point-based and range map-based encoders can process scene point clouds with a variable size of K. This makes our SPFNet applicable to real-world scenarios such as autonomous driving datasets where the LiDAR sensor often collects scene point clouds with a different number of points K at different frames, *i.e.*, not a fixed K across all frames.

4.3 Justification to Our Encoder Design

Why not use more advanced encoders? Although our SPFNet has achieved promising performance, we are aware that the components of our SPFNet such as the encoder might not be optimal as there could be many other choices to explore (*e.g.*, replace the PointNet with PointNet++ [5] or other recent point cloud processing techniques such as [6, 7, 8]). However, we would like to emphasize that the main goal of this paper is not to exhaustively search for the best model (encoder, decoder, loss functions) for our SPF task. Instead, our goal is, for the first time, to introduce this interesting new SPF task and a new perception and forecasting pipeline SPF^2 which is built upon our SPFNet and achieves competitive performance compared to the conventional pipeline. We hope that our SPFNet only serves as a start, and future research in this direction can be encouraged and stronger neural network architectures to the proposed SPF task can be proposed.

Why compress the feature in the encoder? Since our proposed SPF is a dense prediction task, *i.e.*, predicting the future locations of $> 100k$ points, it seems to be straightforward to use a fully convolutional architecture without feature compression (*e.g.*, the pooling operator, a large stride in the convolution layer, or the graph pooling layer) in order to maintain fine-grained details in the feature space of the scene point cloud. This is also what we planned to implement at the beginning. However, we later realized that predicting a sequence of large-scale scene point clouds without feature compression can easily reach the GPU memory limits. Note that our sequences are very large (e.g., 30 frames) of large-scale point clouds with $> 100k$ points. As a result, it is necessary to compress the intermediate feature to reduce the GPU memory requirement, *e.g.*, in our two encoders, we compress the feature to a dimension of 1024. We hope that future research in memory-efficient point cloud sequence processing techniques will be encouraged to further improve performance of the SPF task without feature compression.

4.4 Justification to the Loss Function

As mentioned in Sec. 3.2 of the main paper, we use the Chamfer distance (CD) as our primary loss function for both the point-based and range map-based methods[2], which is also a standard practice in most prior works for object point cloud generation/completion [9, 10, 11, 12, 13, 14]. Although a few works [13, 15] also considered using the other well-known Earth mover distance (EMD) as the loss function during training, it is commonly accepted that computing the EMD is much more computationally expensive than computing CD, which is especially true for large-scale scene point clouds in our SPF task. In fact, if we use the CD as our loss function, our SPFNet for predicting future 10 frames of scene point clouds will require about one day to train. However, if we use the EMD as our loss function, the training time for the same model increases to more than a week. Therefore, we only considered using the CD loss for training purpose due to its efficiency.

4.5 Other Implementation Details

We train the entire network SPFNet in an end-to-end fashion using the Adam [16] optimizer for 30 epochs with a learning rate of 1×10^{-4}, betas of 0.9 and 0.999, and batch size of 16. For range map-based methods, we use λ_1 of 0.1 and λ_2 of 0.1 for the additional loss components in the Eq. (1) of the main paper. For point-based methods, λ_1 and λ_2 are simply zeros. For temporal dynamics modeling in Sec. 3.2 of the main paper, we use a standard two-layer LSTM with the hidden feature dimension of 1024.

[2]Two additional losses are only applied for range map-based methods.

5 End-to-End Perception and Trajectory Forecasting Evaluation

5.1 Matching Step and Evaluation Metrics

As mentioned in Sec. 4 of the main paper, we match the predicted future trajectories and GT future trajectories using the Hungarian algorithm in order to establish the correspondences and compute the ADE/FDE metrics. Specifically, we use the pairwise ADE[3] as the metric for the matching. First, we compute the pairwise ADE between every possible pair of predicted and GT future trajectories. For example, if we have U predicted and V GT future trajectories, we will obtain a $U \times V$ matrix with each entry being the pairwise ADE for the pair of GT and predicted trajectories. Note that, for GT objects that do not have trajectories in all future N frames, we mask out the missing frames and still use the available trajectories in valid frames to compute the pairwise ADE.

