Self-Supervised Arbitrary-Scale Point Clouds Upsampling via Implicit Neural Representation

Wenbo Zhao[1,2], Xianming Liu[1,2]*, Zhiwei Zhong[1,2], Junjun Jiang[1,2], Wei Gao[3], Ge Li[3], Xiangyang Ji[4]
[1]Harbin Institute of Technology, [2]Peng Cheng Laboratory
[3]Peking University Shenzhen Graduate School, [4]Tsinghua University

Abstract

Point clouds upsampling is a challenging issue to generate dense and uniform point clouds from the given sparse input. Most existing methods either take the end-to-end supervised learning based manner, where large amounts of pairs of sparse input and dense ground-truth are exploited as supervision information; or treat up-scaling of different scale factors as independent tasks, and have to build multiple networks to handle upsampling with varying factors. In this paper, we propose a novel approach that achieves self-supervised and magnification-flexible point clouds upsampling simultaneously. We formulate point clouds upsampling as the task of seeking nearest projection points on the implicit surface for seed points. To this end, we define two implicit neural functions to estimate projection direction and distance respectively, which can be trained by two pretext learning tasks. Experimental results demonstrate that our self-supervised learning based scheme achieves competitive or even better performance than supervised learning based state-of-the-art methods. The source code is publicly available at https://github.com/xnowbzhao/sapcu.

1. Introduction

Point clouds serve as a popular tool to represent 3D data due to their flexibility and compactness in describing objects/scenes with complex geometry and topology. They can be easily captured by modern scanning devices, and have been widely used in many applications, such as autonomous driving, robotics, etc. However, due to the inherent limitations of 3D sensing technology, raw point clouds acquired from 3D scanners are usually sparse, occluded and non-uniform. In many downstream applications, such as surface reconstruction and understanding, dense point clouds are desired for representing shapes with richer geometric details. Accordingly, people turn to develop a computational approach, referred to as point clouds upsampling, which has attracted extensive attention in both industry and academia [7, 8, 13, 20–22].

*Corresponding author: csxm@hit.edu.cn

Unlike conventional grid based images, point clouds are irregular and unordered, which make point clouds upsampling a more challenging task than its 2D image counterpart. The goal of point clouds upsampling is two-fold: 1) generating a dense point set from the sparse input to provide richer details of the object; 2) generating a uniform and complete point set to cover the underlying surface faithfully.

In recent years, deep neural networks based point clouds upsampling approaches emerge and become popular, which adaptively learn structures from data and achieve superior performance than traditional methods, such as optimization-based ones [1, 5, 6]. For instance, Yu *et al.* [21] propose to learn multi-level features per point and expand the point set via a multi-branch convolution unit implicitly in feature space, which is then split to a multitude of features for reconstruction of an upsampled point set. Wang *et al.* [20] propose to progressively train a cascade of patch-based upsampling networks on different levels of detail. Li *et al.* [7] apply generative adversarial network into point clouds upsampling, which constructs an up-down-up expansion unit in the generator for upsampling point features with error feedback and self-correction, and formulate a self-attention unit to enhance the feature integration. To better represent locality and aggregate the point neighborhood information, Qian *et al.* [13] propose to use a Graph Convolutional Network to perform point clouds upsampling.

In summary, the outlined deep learning based methods take a general approach: first design an upsampling module to expand the number of points in the feature space, then formulate losses to enforce the output points to be as close as possible to the ground truth dense points. However, these methods suffer from the following two limitations:

End-to-End Training. These methods are trained in an end-to-end supervised learning manner, which requires a large amount of pairs of input sparse and ground-truth dense point sets as the supervision information. The training data is constructed by sampling from synthetic models, whose distributions are inevitably biased from that of the realscanned data. This would lead the trained models to have poor generalization ability in real-world applications. Thus, it is more desirable to develop self-supervised or unsupervised point cloud upsampling schemes.

Fixed Upsampling Factor. Due to resource constraints, such as display resolution and transmission bandwidth, the required upsampling factor is usually various. These existing methods treat upscaling of different scale factors as independent tasks, which train a specific deep model for a pre-defined factor and have to build multiple networks to handle upsampling with varying factors. This manner is clumsy, which increases both model complexity and training time significantly. Thus, it is more desirable to develop unified point cloud upsampling schemes that can handle arbitrary scale factor.

Some methods developed very recently [9, 14, 19, 22] investigate the above limitations and attempt to address them:

- Regarding self-supervised point cloud upsampling, Liu *et al.* [9] propose the coarse-to-fine framework, which downsamples the input sparse patches into sparser ones and then exploits them as pairs of supervision information to perform end-to-end training. [22] proposes an end-to-end self-supervised learning manner, in which the loss functions enforce the input sparse point cloud and the generated dense one to have similar 3D shapes and rendered images. However, these two methods are still limited to a fixed upsampling factor.

- Regarding arbitrary-scale upsampling, inspired by the counterpart Meta-SR in image [4], Ye *et al.* [19] propose Meta-PU for magnification-flexible point cloud upsampling, in which the meta-subnetwork is learned to adjust the weights of the upsampling blocks dynamically. Qian *et al.* [14] design a neural network to adaptively learn unified and sorted interpolation weights as well as the high-order refinements, by analyzing the local geometry of the input point cloud. However, these two methods still follow the end-to-end supervised learning manner, which need to construct a large-scale training set including ground truth dense point sets with scales within a wide range.

