

# Neural Points: Point Cloud Representation with Neural Fields for Arbitrary Upsampling

Wanquan Feng<sup>1</sup> Jin Li<sup>2</sup> Hongrui Cai<sup>1</sup> Xiaonan Luo<sup>2</sup> Juyong Zhang<sup>1\*</sup>

<sup>1</sup>University of Science and Technology of China <sup>2</sup>Guilin University Of Electronic Technology

lcfwq@mail.ustc.edu.cn, 20032201014@mails.guet.edu.cn, hrcai@mail.ustc.edu.cn,  
luoxn@guet.edu.cn, juyong@ustc.edu.cn

## Abstract

In this paper, we propose Neural Points, a novel point cloud representation and apply it to the arbitrary-factored upsampling task. Different from traditional point cloud representation where each point only represents a position or a local plane in the 3D space, each point in Neural Points represents a local continuous geometric shape via neural fields. Therefore, Neural Points contain more shape information and thus have a stronger representation ability. Neural Points is trained with surface containing rich geometric details, such that the trained model has enough expression ability for various shapes. Specifically, we extract deep local features on the points and construct neural fields through the local isomorphism between the 2D parametric domain and the 3D local patch. In the final, local neural fields are integrated together to form the global surface. Experimental results show that Neural Points has powerful representation ability and demonstrate excellent robustness and generalization ability. With Neural Points, we can resample point cloud with arbitrary resolutions, and it outperforms the state-of-the-art point cloud upsampling methods. Code is available at <https://github.com/WanquanF/NeuralPoints>.

## 1. Introduction

Point cloud, which is the most fundamental and popular representation of 3D scenes, has been widely used in many applications like 3D reconstruction [11, 14, 24, 33], virtual/augmented reality [10, 34] and autonomous driving [16, 36]. In the traditional representation of point cloud, each point only represents a position in the 3D space, and it can be further extended to represent a local plane if its normal vector is assigned. Therefore, the representation ability of point cloud is still limited by its resolution. Although point cloud upsampling methods [17, 18, 38, 43] have been proposed to improve its representation ability, their strategy is still a “discrete-to-discrete” manner and can not overcome the limitation of current point cloud representation.

In this work, we propose *Neural Points*, a novel point

cloud representation, which significantly improves the representation ability and can be naturally applied to the upsampling task. Different from traditional representation, each point of Neural Points encodes a local surface patch represented via neural fields. Specifically, each point-wise surface patch is represented as a local isomorphism between the 2D parametric domain and the 3D local surface patch, and the isomorphism is implicitly represented via neural fields. Thanks to its continuous nature, free of the limitation of finite resolution, powerful representation ability to involve more shape information, neural fields enable Neural Points several advantages over traditional discrete point cloud representation. Meanwhile, the trained model of neural fields is shared for all local patches, and thus the storage overhead of Neural Points is quite small.

Unlike some existing methods which represent the whole surface model via one neural field [26, 41], both the neural fields and the point features passed into Neural Points are all local. With this design, our Neural Points representation shows several advantages including excellent ability to express details, strong generalization ability, and low need for training data. Specifically, we employ local neural fields to construct the continuous bijective mapping between the 2D parametric domain and the 3D local surface patch, based on the fact that 3D local patch on the 2D manifold is isomorphic to a 2D simply connected disk. With the neural field, each 3D local patch can be viewed as a parametric surface defined in a 2D parametric domain. To this end, we employ an encoder to extract point-wise local features and take the local feature as a part of the input passed into the neural field network. With the local shape information, the neural field can represent the surface patch well, and the trained network model is shared by all local patches of all 3D models. In the final, we design an integration strategy to integrate all the local neural fields together to form the final global shape. The model of Neural Points is trained with geometric surface in high resolution with rich geometric details, such that the trained Neural Points model can represent rich geometric shapes.

\*Corresponding author

As for the point cloud upsampling task, its target is to predict the upsampled point cloud with higher resolution from the given low-resolution input such that it can better capture the geometric features of the underlying continuous surface. Existing point upsampling methods [17, 18, 21, 32, 38, 42–44] usually employ a “discrete-to-discrete” manner. Dissimilarly, we convert the input point cloud representation to continuous Neural Points, and then apply arbitrary-factored sampling.

The experimental results show that Neural Points can produce high-quality continuous surfaces and overcome the limitation of point resolution well. For the upsampling task, our strategy outperforms the state-of-the-art methods. In summary, the contributions of this work include:

- We propose Neural Points, a novel point cloud representation that can be naturally applied to the upsampling task, with low storage overhead.
- We employ the neural implicit functions along with the deep local features of surface patches to represent the local neural fields, and design an integration strategy to form the final global shape.
- Evaluation results demonstrate that the Neural Points representation has excellent robustness and generalization ability for various inputs on the upsampling task.