Once we compute the pairwise ADE matrix with a shape of $U \times V$, we feed it to the Hungarian algorithm to obtain a one-to-one correspondence (*i.e.*, a bijection) between the predicted and GT trajectories. As U and V may be not equal, there could be predicted trajectories or GT trajectories without having a correspondence. Also, for those predicted trajectories which have matched GT trajectories, we reject the matching if the corresponding pairwise ADE is higher than a threshold. In other words, for this predicted trajectory, the matched GT trajectory is the best option that the Hungarian algorithm can find and match it with, which however is still not good enough because the predicted trajectory and its matched GT trajectory have a large displacement error (a high pairwise ADE). As a result, we consider the predicted trajectories as true positives if and only if they have the corresponding matched GTs and also the corresponding pairwise ADEs are lower than the ADE threshold. The rest of unmatched (or matching rejected) predicted trajectories are considered as false positives and the unmatched (or matching rejected) GT trajectories are considered as false negatives. Then, based on the established correspondences for true positives, we can now compute the final ADE/FDE metrics as in the conventional modularized trajectory forecasting evaluation:

$$\text{ADE} = \frac{\sum_{i=1}^{|\mathcal{T}|} \text{ADE}^i_{\text{pairwise}}}{|\mathcal{T}|}, \text{ where } ADE^i_{\text{pairwise}} = \frac{\sum_{j=1}^{N^i_{\text{valid}}} |\widetilde{\mathbf{x}}^i_j - \mathbf{x}^i_j|_2}{N^i_{\text{valid}}}, \tag{2}$$

$$\text{FDE} = \frac{\sum_{i=1}^{|\mathcal{T}|} \text{FDE}^i_{\text{pairwise}}}{|\mathcal{T}|}, \text{ where } FDE^i_{\text{pairwise}} = |\widetilde{\mathbf{x}}^i_{N^i_{\text{valid}}} - \mathbf{x}^i_{N^i_{\text{valid}}}|_2, \tag{3}$$

where $\mathcal{T}$ is the set containing all true positives and $|\mathcal{T}|$ denotes the number of true positives. Also, N^i_{valid} denotes the valid number of frames for the pair of ith predicted and GT trajectory which can vary across true positives, $\mathbf{x}^i_j$ and $\widetilde{\mathbf{x}}^i_j$ denote the ground truth and predicted ground positions at the frame j for the pair of the ith true positive.

Moreover, as we can compute the above ADE/FDE metrics at a given pairwise ADE threshold[4], we can actually compute the full ADE-over-recall and FDE-over-recall curves and ultimately compute the AADE/AFDE metrics through integration. To approximate the integration, we average the values of ADE/FDE at a maximum of L recall values linearly distributed between 0% and 100% to compute the AADE/AFDE. We use L=40 in our experiments so that the interval of recall is 2.5%. Note that not ADE/FDE values at all 40 recall values are used. In fact, we only integrate the ADE/FDE values up to the recall value that all methods can reach. This is because there is no ADE/FDE available for integration at a recall value that is beyond the maximum recall a method can reach. For example, in Fig. 3 (right) of the main paper, the recall that all three methods can reach is 0.425, so we integrate the ADE/FDE to compute AADE/AFDE for all three methods at the recall values from 0.025, 0.05, $\cdots$, to 0.425. Specifically, the AADE/AFDE metrics are computed as following:

$$\text{AADE} = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \text{ADE}_r, \tag{4}$$

$$\text{AFDE} = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \text{FDE}_r, \tag{5}$$

where $\mathcal{R}$ is the set containing all the recall values at which the ADE/FDE are used for integration and $|\mathcal{R}|$ denotes the number of recall values. Again, in the example of Fig. 3 of the main paper, the recall set $\mathcal{R} = \{0.025, 0.05, \cdots, 0.425\}$ and $|\mathcal{R}| = 17$.

[3]Note that the pairwise ADE is different from the final ADE metric used in evaluation. The pairwise ADE is computed for one pair of GT and predicted trajectory by averaging the displacement error over all frames while the final ADE metric is computed by averaging the displacement error over all true positives and all frames.

[4]A different threshold for the pairwise ADE corresponds to a specific recall value of the pipeline.

5.2 Alternatives of the Matching Step

In the above subsection, we have described the matching step used in our experiments in the main paper. Specifically, the matching is conducted between the predicted and GT trajectories in future N frames based on the pairwise ADE values. Alternatively, there could be other variants of this matching step: (1) instead of performing the matching in future N frames, we could match the trajectories from the tracking outputs and GT trajectories in past M frames; (2) moreover, we could match the estimated trajectories (including both the tracking and forecasting outputs) and GT trajectories over the entire past and future $M + N$ frames; (3) instead of using the pairwise ADE for matching, we could compute the pairwise FDE matrix for matching. As a result, there are total 6 different variants for matching, which we believe are all fair for comparison, although each variant might yield slightly different results. We will explore how different matching mechanism affects the performance in the future.

6 Experiments

6.1 Datasets

We further clarify that our experiments (Table 1 and 3 of the main paper) on the KITTI and nuScenes datasets for all methods are conducted for the cars only, not including other objects so far, e.g., Cyclists, Pedestrians. As mentioned in the Sec. 5.1 of the main paper, we have conducted experiments on the nuScenes dataset only for the keyframes, despite the fact that there are many non-key frames. The reason that we only use the keyframes for training and evaluating our SPFNet is that we want to reuse the trained SPFNet in our SPF^2 pipeline. Since the nuScenes dataset only provides object labels for keyframes, our SPF^2 pipeline can only be evaluated on the keyframes. As a result, to reuse our SPFNet in our SPF^2 pipeline, the easiest way is also to train SPFNet for the keyframes, because otherwise the input frame rate might be different which will degrade performance of our SPFNet.