In this paper, we propose a novel and powerful point clouds upsampling method via implicit neural representation, which can achieve self-supervised and magnification-flexible upsampling simultaneously. Specifically, to get rid of the requirement of ground truth dense point clouds, we do not directly learn the mapping between the input sparse and output dense point sets. Alternatively, inspired by the notion that an implicit surface can be represented by signed distance function (SDF) [10, 11, 15], we turn to seek the nearest projection points on the object surface for given seed points through two implicit neural functions, which are used to estimate projection direction and distance respectively. The two function can be trained by two constructed pretext self-supervised learning tasks. In the way, as long as the seed points are sampled densely and uniformly, we can produce high-resolution point clouds that are dense, uniform and complete. To guarantee the uniformity of seeds sampling, we exploit equally-paced 3D voxels to divide the space of point cloud. Experimental results demonstrate our self-supervised learning based scheme achieves competitive or even better performance that supervised learning based state-of-the-art methods. The main contributions of this work are highlighted as follows:

- To the best of our knowledge, we are the first in the literature to simultaneously consider self-supervised and arbitrary-scale point clouds upsampling.

- We formulate point clouds upsampling as the task of seeking nearest projection points on the implicit surface for seed points, which can be done by two implicit neural functions trained by pretext tasks. From the generated dense point clouds, we can achieve arbitrary-scale upsampling by farthest point sampling.

- Although our method is self-supervised, it produces high-quality dense point clouds that are uniform and complete, and achieves competitive objective performance and even better visual performance compared with state-of-the-art supervised methods.

2. Method

Define $\mathcal{X} = \{\mathbf{p}_i\}_{i=1}^{n} \in \mathbb{R}^{n\times 3}$ as the input sparse point cloud. For a desirable scaling factor r, we target to obtain a corresponding dense point cloud $\mathcal{Y} = \{\mathbf{p}_i\}_{i=1}^{N} \in \mathbb{R}^{N\times 3}$ including $N = \lfloor r \times n \rfloor$ points. $\mathcal{S}$ is defined as the underlying surface of the dense point cloud. The high-resolution point cloud $\mathcal{Y}$ is required to be dense and uniform, as well as be able to handle occlusion and noise, *i.e.*, to be complete and clean.

Unlike the existing methods that take the end-to-end training framework, we do not directly learn the mapping between the input sparse and output dense point sets, but instead seek the nearest projection point on the object surface for a given seed point in a self-supervised manner. By densely and uniformly sampling seed points in the space, we can obtain dense and approximately uniform projection points, which can describe the underlying surface faithfully. The proposed self-supervised point clouds upsampling strategy includes the four steps:

- Seeds Sampling. We represent the geometric space of the point cloud by 3D voxel grid, from which we choose the centres of voxels that are close to the implicit surface $\mathcal{S}$ as the seed points.

- Surface Projection. For seed points, we project them to the implicit surface $\mathcal{S}$ to obtain the projected points, which construct the generated dense point cloud.
- Outliers Removal. We further remove the projected points that are generated by far seed points to achieve cleaner point cloud.
- Arbitrary-Scale Point Cloud Generation. To obtain the desired upsampling factor, we adjust the number of vertices of the generated dense clouds by farthest point sampling.

In the following, we introduce each step in detail.

2.1. Seeds Sampling

To obtain uniformly sampled seed points, given a point cloud, we divide the 3D space into equally spaced voxels $\{\mathbf{V}_{(x,y,z)}\}$, where $\mathbf{V}_{(0,0,0)}$ represents the voxel locating in the origin of the 3D Cartesian coordinate system. We define the resolution of a 3D voxel volume as $l \times l \times l$. The centre of a voxel $\mathbf{V}_{(x,y,z)}$ is thus $\mathbf{c}_{(x,y,z)} = [x + 0.5 * l, y + 0.5 * l, z + 0.5 * l]$. The centres of voxels are equally distributed in the space, which serve as good candidates of seed points. However, we do not use them all, but choose the ones that are closed to the underlying surface of the point cloud.

A reasonable principle to choose centres is according to their distances to the surface $\mathcal{S}$. We choose a centre $\mathbf{c}_{(x,y,z)}$ as the seed if its distance to the surface within a preset range: $\text{Dist}(\mathbf{c}_{(x,y,z)}, \mathcal{S}) \in [D_l, D_u]$. The difficulty lies in that we cannot directly compute the distance $\text{Dist}(\mathbf{c}_{(x,y,z)}, \mathcal{S})$, since the underlying surface $\mathcal{S}$ is unknown. We propose an alternative strategy to approximate $\text{Dist}(\mathbf{c}_{(x,y,z)}, \mathcal{S})$. Specifically, from the input sparse point sets $\mathcal{X}$, we choose M points that are nearest to $\mathbf{c}_{(x,y,z)}$, denoted as $\{\mathbf{p}_{c,1}, \mathbf{p}_{c,2}, \cdots, \mathbf{p}_{c,m}, \cdots, \mathbf{p}_{c,M}\}$ that are ordered from near to far. From these points, we can form a set of triangles $\{T_m = (\mathbf{p}_{c,1}, \mathbf{p}_{c,2}, \mathbf{p}_{c,m})\}_{m=3}^{M}$. We then perform the following approximation:

$$\text{Dist}(\mathbf{c}_{(x,y,z)}, \mathcal{S}) \approx \min \text{Dist}(\mathbf{c}_{(x,y,z)}, t), t \in \{T_m\}_{m=3}^{M} \tag{1}$$

where t represents a point contained in the constructed triangles. Finally, we obtain the seed points set C. By setting appropriate l, we can generate dense and uniformly distributed seed points.

2.2. Surface Projection

With the sampled seed points, the next step is to seek the their projection points on the surface, which are the target points of the generated dense point cloud.

In the field of 3D computer vision and graphics, it is well-known that an implicit surface can be defined as a signed distance function (SDF) [10, 11, 15]. SDF, when passed the coordinates of a point in space, outputs the closest distance of this point to the surface, whose sign indicates whether the point is inside or outside of the surface. Inspired by SDF, we propose the following feasible approach to estimate the projected point on the surface for a query seed point.