## 2. Related Works

**Neural Implicit Function.** Neural implicit representation has recently shown promising advantages in some applications over discrete scene representations, such as point clouds, meshes and voxel grids, owing to its continuous nature and free of the limitation of finite resolution. It has also achieved great performance in many applications including 3D shape representation [4, 22, 26], novel view synthesis [23, 35] and multi-view reconstruction [25, 41].

For 3D shape representation, DeepSDF [26] and ONet [22] proposed to use neural implicit function to predict the signed distance and occupancy probability, respectively. However, these conventional frameworks are weak in representing complex shapes. To alleviate this issue, some recent works focus on geometric detail preservation and local shape reduction. IF-Nets [5] extracted learnable multi-scale features, which encodes local and global properties of 3D shape. LGCL [40] represented 3D shape as zero-surfaces with local latent codes, leading each local SDF responsible for a part of the shape. Due to local feature learning, these methods represent higher-quality details than common methods which use a single vector to encode a shape. In our method, we also devise a novel and effective strategy to extract deep local features from point cloud.

Neural implicit functions have also been used to represent 2D images. LIIF [3] took a pixel coordinate and its surrounding feature as input to predict the corresponding

RGB value. It demonstrates excellent super-resolution effect, building a bridge between discrete and continuous representation in the 2D domain. Different from regular 2D image, irregular point cloud is a discrete representation of 3D surface embedded in 2D manifold. To handle the irregular representation, we construct the mapping between the irregular local surface patch and the regular 2D domain.

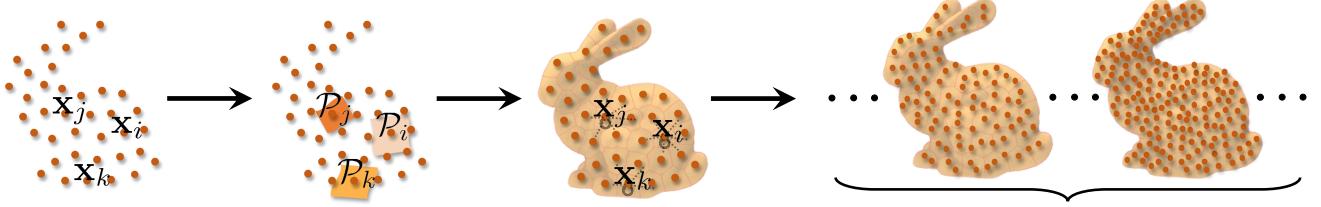
**Optimization Based Point Upsampling.** In optimization based point upsampling methods, shape priors like global structures and local smoothness are formulated as objective energy to constrain the results. Alexa *et al.* [2] employed Voronoi diagram and interpolated points at vertices of the Voronoi diagram in local tangent space as upsampled points. Then Lipman *et al.* [20] proposed a parameter-free method based on the locally optimal projection operator (LOP) for point sampling and surface reconstruction. Later, weighted LOP [12] conducted an iterative normal estimation process to consolidate the upsampled point. EAR [13] sampled points away from edges and progressively approaching edges and corners. Then a point-set consolidation method [39] proposed to fill large holes.

**Learning Based Point Upsampling.** Considering that the point upsampling task is an ill-posed problem, learning priors from dataset is a natural way to tackle it. In recent years, point-based network structures [7, 15, 19, 28, 29, 37] have been successfully employed to solve point cloud related tasks and achieved state-of-the-art performance. PU-Net [43] was the first point upsampling network. MPU [38] was a progressive upsampling method that achieved great performance for large upsampling factors. PU-GAN [17] was the first the GAN [9] based structure to synthesize uniformly distributed points. While the above mentioned works are mainly focused on the network design, PUGeo-Net [31] and MAFU [32] utilized local differential geometry constraints to improve the upsampling results. Meta-PU [42] firstly upsampled to a large resolution and then downsampled to a target resolution. PU-GCN [30] improved the performance by proposing a novel NodeShuffle module. Dis-PU [18] employed a global refinement module at the end of the pipeline instead of using a single upsampling module. However, all the above methods adopt the strategy to directly predict a denser point cloud from the input sparse point cloud. Unlike all these methods, we utilize neural fields to represent the high-resolution surface, which has a more powerful representation ability and can then be sampled in arbitrary resolutions.