6.2 Ablation Study

To validate the design of our proposed SPFNet, we conduct ablation study on the KITTI validation set. Also, since the baselines we have compared in Table 1 of the main paper may be weak, we believe that different variants of our SPFNet can also serve as stronger baselines. Note that all experiments conducted in the ablation study are based on our range map-based SPFNet.

Table 1: Effect of the loss components.

$\mathcal{L}_{\text{cd}}$	$\mathcal{L}_1$	$\mathcal{L}_{\text{bce}}$	CD↓	EMD↓
✓		✓	1.15	132.63
✓	✓		0.82	110.03
	✓		1.91	219.17
	✓	✓	1.85	203.72
✓	✓	✓	0.73	103.14

Effect of Loss Components. To show that each component of our full object function in Eq. 1 of the main paper is effective, we turn on and off each loss component and check how the performance is affected. Note that the experiments are conducted in the 1.0s prediction setting. We summarize the results in Table 1. To start, the bottom row shows the performance of our full range map-based method on the KITTI val set. Then, to demonstrate that $\mathcal{L}_{\text{cd}}$ is useful, we turn it off and only use $\mathcal{L}_1$ and $\mathcal{L}_{\text{bce}}$ loss functions during training and the performance is shown in the second last row, where the CD and EMD metrics increase by a large margin. This demonstrates that using $\mathcal{L}_{\text{cd}}$ loss is crucial to the performance of our SPFNet on the CD and EMD metrics. Moreover, we removed $\mathcal{L}_{\text{bce}}$ and saw that the CD and EMD metrics slightly increase in the third last row. We believe that this is because using the range mask can help remove some outliers in the generated scene point clouds. Additionally, to show that $\mathcal{L}_1$ loss is also necessary, we turn it off and the performance is shown in the top row. Compared to the last row, we can see that the CD and EMD metrics clearly increase (*e.g.*, from 0.73 to 1.15 for CD), demonstrating that regularizing the intermediate range map output with $\mathcal{L}_1$ largely improves the quality of the generated scene point clouds. Similarly, we can turn $\mathcal{L}_{\text{bce}}$ off and the performance is shown in the second row. As a result, the performance for the CD and EMD metrics slightly decreases (*e.g.*, from 0.73 to 0.82 for CD), although not by a large margin, showing that it is helpful to keep the $\mathcal{L}_{\text{bce}}$ in our SPFNet.

References

[1] 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. *arXiv:1512.03012*, 2015.

[2] X. Liu, C. R. Qi, and L. J. Guibas. FlowNet3D: Learning Scene Flow in 3D Point Clouds. *CVPR*, 2019.

[3] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. *CVPR*, 2016.

[4] L. Caccia, Herke Van Hoof, A. Courville, and J. Pineau. Deep Generative Modeling of LiDAR Data. *IROS*, 2019.

[5] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *NIPS*, 2017.

[6] W. Wu, Z. Qi, and L. Fuxin. PointConv: Deep Convolutional Networks on 3D Point Clouds. *CVPR*, 2019.

[7] Y. Lin, Z. Yan, H. Huang, D. Du, L. Liu, T. Chinese, and H. Kong. FPConv: Learning Local Flattening for Point Convolution. *CVPR*, 2020.

[8] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics*, 2018.

[9] Y. Xia, Y. Zhang, D. Zhou, X. Huang, C. Wang, and R. Yang. RealPoint3D: Point Cloud Generation from a Single Image with Complex Background. *AAAI*, 2019.

[10] C. Zou and D. Hoiem. Silhouette Guided Point Cloud Reconstruction beyond Occlusion. *WACV*, 2020.

[11] X. Wang, M. H. Ang, and G. H. Lee. Cascaded Refinement Network for Point Cloud Completion. *CVPR*, 2020.

[12] Z. Huang, Y. Yu, J. Xu, F. Ni, and X. Le. PF-Net: Point Fractal Network for 3D Point Cloud Completion. *CVPR*, 2020.

[13] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert. PCN: Point Completion Network. *3DV*, 2018.

[14] M. Stypulkowski, M. Zamorski, M. Zieba, and J. Chorowski. Conditional Invertible Flow for Point Cloud Generation. *NeurIPSW*, 2019.

[15] R. Wu, X. Chen, Y. Zhuang, and B. Chen. Multimodal Shape Completion via Conditional Generative Adversarial Networks. *ECCV*, 2020.

[16] D. P. Kingma and J. L. Ba. Adam: A Method for Stochastic Optimization. *ICLR*, 2015.