It is worth noting that, the computation strategies of SDF, such as [10, 11, 15], cannot be directly applied for our purpose. For a 3D query point $\mathbf{x}$, SDF outputs: $SDF(\mathbf{x}) = s, s \in \mathbb{R}$. The sign of s only indicates it is inside or outside of a shape, but does not provide the direction to the surface. In our method, for a seed point $\mathbf{c} \in C$, we divide the task of estimating projection point into two sub-tasks: 1) estimating the projection direction $\mathbf{n} \in [-1, 1]^3$; 2) estimating the projection distance $d \in \mathbb{R}$. Then the coordinate of the projection point of the seed point $\mathbf{c}$ can be obtained as: $\mathbf{c}_p = \mathbf{c} + \mathbf{n} * d$.

Projection Direction Estimation. We train a multi-layer fully-connected neural network $f_n(\cdot; \Theta_n)$ for this purpose, which takes the query point $\mathbf{c}$ and the sparse point cloud $\mathcal{X}$ as inputs:

$$\mathbf{n} = f_n(\mathbf{c}, \mathcal{X}; \Theta_n) \tag{2}$$

To reduce the computational complexity, we take k nearest points to $\mathbf{c}$ in $\mathcal{X}$ instead of the whole $\mathcal{X}$ as input. We denote this subset of points as $\mathcal{X}_c = \{\mathbf{p}_1, \cdots, \mathbf{p}_k\}$. Moreover, to facilitate the inference process of neural networks, we perform normalization on the point coordinates by setting $\mathbf{c}$ as the origin. In this way, we can simplify the estimation function as:

$$\mathbf{n} = f_n(\widehat{\mathcal{X}}_c; \Theta_n) \tag{3}$$

where $\widehat{\mathcal{X}}_c = \{\mathbf{p}_1 - \mathbf{c}, \cdots, \mathbf{p}_k - \mathbf{c}\}$.

Projection Distance Estimation. Similarly, for estimating the projection distance $\mathbf{d}$, we also train a multi-layer fully-connected neural network $f_d(\cdot; \Theta_n)$, which takes the query point $\mathbf{c}$, the subset of nearest points $\mathcal{X}_c$ and the estimated projection direction $\mathbf{n}$ as inputs:

$$d = f_d(\mathbf{c}, \mathcal{X}_c, \mathbf{n}; \Theta_d) \tag{4}$$

Normalization is also helpful for this network. Different from f_n, here the input $\mathbf{n}$ involves direction. Therefore, it should perform normalization on both position and direction, which can be done in two stages: 1) moving $\mathbf{c}$ to the origin; 2) applying the rotation matrix $\mathbf{W}_r$ to rotate $\mathbf{n}$ to a specific direction $\mathbf{n}_t$, *i.e.*, $\mathbf{n}_t = \mathbf{W}_r\mathbf{n}$. After normalization, $\mathcal{X}_c$ becomes $\widetilde{\mathcal{X}}_c = \{\mathbf{W}_r(\mathbf{p}_1 - \mathbf{c}), \cdots, \mathbf{W}_r(\mathbf{p}_k - \mathbf{c})\}$, which is the only required input for f_d:

$$d = f_d(\widetilde{\mathcal{X}}_c; \Theta_d) \tag{5}$$

2.3. Outliers Removal

In the step of seeds sampling, some points that are actually far away from $\mathcal{S}$ may be included into the seed points

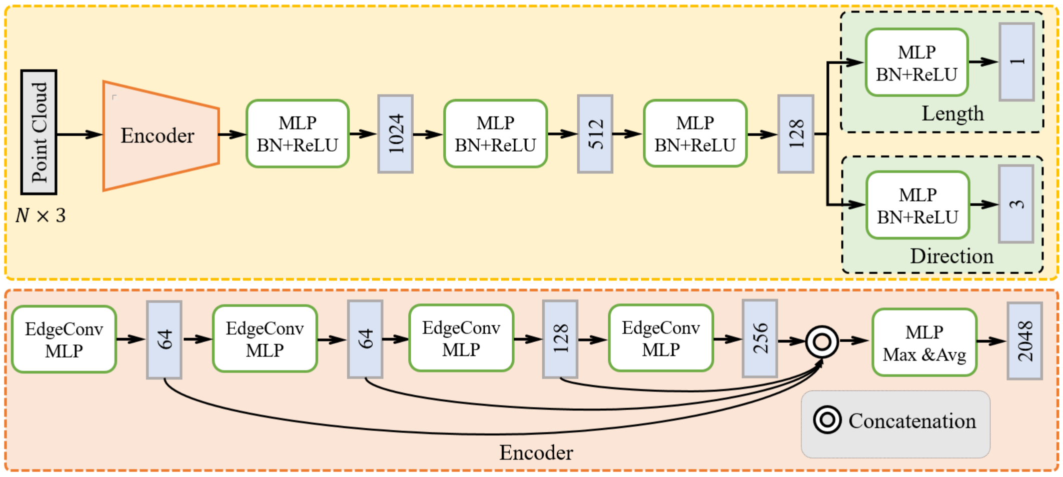

Figure 1. The network architecture of implicit neural function.

set C due to the error in approximation. The normal vector and distance of these points cannot be well estimated, leading to outliers in the resulting dense point cloud. We turn to exploit the post-processing procedure to remove them.

Specifically, for a projection point $\mathbf{c}_p$, we find its v nearest points $\{\mathbf{c}_{p,1}, \cdots, \mathbf{c}_{p,v}\}$. We then compute the average bias between $\mathbf{c}_p$ and them:

$$b_p = \frac{1}{v}\sum_{i=1}^{v} \text{Dist}(\mathbf{c}_p, \mathbf{c}_{p,i}) \tag{6}$$

For all projection points, we do in the same way to get $\{b_p\}$, the average of which is denoted as $\bar{b}$. We determine a point as outlier if it satisfies $b_p > \lambda\bar{b}$, where λ is set as 1.5 in practical implementation.

2.4. Arbitrary-Scale Point Cloud Generation

Note that the above process cannot accurately control the number of vertices generated. Thus, it is necessary to adjust the number of vertices to achieve upsampling with the desired scale factor. In our context, we first perform inverse normalization on the generated point cloud, and then adjust the number of vertices to N by the farthest point sampling algorithm [12].