## 3. Neural Points

### 3.1. Overview: Point based Representation

Point cloud  $\mathcal{X} = \{\mathbf{x}_i \in \mathbb{R}^3\}_{i=1}^I$  is the discrete representation of its underlying continuous surface  $\mathcal{S}$ . For the traditional point cloud representation, where each point only represents a 3D position, its representation ability totally



Input Point Cloud

Local Neural Fields

Neural Points

Arbitrary-factored Resampling Point Clouds

Figure 1. Algorithm pipeline. For the input point cloud, a discrete point-wise local patch is represented via local continuous neural fields, and the global continuous Neural Points surface is constructed by integrating all the local neural fields. Arbitrary resolutions of point cloud can be generated by sampling on the constructed continuous Neural Points surface.

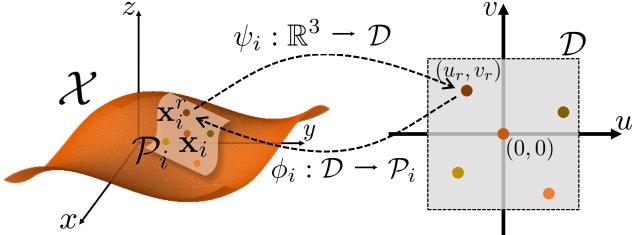


Figure 2. Local neural field and the bijective mapping function between isomorphic 3D and 2D domain.  $\mathcal{P}_i$  is the 3D local surface patch around the center point  $\mathbf{x}_i$ , and  $\mathcal{D}$  is the 2D parametric domain.  $\phi_i$  and  $\psi_i$  are the mappings that constructs the correspondence between  $(u_r, v_r) \in \mathcal{D}$  and  $\mathbf{x}_i^r \in \mathcal{P}_i$ .

depends on its resolution. Therefore, one direct strategy to improve its representation ability is to do upsampling:

$$\mathcal{S} \xrightarrow{\text{Discretize}} \{\mathbf{x}_i\}_{i=1}^I \xrightarrow{\text{Upsample}} \{\mathbf{x}_i^r\}_{i=1}^{I,R} \subset \mathcal{S}, \quad (1)$$

which was studied in the point cloud upsampling works [18, 43]. However, the upsampling manner in Eq. (1) is discrete-to-discrete, where the upsampled result is still discrete and limited by the resolution.

In this work, we propose *Neural Points*, a novel point cloud representation for arbitrary upsampling, which has better representation ability than the traditional point cloud. Neural Points representation employs the discrete-to-continuous strategy, which is totally different from the form of Eq. (1). Given the input point cloud  $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^I$ , we describe the underlying continuous surface

$$\mathcal{S} \xrightarrow{\text{Discretize}} \{\mathbf{x}_i\}_{i=1}^I \xrightarrow{\text{Neural Points}} \mathcal{S}' \approx \mathcal{S}, \quad (2)$$

where  $\mathcal{S}'$  is a continuous surface represented by Neural Points. With the form in Eq. (2), our proposed framework can overcome the limitation of point cloud resolution and achieve arbitrary-factored point cloud sampling on  $\mathcal{S}'$ . In the following, we will describe the algorithm details of our Neural Points representation.

### 3.2. Pipeline

The whole pipeline is given in Fig. 1. (1) Given the input point cloud, we first construct local neural fields for each local patch, which is based on local parameterization; (2) The local neural fields are integrated together to form the global

shape; (3) With the constructed continuous neural representation, we can resample an arbitrary number of points. In the following, we introduce the details of each component.

**Local Neural Fields.** We employ local neural fields to describe the underlying continuous surface. The input point cloud is still denoted as  $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^I$ . Like in [31], we take  $\{\mathbf{x}_i\}_{i=1}^I$  as the center points to divide the surface into overlapping local patches  $\{\mathcal{P}_i\}_{i=1}^I \subset \mathbb{R}^3$ . At each point  $\mathbf{x}_i$ , the 3D local patch  $\mathcal{P}_i$  is isomorphic to the 2D parametric domain  $\mathcal{D} \subset \mathbb{R}^2$  (we use  $\mathcal{D} = [-1, 1]^2$  in our work), which means that we can construct a bijective mapping between them:

$$\phi_i : \mathcal{D} \rightarrow \mathcal{P}_i, \quad \psi_i : \mathbb{R}^3 \rightarrow \mathcal{D},$$