3. Implicit Neural Networks Training

In this section, we introduce the architecture of implicit neural networks and the training strategy.

3.1. Architectures

The networks f_n and f_d share the same architecture as shown in Figure 1, which borrows the idea of encoder-decoder framework [10]. The network takes the normalized subset of points as input, which are feed into the encoder to obtain a 2048-dimensional feature vector. Here we employ a state-of-the-art method DGCNN [17] as the encoder to preserve surface information in multi-levels. The feature vector is then passed through 4 full-connected (FC) layers with batch normalization and ReLU, the output dimensional of which are 1024, 512 and 128 respectively. The output dimensional of the last FC layer is 3 for the projection direction $\mathbf{n}$ and 1 for the the projection distance d. Note that the design of the network is not the main contribution of this paper. We can exploit any suitable network for our purpose.

3.2. Training Data Preparation

To train the two implicit neural functions, we construct two pretext tasks, for which we prepare training samples that consist of 3D points and the corresponding ground truth projection direction and distance values. We train with normalized watertight meshes that are constructed by the TSDF-Fusion presented in [10, 16] from a subset of the ShapeNet [2] that consists of 13 major categories. Since f_n and f_d are designed with different purposes, we prepare different training pairs for them:

- For preparing the training data of f_n, we firstly generate the seed points: 50K seed points are randomly selected around the mesh surface by limiting that the distances between them and the surface are within a preset range $[D_l - \epsilon, D_u + \epsilon]$, where ϵ is introduced to increase robustness. For a seed point $\mathbf{c}$, we find the nearest point on the mesh and sample 5 points around it, then compute the average vector $\mathbf{d}$ between $\mathbf{c}$ and them. In this way, we can effectively handle the error in mesh reconstruction. The ground truth $\widehat{\mathbf{n}}$ is finally derived as the normalization of $\mathbf{d}$. Secondly, for every 16 seed points, we randomly select 2048 points as the corresponding sparse point cloud $\mathcal{X}$.

- For preparing the training data of f_d, 50K seed points are randomly selected around the mesh surface in the

Scale		2×					4×				
Metric (10^{-3})		CD ↓	EMD ↓	F-score ↑	mean ↓	std ↓	CD ↓	EMD ↓	F-score ↑	mean ↓	std ↓
Fixed Scale	PU-Net [21]	12.9 (6)	6.75 (6)	334 (5)	4.02 (5)	5.62 (4)	11.3 (7)	7.02 (7)	462 (7)	5.05 (7)	6.81 (6)
	MPU [20]	-	-	-	-	-	10.4 (4)	5.64 (5)	527 (4)	3.61 (4)	5.50 (3)
	PU-GAN [7]	12.7 (5)	5.09 (5)	326 (6)	4.32 (6)	6.01 (5)	10.9 (5)	6.66 (6)	484 (6)	4.66 (6)	6.56 (5)
	PU-GCN [13]	12.2 (3)	4.94 (4)	360 (3)	3.47 (2)	5.09 (2)	10.2 (3)	5.50 (4)	537 (3)	3.35 (2)	5.01 (2)
	PU-DR [8]	11.0 (1)	3.55 (1)	409 (1)	2.30 (1)	4.05 (1)	8.71 (1)	3.98 (2)	625 (1)	2.24 (1)	3.89 (1)
Arbitrary Scale	Meta-PU [19]	12.6 (4)	3.94 (2)	339 (4)	3.77 (4)	5.41 (3)	10.9 (5)	3.56 (1)	506 (5)	3.89 (5)	5.60 (4)
	Proposed	12.1 (2)	4.93 (3)	371 (2)	3.49 (3)	10.2 (6)	10.1 (2)	4.87 (3)	561 (2)	3.49 (3)	9.35 (7)
Scale		8×					16×				
Metric (10^{-3})		CD ↓	EMD ↓	F-score ↑	mean ↓	std ↓	CD ↓	EMD ↓	F-score ↑	mean ↓	std ↓
Fixed Scale	PU-Net [21]	9.67 (5)	8.91 (6)	611 (6)	4.85 (6)	6.81 (5)	9.25 (7)	10.4 (7)	632 (7)	6.04 (7)	8.04 (6)
	MPU [20]	-	-	-	-	-	8.12 (5)	7.74 (6)	727 (5)	4.01 (6)	6.11 (5)
	PU-GAN [7]	8.82 (4)	5.05 (3)	676 (4)	3.76 (4)	5.51 (3)	7.52 (2)	6.02 (4)	770 (3)	3.14 (2)	4.92 (2)
	PU-GCN [13]	8.78 (3)	6.41 (5)	678 (3)	3.33 (2)	5.06 (2)	7.80 (4)	7.44 (5)	749 (4)	3.39 (4)	5.11 (3)
	PU-DR [8]	8.34 (1)	3.95 (1)	708 (1)	2.99 (1)	4.97 (1)	7.29 (1)	4.51 (1)	779 (1)	2.92 (1)	4.72 (1)
Arbitrary Scale	Meta-PU [19]	9.71 (6)	4.33 (2)	625 (5)	3.95 (5)	5.68 (4)	8.96 (6)	5.62 (2)	676 (6)	3.87 (5)	5.59 (4)
	Proposed	8.70 (2)	5.53 (4)	706 (2)	3.48 (3)	8.84 (6)	7.65 (3)	5.98 (3)	772 (2)	3.35 (3)	8.34 (7)

Table 1. Objective performance comparison with respect to CD, EMD, F-score, mean and std with state-of-the-art methods. The ranking numbers are also provided.

same way as f_n. We then find the corresponding nearest projection points on the surface. The distance between a seed point and the projection point is used as the ground truth $\widehat{d}$. The generation of $\mathcal{X}$ is in the same way as f_n.

It is worth noting that, although the training data are generated from watertight meshes, our scheme is capable to handle non-watertight point clouds, which can be observed in the real-world cases shown in Figure 6.

3.3. Training Details

The training of f_n and f_d is done by minimizing the sum over losses between the predicted and real direction/length values under the mean squared error (MSE) loss function. The training process is conducted on a server with two Tesla V100 GPUs. The networks are trained for 1200 epochs with a batch size of 64, using the Adam algorithm. Following [10], the learning rate is set as 10^{-4}, and the other hyperparameters of Adam are set as $\beta_1 = 0.9$, $\beta_2 = 0.999$, $epsilon = 10^{-8}$, weight decay = 0.

4. Experiments

In this section, we provide extensive experimental results to demonstrate the superior performance of our method.

4.1. Comparison Study

We compare the proposed self-supervised arbitrary-scale point clouds upsampling (SSAS) method with several state-of-the-art works, which can be divided into two categories in term of the scale factor: 1) fixed scale methods, including PU-Net [21], MPU [20], PU-GAN [7], PU-GCN [13], PU-DR [8]; 2) arbitrary scale method, including Meta-PU [19]. Note that these methods are all supervised learning based. The compared models are trained with the released codes by their authors, following the default settings in their papers.

We train all these compared methods following the approach mentioned in [19] for fair comparison. The test samples are from the dataset adopted by [19, 21]. We non-uniformly sample 2048 points using Poisson disk sampling from 20 test models to form the test set.

4.2. Parameters Setting

The side length of voxel l and the range of the distance between seed point and surface $[D_l, D_u]$ decide the number of seed points. However, the number is also depended on the shape of input point cloud. To ensure that enough points are generated, we set $l = 0.004$ and $[D_l, D_u] = [0.011, 0.015]$. Under this condition, the minimum number of generated points is 99001 (*Chair*), which meets the demands of 16× or higher scale upsampling. The specific direction $\mathbf{n}_t$ in normalization can be arbitrarily chosen. We set $\mathbf{n}_t = (1, 0, 0)$ in our experiments. The number of nearest points k is set as 100 in Section 2.2 and 30 in Section 2.3. The number of nearest point M for computing Dist($\mathbf{c}_{(x,y,z)}, \mathcal{S}$) affects the number of outliers and continuity of seed points. We set $M = 10$ and further discuss the effect of different M in ablation study.

4.3. Objective Performance Comparison

Objective Evaluation. We employ six popular metrics for objective evaluation: 1) **Chamfer Distance (CD)** and **Earth Mover Distance (EMD)** [21]: which evaluate the similarity between the predicted points and ground truth one in the Euclidean space. For both metrics, smaller is better. 2) **F-score** [18]: which treats upsampling as a clas-

Scale		2×					4×				
p (10^{-2})		0.2% ↓	0.4% ↓	0.6% ↓	0.8% ↓	1.0% ↓	0.2% ↓	0.4% ↓	0.6% ↓	0.8% ↓	1.0% ↓
Fixed Scale	PU-Net [21]	3.05 (6)	2.33 (6)	2.03 (6)	1.86 (6)	1.75 (6)	2.72 (7)	2.19 (7)	1.97 (7)	1.84 (7)	1.76 (7)
	MPU [20]	-	-	-	-	-	2.53 (6)	2.03 (6)	1.80 (6)	1.67 (6)	1.59 (6)
	PU-GAN [7]	2.46 (3)	1.86 (3)	1.62 (4)	1.50 (4)	1.43 (4)	2.45 (4)	1.94 (5)	1.73 (5)	1.62 (5)	1.56 (5)
	PU-GCN [13]	2.68 (5)	2.06 (5)	1.80 (5)	1.65 (5)	1.57 (5)	2.40 (3)	1.93 (4)	1.72 (4)	1.61 (4)	1.54 (4)
	PU-DR [8]	1.83 (1)	1.34 (1)	1.17 (1)	1.09 (1)	1.06 (1)	1.77 (2)	1.46 (2)	1.34 (2)	1.28 (2)	1.24 (2)
Arbitrary Scale	Meta-PU [19]	2.53 (4)	1.86 (3)	1.58 (3)	1.43 (3)	1.35 (3)	2.50 (5)	1.87 (3)	1.60 (3)	1.45 (3)	1.37 (3)
	Proposed	1.96 (2)	1.50 (2)	1.33 (2)	1.25 (2)	1.21 (2)	1.72 (1)	1.40 (1)	1.27 (1)	1.21 (1)	1.19 (1)
Scale		8×					16×				
p (10^{-2})		0.2% ↓	0.4% ↓	0.6% ↓	0.8% ↓	1.0% ↓	0.2% ↓	0.4% ↓	0.6% ↓	0.8% ↓	1.0% ↓
Fixed Scale	PU-Net [21]	2.64 (5)	2.21 (6)	2.02 (6)	1.91 (6)	1.84 (6)	2.86 (6)	2.42 (6)	2.22 (7)	2.10 (7)	2.02 (7)
	MPU [20]	-	-	-	-	-	2.30 (4)	1.96 (4)	1.82 (4)	1.73 (4)	1.69 (4)
	PU-GAN [7]	1.72 (2)	1.47 (3)	1.38 (3)	1.34 (3)	1.32 (3)	1.79 (3)	1.57 (3)	1.48 (3)	1.44 (3)	1.41 (3)
	PU-GCN [13]	2.31 (4)	1.93 (4)	1.75 (4)	1.65 (5)	1.59 (5)	2.42 (5)	2.07 (5)	1.91 (5)	1.82 (5)	1.76 (5)
	PU-DR [8]	1.42 (1)	1.20 (1)	1.13 (1)	1.11 (1)	1.11 (1)	1.56 (1)	1.38 (1)	1.32 (1)	1.29 (1)	1.28 (1)
Arbitrary Scale	Meta-PU [19]	2.72 (6)	2.05 (5)	1.76 (5)	1.60 (4)	1.51 (4)	3.27 (7)	2.51 (7)	2.14 (6)	1.93 (6)	1.79 (6)
	Proposed	1.72 (2)	1.45 (2)	1.34 (2)	1.29 (2)	1.27 (2)	1.75 (2)	1.51 (2)	1.41 (2)	1.36 (2)	1.33 (2)

Table 2. Uniformity performance comparison with respect to NUC scores. The ranking numbers are also provided.