where  $\psi_i|_{\mathcal{P}_i} = \phi_i^{-1}$ . The illustration of the neural field is shown in Fig. 2. Given any 2D sampling point  $(u_r, v_r) \in \mathcal{D}$ , we can compute  $\mathbf{x}_i^r = \phi_i(u_r, v_r) \in \mathcal{P}_i$  as its corresponding 3D sampling point. Similarly, for any 3D point  $\mathbf{x}_i^r \in \mathcal{P}_i$ , we can compute its corresponding 2D coordinate  $(u_r, v_r) = \psi_i(\mathbf{x}_i^r) \in \mathcal{D}$ . Furthermore, we can compute the point normal  $\mathbf{n}_{\mathbf{x}_i^r}$  at point  $\mathbf{x}_i^r$  via:

$$\mathbf{n}_{\mathbf{x}_i^r} = \nabla_u \phi_i \times \nabla_v \phi_i,$$

and then normalize it to unit length.

**Neural Fields Integration.** Although the local neural fields  $\{\phi_i\}_{i=1}^I$  can construct correspondence between 2D parametric domain and 3D local patches, different local neural fields are defined in different local coordinate systems. As a result, we need to integrate them together to obtain a globally continuous neural field based surface.

For  $\forall \mathbf{x} \in \mathbb{R}^3$ , we aim to pull  $\mathbf{x}$  onto the neural fields based surface. We find the nearest center points in  $\{\mathbf{x}_i\}_{i=1}^I$  to  $\mathbf{x}$  and denote the set of indexes of the neighbour points as  $\mathcal{N}(\mathbf{x})$ . For  $\forall k \in \mathcal{N}(\mathbf{x})$ , a combination weight is computed according to the distance from  $\mathbf{x}$  to  $\mathbf{x}_k$ :

$$w_k = e^{-\alpha_1 \|\mathbf{x} - \mathbf{x}_k\|_2^2}, \forall k \in \mathcal{N}(\mathbf{x}). \quad (3)$$

With  $w_k$ , the point on the neural fields based surface can be computed by:

$$\rho(\mathbf{x}) = (\sum_{k \in \mathcal{N}(\mathbf{x})} w_k \cdot \phi_k \circ \psi_k(\mathbf{x})) / (\sum_{k \in \mathcal{N}(\mathbf{x})} w_k), \forall \mathbf{x} \in \mathbb{R}^3. \quad (4)$$

Specifically, the input point  $\mathbf{x}$  is mapped to 2D parametric domain through the neighbour neural fields as  $\psi_k(\mathbf{x}) \in \mathcal{D}$ , and then map the 2D point back to 3D coordinate as  $\phi_k \circ \psi_k(\mathbf{x}) \in \mathcal{P}_k$ . In this way, a neural fields based continuous surface is constructed.

The normal at point  $\rho(\mathbf{x})$  is computed as:

$$\mathbf{n}_{\rho(\mathbf{x})} = \left( \sum_{k \in \mathcal{N}(\mathbf{x})} w_k \cdot \mathbf{n}_{\phi_k \circ \psi_k(\mathbf{x})} \right) / \left( \sum_{k \in \mathcal{N}(\mathbf{x})} w_k \right), \forall \mathbf{x} \in \mathbb{R}^3, \quad (5)$$

and the combination vector is normalized to unit length. Note that the orientations of  $\{\mathbf{n}_{\phi_k \circ \psi_k(\mathbf{x})}\}_{k \in \mathcal{N}(\mathbf{x})}$  are adjusted to be along the same direction before computing the weighted sum.

**Point Cloud Sampling.** We can also resample the point cloud from the neural fields represented continuous surface. Considering that the point cloud sampling process is essentially applied on the 2D manifold, we start the sampling operation in the 2D parametric domain  $\mathcal{D}$ .

We sample points uniformly in the 2D parametric domain  $\mathcal{D}$ , and the 2D sampled points are mapped onto the 3D local patch. Specifically, for center point  $\mathbf{x}_i$ , we uniformly sample  $R$  points  $\{(u_r, v_r) \in \mathcal{D}\}_{r=1}^R$  and then map them to 3D as  $\{\mathbf{x}_i^r \in \mathcal{P}_i\}_{r=1}^R$ . For the whole input point cloud, we obtain the union of sampled points from all patches:

$$\mathcal{X}_R = \bigcup_{i=1}^I (\{\mathbf{x}_i^r \in \mathcal{P}_i\}_{r=1}^R) = \{\mathbf{x}_i^r \in \mathcal{P}_i\}_{i,r=1}^{I,R}.$$

Then we can uniformly sample  $J$  points from  $\mathcal{X}_R$ :

$$\mathcal{Y}^* = \{\mathbf{y}_j^* \in \bigcup_{i=1}^I \mathcal{P}_i\}_{j=1}^J,$$

where the value of  $J$  is arbitrary in the training and inferring stages. Then we pull  $\mathbf{y}_j^*$  onto the Neural Points surface with Eq. (4) as:

$$\mathcal{Y} = \{\mathbf{y}_j = \rho(\mathbf{y}_j^*)\}_{j=1}^J.$$

### 3.3. Network Structure

In this part, we introduce the network structures, including the local feature extraction and the local neural fields.

**Local Feature Extraction.** As discussed above, for each center point  $\mathbf{x}_i$ , we extract a local feature as a part of the input into the neural field  $\phi_i$ , which we denote as  $\mathbf{f}_i$ . For each  $\mathbf{x}_i$ , we first extract its neighbour points  $\{\mathbf{x}_k\}_{k \in \mathcal{N}(\mathbf{x}_i)}$  and decentralize to  $\{\mathbf{x}_k - \mathbf{x}_i\}_{k \in \mathcal{N}(\mathbf{x}_i)}$ . Then we apply the DGCNN [37] backbone to extract feature on the point set with  $|\mathcal{N}(\mathbf{x}_i)|$  points. Specifically, we employ several EdgeConv layers with dynamic graph update. The features of each layer are concatenated and then passed into another EdgeConv layer and max-pooling layer to get  $\mathbf{f}_i^*$ . For each

$i$ , we concatenate  $\mathbf{f}_i^*$  with the local pooling of its neighbors to get the final local feature:

$$\mathbf{f}_i = \mathbf{f}_i^* \oplus \text{MaxPool}\{\mathbf{f}_k^*\}_{k \in \mathcal{N}(\mathbf{x}_i)},$$

where  $\oplus$  denotes the concatenation operation.

**Structure of Neural Fields.** We employ the MLP-based network equipped with the ReLU activation layers to represent  $\phi_i$ . The implicit functions are shared by all patches of all 3D models, and they are denoted as  $\Phi$ . As we mentioned above, the input of  $\Phi$  should contain the local shape information involved in  $\mathbf{f}_i$  and the 2D parametric coordinates as the querying point.

For the 2D querying coordinates passed into  $\phi_i$ , we apply position encoding as the design in [23] and denote the position encoding function as  $\gamma$ . The position code is concatenated with the local feature as the input of the neural implicit function. Specifically, we can formulate  $\Phi$  as:

$$\Phi((u_r, v_r), \mathbf{f}_i, \mathbf{x}_i) = \mathbf{x}_i + \theta_\Phi(\gamma(u_r, v_r) \oplus \mathbf{f}_i),$$

where  $\theta_\Phi$  is the MLP in  $\Phi$ . Naturally, for each  $i$ , we can formulate  $\phi_i(\cdot) = \Phi(\cdot, \mathbf{f}_i, \mathbf{x}_i)$ .

Another key function is  $\psi_i$ , who is the extension of  $\phi_i^{-1}$ . In our implementation,  $\psi_i$  is defined as:

$$\psi_i(\mathbf{x}) = \phi_i^{-1}(\text{Proj}(\mathbf{x}, \phi_i(\mathcal{D}))), \forall \mathbf{x} \in \mathbb{R}^3, \quad (6)$$

where we use  $\text{Proj}(\mathbf{x}, \phi_i(\mathcal{D}))$  to denote the nearest point in  $\phi_i(\mathcal{D})$  to  $\mathbf{x}$ . The advantage of this definition is that we can formulate  $\phi_i \circ \psi_i$  as:

$$\phi_i \circ \psi_i(\mathbf{x}) = \text{Proj}(\mathbf{x}, \phi_i(\mathcal{D})), \forall \mathbf{x} \in \mathbb{R}^3, \quad (7)$$

which is used in Eq. (4).

At last, we explain our projection process in Eq. (6) and Eq. (7). In our implementation, we use the approximation for  $\phi_i(\mathcal{D})$  as:

$$\phi_i(\mathcal{D}) \approx \{\mathbf{x}_i^r\}_{r=1}^R.$$

Then, the projection operation should be formulated from a 3D point and a point set. For the convenience of description, we consider a 3D point denoted as  $\mathbf{p}$  and a point set denoted as  $\mathcal{Q} = \{\mathbf{q}_t\}_{t=1}^T$ . We find the index set of neighbor points of  $\mathbf{p}$  in  $\mathcal{Q}$  and denote it as  $\mathcal{N}(\mathbf{p}; \mathcal{Q})$ . The projection from  $\mathbf{q}$  to  $\mathcal{Q}$  is formulated as:

$$\text{Proj}(\mathbf{p}, \mathcal{Q}) = \left( \sum_{k \in \mathcal{N}(\mathbf{p}; \mathcal{Q})} w_k \cdot \mathbf{q}_k \right) / \left( \sum_{k \in \mathcal{N}(\mathbf{p}; \mathcal{Q})} w_k \right), \quad (8)$$

where  $w_k$  is computed as:

$$w_k = e^{-\alpha_2 \|\mathbf{p} - \mathbf{q}_k\|_2^2}, \forall k \in \mathcal{N}(\mathbf{p}; \mathcal{Q}). \quad (9)$$