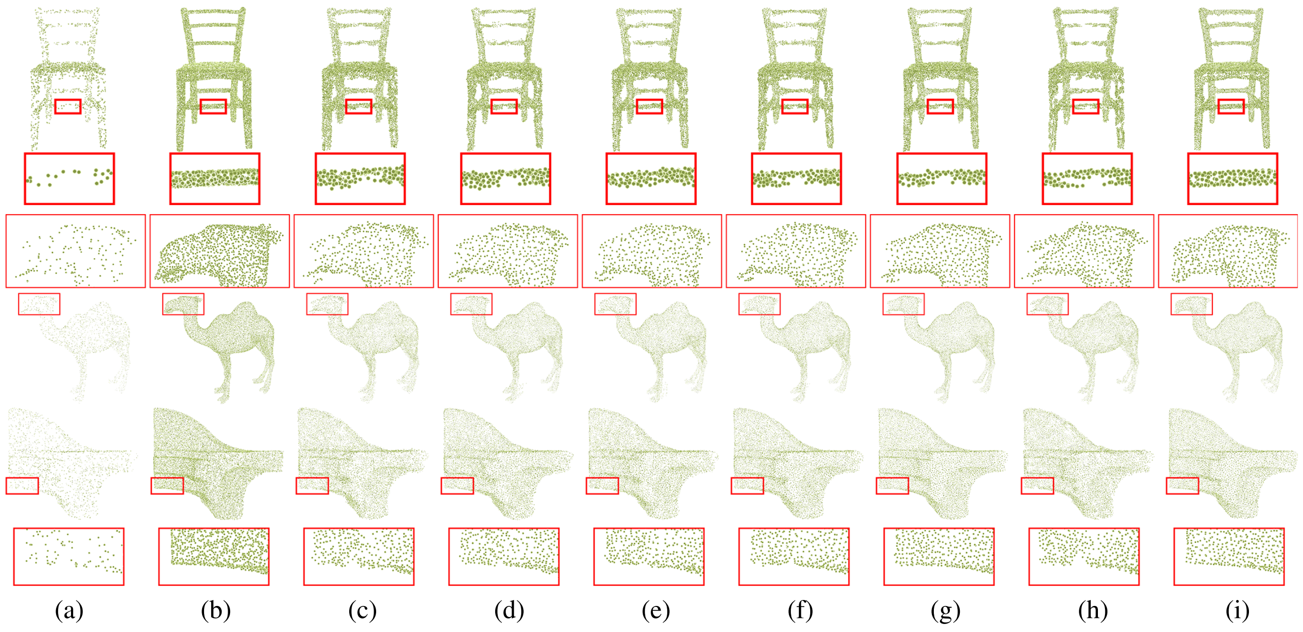

Figure 2. $4\times$ point upsampling results of *Chair*, *Camel* and *Fandisk*. (a) the input point cloud; (b) the ground truth; (c) to (i) the results of PU-Net [21] , MPU [20], PU-GAN [7], PU-GCN [13], PU-DR [8], Meta-PU [19] and ours. Please enlarge the PDF for more details.

sification problem. For this metric, larger is better. 3) **mean** and **std** [21]: which evaluate the distance between the predicted point cloud and ground truth mesh. For both metrics, smaller is better. 4) **Normalized Uniformity Coefficient (NUC)** [21]: which evaluates the uniformity of points on randomly selected disk with different area percentage $p = 0.2\%, 0.4\%, 0.6\%, 0.8\%, 1.0\%$. For this metric, smaller is better.

In Table 1, we offer the comparison results for four scale factors $[2\times, 4\times, 8\times, 16\times]$ with respect to CD, EMD, F-score, mean and std. Surprisingly, it can be found that, although our model is trained in a self-supervised manner without accessing to the ground-truth, it achieves competitive performance with those supervised learning based ones with respect to metrics CD, EMD, F-score and mean. Taking CD for example, our method is ranked #2, #2, #2 and #3 among seven compared methods for $[2\times, 4\times, 8\times, 16\times]$ respectively. Similar results can be found for F-score, for

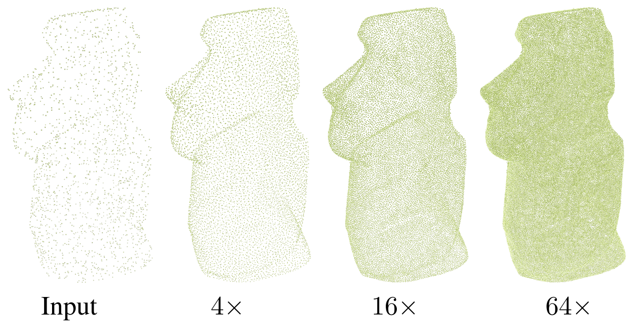

Figure 3. 4×, 16× and 64× upsampling results of *Moai*.

which our method is ranked #2 for all cases. Note that our method performs worst with respect to std, which is because that outliers cannot be completely removed.

Table 2 shows the uniformity evaluation results with respect to NUC. Our method is ranked #2 for $[2\times, 8\times, 16\times]$, just blow PU-DR [8]. In the case of 4×, our method is ranked #1. These results demonstrate that the proposed method produces dense and uniform point clouds.

Inference Time Cost Comparison. This experimental analysis is conducted on a server with two 1080Ti GPU. In our inference process, estimation of direction and length are the most time-consuming steps. According to the experiments, generating 40000 projected points by our method would cost 46s in average. As a comparison, the time costs of generating 40000 points (16×) are 342.4s by MPU [20] and 0.228s by Meta-PU [19]. It should be noticed that the estimation of each point is independently performed and thus can be done in parallel to speed up significantly.