Similar to Eq. (5), its normal  $\mathbf{n}_{\text{Proj}(\mathbf{p}, \mathcal{Q})}$  can be obtained.

### 3.4. Loss Function

In this part, we introduce the loss terms, including the constraints on surface shape, point normal, and the integration quality. The total loss is:

$$\mathcal{L} = \mathcal{L}_{\text{shape}} + \omega_1 \cdot \mathcal{L}_{\text{nor}} + \omega_2 \cdot \mathcal{L}_{\text{int}}.$$

Point clouds with higher resolution are used as ground truth in our current implementation, and other shape representations can also be used accordingly. We denote the ground truth point cloud as  $\mathcal{Z} = \{\mathbf{z}_j\}_{j=1}^L$  and denote the normal of  $\mathbf{z}_l$  as  $\mathbf{n}_{\mathbf{z}_l}$ . As described in Sec. 3.2, the output of the framework can be summarized as  $\mathcal{X}_R = \{\mathbf{x}_i^r; \mathbf{n}_{\mathbf{x}_i^r}\}_{i,r=1}^{I,R}$  and  $\mathcal{Y} = \{\mathbf{y}_j; \mathbf{n}_{\mathbf{y}_j}\}_{j=1}^J$ .

The supervision is discrete, but  $\mathcal{X}_R$  and  $\mathcal{Y}$  should be viewed as arbitrary sampling from continuous surface. To supervise a continuous surface with the discrete supervision, we do not employ the Chamfer loss which is based on closest point searching, though it was widely used in the previous point upsampling works [18, 30]. Instead, we employ the projection strategy described in Eq. (8). Generally, for two point clouds  $\mathcal{P} = \{\mathbf{p}_s\}_{s=1}^S$  and  $\mathcal{Q} = \{\mathbf{q}_t\}_{t=1}^T$ , we define their distance as:

$$d(\mathcal{P}, \mathcal{Q}) = \frac{1}{S} \sum_{s=1}^S \|\mathbf{p}_s - \text{Proj}(\mathbf{p}_s, \mathcal{Q})\|_2^2.$$

We also define their difference of point-wise normal as:

$$d_{\mathbf{n}}(\mathcal{P}, \mathcal{Q}) = \frac{1}{S} \sum_{s=1}^S \|\mathbf{n}_{\mathbf{p}_s} - \mathbf{n}_{\text{Proj}(\mathbf{p}_s, \mathcal{Q})}\|_2^2.$$

To constrain  $\mathcal{X}_R$  and  $\mathcal{Y}$  to be close to  $\mathcal{Z}$ , the loss term is:

$$\mathcal{L}_{\text{shape}} = d(\mathcal{X}_R, \mathcal{Z}) + d(\mathcal{Z}, \mathcal{X}_R) + d(\mathcal{Y}, \mathcal{Z}) + d(\mathcal{Z}, \mathcal{Y}).$$

To supervise the point-wise normal, the loss term is:

$$\mathcal{L}_{\text{nor}} = d_{\mathbf{n}}(\mathcal{X}_R, \mathcal{Z}) + d_{\mathbf{n}}(\mathcal{Z}, \mathcal{X}_R) + d_{\mathbf{n}}(\mathcal{Y}, \mathcal{Z}) + d_{\mathbf{n}}(\mathcal{Z}, \mathcal{Y}).$$

Furthermore, we also employ a loss term for the integration quality. The overlap shape from neighbour neural fields should be identical, and the constructed global neural points surface should cover the input surface. Based on these requirements, we design the loss term as:

$$\mathcal{L}_{\text{int}} = \sum_{j=1}^J \sum_{k \in \mathcal{N}(\mathbf{y}_j; \mathcal{X})} \|\mathbf{y}_j - \text{Proj}(\mathbf{y}_j, \{\mathbf{x}_k^r\}_{r=1}^R)\|_2^2.$$

## 4. Experiments

In this section, we give the implementation details, ablation studies, results, comparisons and the test on generalization and robustness.