4.4. Subjective Performance Comparison

Visual Comparison. Figure 2 illustrates the 4× upsampling results generated by our method and the compared state-of-the-art methods on three models *Chair*, *Camel* and *Fandisk*. The results show that our method achieves better visual performance than other methods. The produced high-resolution point clouds are dense and uniform, which also have continuous and complete contours. Specifically, results of the highlighted part of *Chair* show that our method succeeds in recovering structure from very few points; results of the highlighted part of *Camel* show that our method can handle complex contour. Furthermore, our method can reconstruct the edge region very well, as demonstrated in the highlighted part of *Fandisk*. The above visual comparisons verify the superiority of our proposed method.

Results on Variable Scales. Figure 3 shows the upsampling results of *Moai* with different scale factors. It can be observed that the contours of all the results are consistent, and the uniformity of points is well preserved.

Robustness against Varying Sizes of Input. Figure 4 shows the 4× upsampling results of *Eight* with different

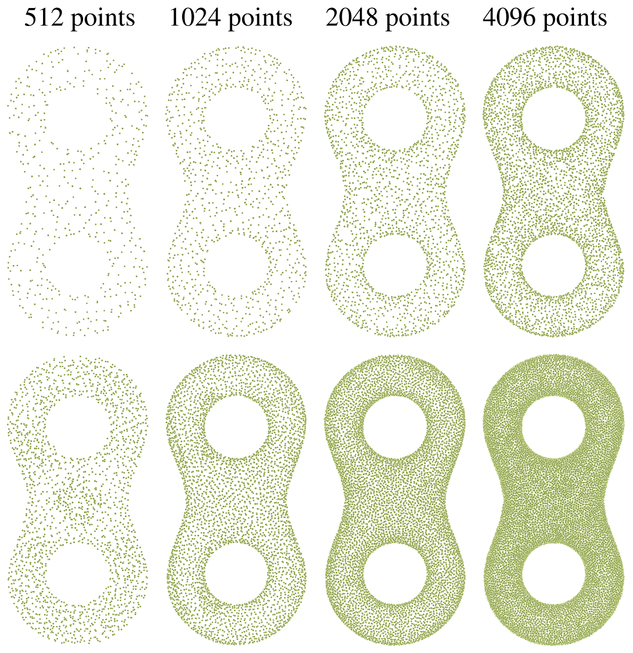

Figure 4. 4× Upsampling results of *Eight* with varying size of input. The first row are the inputs, the second row are the corresponding upsampling results.

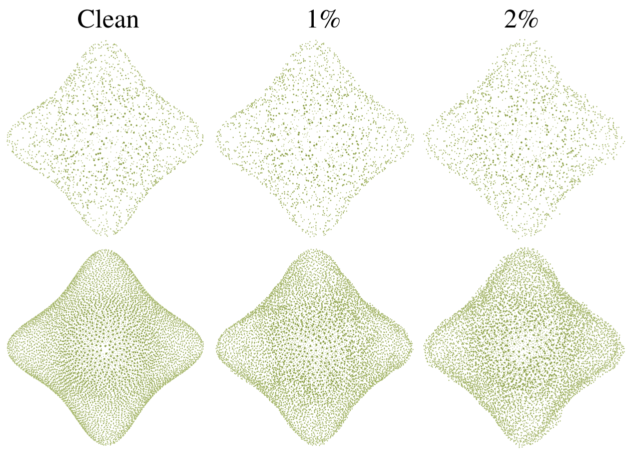

Figure 5. 4× Upsampling results of *Star* with different additive Gaussian noise level.

sizes of input point sets. Our method generates consistent outlines regardless of the number of input points. Figure 5 shows the 4× upsampling results of *star* with noise level 0%, 1% and 2%. Our scheme also works well on the noisy input while the uniformity is well preserved. Overall, our method is robust to the input size and noise.

Result on Real-world Sample. We choose one real-world sample from KITTI [3] to evaluate the generalization capability of our method. In Figure 6, the upsampling results

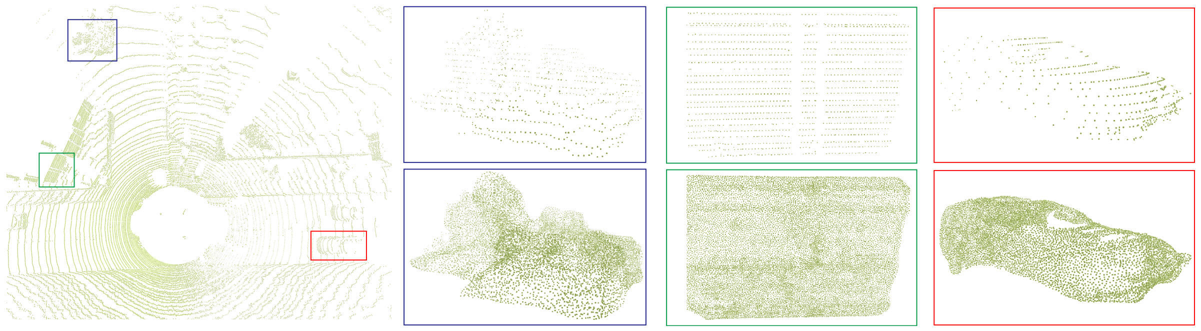

Figure 6. 8× Upsampling result on a real-world sample from KITTI. Please enlarge the PDF for more details.

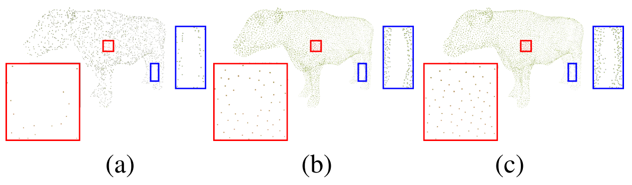

Figure 7. Ablation on the choice of M. (a) input point cloud, (b) $M = 3$, (c) $M = 10$.