### 4.1. Implementation Details

**Dataset.** We train and test our model on Sketchfab [1] dataset collected by PUGeo-Net [31], which contains 90 training and 13 testing models with rich geometry features. We train all the comparison methods with the same dataset for fair comparison. Similar to other point cloud upsampling methods, we employ Poisson disk sampling [6] algorithm to extract points on the models to obtain the input and ground truth of the upsampling algorithm. Specifically, we extract 10,000 points as the whole input point cloud and extract 40,000 and 160,000 points as the whole ground truth point cloud for 4 $\times$  and 16 $\times$  experiments. We also follow the previous point upsampling works to extract some anchor points and the neighboring local parts (subsets of the whole point cloud) as the input of the network instead of passing the whole model into the network. For all, we choose 1,000 anchors on each training model and 114 anchors on each testing model. During training, the point number of the input point clouds is set as 256.

To test the effectiveness of our trained model, we not only test on the testing set of Sketchfab, but also test on more unseen datasets without retraining our network. We test all the methods on dataset collected by PU-GAN [17], where the number of testing models is 27. The shape of the testing models in this dataset is relatively simple, and thus we extract 2,000 points on each model as the whole input point cloud. In addition to the synthetic data mentioned above, we further test on real captured data. We also evaluate our method on point clouds captured by depth sensor on iPhone X and LiDAR data from KITTI [8].

**Experimental Setting.** All the inputs passed into the network are normalized into the 3D unit ball. In all our experiments, we set  $R = \lfloor 4 \cdot J/I \rfloor$ . In the local feature extracting network, the number of neighboring point set is 10. For all the  $\text{Proj}(., .)$  operations, we set the number of neighboring points as 4. For the loss terms,  $\omega_1$  and  $\omega_2$  are set as 0.01 and 0.3, respectively.  $\alpha_1$  in Eq. (3) and  $\alpha_2$  in Eq. (9) are set as  $10^2$  and  $10^3$  respectively to compute the exponential weights. We employ 5 convolution layers in the DGCNN backbone and 3 linear layers in the MLP of  $\Phi$ . We set the batch size as 8 and train the network with 25,000 iterations for all. The learning rate starts as 0.01, and is multiplied by 0.5 every 1,250 iterations. For all the comparison methods, we train them with the same settings as our own method. The model is trained with PyTorch [27]. All the training and testing is conducted on a workstation with four 32G V100 GPUs, 32 Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz, and 128GB of RAM. Our trained network model can be applied to all point clouds. The storage overhead of Neural Points is to store the pre-trained model, whose total size is only 2.53MB (1.35MB for the local feature extracting network and 1.18MB for the Neural field MLP network.).

**Evaluation Metric.** Similar to recent point cloud upsam-

Ablation Settings	CD	HD	P2F
w/o input local KNN	2.49	9.54	3.03
w/o $\mathcal{L}_{\text{nor}}$	0.70	3.46	0.75
w/o $\mathcal{L}_{\text{int}}$	0.77	4.09	0.81
w/o integration full model	0.83	5.64	0.88
	<b>0.66</b>	<b>3.32</b>	<b>0.69</b>

Table 1. Results of the ablation study, with metrics CD( $\times 10^{-5}$ ), HD( $\times 10^{-3}$ ), and P2F( $\times 10^{-3}$ ).

Factor	Method	Sketchfab			PU-GAN		
		CD	HD	P2F	CD	HD	P2F
4x	PU-Net	5.93	4.98	4.51	23.61	13.91	10.02
	PU-GAN	3.30	3.45	3.61	16.79	9.36	7.04
	PU-GCN	2.85	3.21	2.79	14.74	11.97	6.36
	Dis-PU	2.61	3.25	2.67	13.79	11.83	7.14
	PUGeo-Net	2.28	<b>2.10</b>	1.04	11.26	3.54	2.14
	MAFU	2.26	2.31	0.99	11.18	3.74	2.09
	Ours	<b>2.17</b>	2.46	<b>0.93</b>	<b>8.17</b>	<b>3.08</b>	<b>1.59</b>
16x	PU-Net	5.25	5.82	5.99	20.70	15.49	12.16
	PU-GAN	2.97	3.99	3.76	11.89	10.81	7.48
	PU-GCN	2.35	3.84	3.02	11.37	12.69	6.95
	Dis-PU	2.36	3.79	3.31	12.75	13.65	8.09
	PUGeo-Net	0.83	3.50	0.97	3.58	7.14	1.94
	MAFU	0.80	3.57	0.95	3.51	7.20	1.90
	Ours	<b>0.66</b>	<b>3.32</b>	<b>0.69</b>	<b>3.35</b>	<b>6.52</b>	<b>1.49</b>

Table 2. Results and comparisons for 4x and 16x upsampling, with metrics CD( $\times 10^{-5}$ ), HD( $\times 10^{-3}$ ), and P2F( $\times 10^{-3}$ ).

pling works [18, 30], we employ Chamfer Distance (CD), Hausdorff Distance (HD), and Point-to-Surface (P2F) as the metrics. For all the metrics, the smaller the metric, the better the quality of the results.