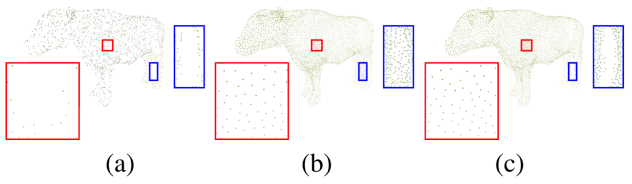

Figure 8. Ablation on the outliers removal. (a) input point cloud, (b) without outliers removal, (c) with outliers removal.

of three different regions are presented. It can be found that, even though the input point cloud is sparse and non-uniform, our scheme can still recover the high-resolution one very well.

4.5. Ablation Study

About the choice of M. M works in the distance approximation of a seed point to the underlying surface, which affects the number of outliers and the continuity of projected points. When M increases, both the continuity and the number of outliers increase. To show the effect of different M, we provided the 4× upsampling result of *cow* with $M = 3$ and $M = 10$. The results are shown in Figure 7. It can be observed that, when $M = 3$, no outlier is introduced to the blue box. However, the surface is discontinuous in the red box. When $M = 10$, the surface in the red box becomes continuous, however, a few amount of outliers is still introduced to the blue box after outliers removal.

About the necessity of outliers removal. To show the effect of outliers removal, we provid the 4× upsampling result of *cow* with and without outliers removal. The results are shown in Figure 8. It can be observed that the outliers removal does not affect the smooth region in the red box, while it can remove most of the outliers in the blue box.

4.6. Limitations

The limitation of our method are two-fold. Firstly, even outliers removal is performed, there still exist a certain number of outliers. Secondly, our method cannot precisely control the number of upsampled point set. We have to first generate a dense one with over-sampled points and then adjust the number of vertices to the target number by the farthest point sampling algorithm.

5. Conclusion

In this paper, we present a novel and effective point clouds upsampling method via implicit neural representation, which can achieve self-supervised and arbitrary-scale upsampling simultaneously. We formulate point clouds upsampling as the task of seeking nearest projection points on the implicit surface for seed points, which can be done by two implicit neural functions trained without the ground truth dense point clouds. Extensive experimental results demonstrate that our method can produce high-quality dense point clouds that are uniform and complete, and achieves competitive objective performance and even better visual performance compared with state-of-the-art supervised methods.

6. Acknowledgements

This work was supported by National Key Research and Development Project under Grant 2019YFE0109600, National Natural Science Foundation of China under Grants 61922027, 6207115 and 61932022.

References

[1] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on visualization and computer graphics*, 9(1):3–15, 2003. 1

[2] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 4

[3] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012. 7

[4] Xuecai Hu, Haoyuan Mu, Xiangyu Zhang, Zilei Wang, Tieniu Tan, and Jian Sun. Meta-SR: A magnification-arbitrary network for super-resolution. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1575–1584, 2019. 2

[5] Hui Huang, Dan Li, Hao Zhang, Uri Ascher, and Daniel Cohen-Or. Consolidation of unorganized point clouds for surface reconstruction. *ACM transactions on graphics (TOG)*, 28(5):1–7, 2009. 1

[6] Hui Huang, Shihao Wu, Minglun Gong, Daniel Cohen-Or, Uri Ascher, and Hao Zhang. Edge-aware point set resampling. *ACM transactions on graphics (TOG)*, 32(1):1–12, 2013. 1

[7] Ruihui Li, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. PU-GAN: A point cloud upsampling adversarial network. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7202–7211, 2019. 1, 5, 6

[8] Ruihui Li, Xianzhi Li, Pheng-Ann Heng, and Chi-Wing Fu. Point cloud upsampling via disentangled refinement. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 344–353, 2021. 1, 5, 6, 7

[9] Xinhai Liu, Xinchen Liu, Zhizhong Han, and Yu-Shen Liu. SPU-Net: Self-supervised point cloud upsampling by coarse-to-fine reconstruction with self-projection optimization, 2020. 2

[10] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. 2, 3, 4, 5

[11] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 165–174, 2019. 2, 3

[12] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 4

[13] Guocheng Qian, Abdulellah Abualshour, Guohao Li, Ali Thabet, and Bernard Ghanem. PU-GCN: Point cloud upsampling using graph convolutional networks. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11678–11687, 2021. 1, 5, 6

[14] Yue Qian, Junhui Hou, Sam Kwong, and Ying He. Deep magnification-flexible upsampling over 3d point clouds. *IEEE Transactions on Image Processing*, 30:8354–8367, 2021. 2

[15] David Stutz and Andreas Geiger. Learning 3d shape completion from laser scan data with weak supervision. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1955–1964, 2018. 2, 3

[16] David Stutz and Andreas Geiger. Learning 3d shape completion under weak supervision. *CoRR*, abs/1805.07290, 2018. 4

[17] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019. 4

[18] Huikai Wu, Junge Zhang, and Kaiqi Huang. Point cloud super resolution with adversarial residual graph networks. *arXiv preprint arXiv:1908.02111*, 2019. 5

[19] Shuquan Ye, Dongdong Chen, Songfang Han, Ziyu Wan, and Jing Liao. Meta-PU: An arbitrary-scale upsampling network for point cloud. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2021. 2, 5, 6, 7

[20] Wang Yifan, Shihao Wu, Hui Huang, Daniel Cohen-Or, and Olga Sorkine-Hornung. Patch-based progressive 3d point set upsampling. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5951–5960, 2019. 1, 5, 6, 7

[21] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. PU-Net: Point cloud upsampling network. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2018. 1, 5, 6

[22] Yifan Zhao, Le Hui, and Jin Xie. *SSPU-Net: Self-Supervised Point Cloud Upsampling via Differentiable Rendering*, page 2214–2223. Association for Computing Machinery, New York, NY, USA, 2021. 1, 2