## 4.2. Ablation Study

We conduct ablation studies to show how each component influences the final result. Specifically, we use the Sketchfab dataset as our benchmark. We mainly design the ablation study for the local feature extractor structure, the loss terms and the integration process. The quantitative results of the ablation study experiment are shown in Tab. 1. In the experiment of the 1-st row, we apply a DGCNN backbone for the whole input point cloud instead of the local KNN point set. We can see the quantitative result is quite poor, which verifies that it is necessary to utilize a shared backbone for KNN point sets and limit the size of the receptive field in the feature extractor. The 2-3-th rows show different loss term settings, where we remove the  $\mathcal{L}_{\text{nor}}$  and  $\mathcal{L}_{\text{int}}$  terms, respectively. We can see that removing any of the two terms would make the result to be not as good as the full model. Among them, removing the  $\mathcal{L}_{\text{int}}$  loss term will greatly reduce the algorithm performance. At last, we remove the integration process and all related algorithm settings, and show the results in the 4-th row. We can see that the result without the integration process is much worse than the full model.

	Method	Net	GAN	GCN	Dis	PUGeo	MAFU	Ours
4x	Size (Mb)	9.4	7.1	<b>1.8</b>	13.2	27.1	4.7	2.5
	Time (s)	<b>0.011</b>	<b>0.011</b>	0.012	0.047	0.014	0.014	0.015
16x	Size (Mb)	23.0	11.5	3.0	13.2	27.1	4.7	<b>2.5</b>
	Time (s)	0.120	0.046	0.039	0.085	<b>0.017</b>	0.018	<b>0.017</b>

Table 3. Inference time (forward once with batch size 1) and network size of methods (from left to right): PU-Net, PU-GAN, PU-GCN, Dis-PU, PUGeo-Net, MAFU, and ours.

## 4.3. Results and Comparisons

We compare our method with: PU-Net [43], PU-GAN [17], PU-GCN [30], Dis-PU [18], PUGeo-Net [31] and MAFU [32]. All the comparison methods are in a discrete-to-discrete manner, and are trained with the same training set with our method. For all methods, we train with 4x and 16x, respectively, with ground truth in the related resolution. For fairness, we share the same settings (batch size, iterations, learning rate, etc) among all methods.

The results and comparisons of 4x and 16x are given in Tab. 2, Fig. 3 and Fig. 5. Our method achieves the best performance both quantitatively and qualitatively. The results of PU-Net are messy overall. PU-GAN performs better than PU-Net but also generates strange noise and outliers. PU-GCN performs better than the above two methods and can preserve flat regions, but still produce some noise points in the feature-rich regions. We tried our best to finetune the training settings for Dis-PU [18]. On flat regions, Dis-PU performs quite well which benefits from its disentangled refinement scheme. However, in high-curvature regions, Dis-PU can not produce high-quality result. PUGeo-Net and MAFU achieves better result than all the other comparison methods. However, from the zoom-in view, we can find that the distribution of their upsampled points is not as smooth as our results. This is because they upsample each patch in a discrete and independent way. From the results, we can observe that our method generates results which are globally smooth and contain rich local geometric details. We list the inference time and network size of all methods in Tab. 3.

## 4.4. Generalization and Robustness

**Generalization to Unseen Data.** We test on another dataset which was commonly used in previous dataset, the dataset collected by PU-GAN [17]. Without retraining, we apply the trained models on the PU-GAN dataset again and show the visual and quantitative results. The results and comparisons are in Tab. 2 and Fig. 4. We can see that our model still achieves the best performance on both of quantitative results and visualization results, proving the good generalization ability of the Neural Points representation.

**Robustness to Noise.** We add Gaussian noise to the input, and show the results in Fig. 5. For the same inputs, our result keeps a good shape while the result of MAFU looks wrinkled. Our quantitative result is also better than MAFU.

**Robustness to Large Upsampling Factor.** We explore the

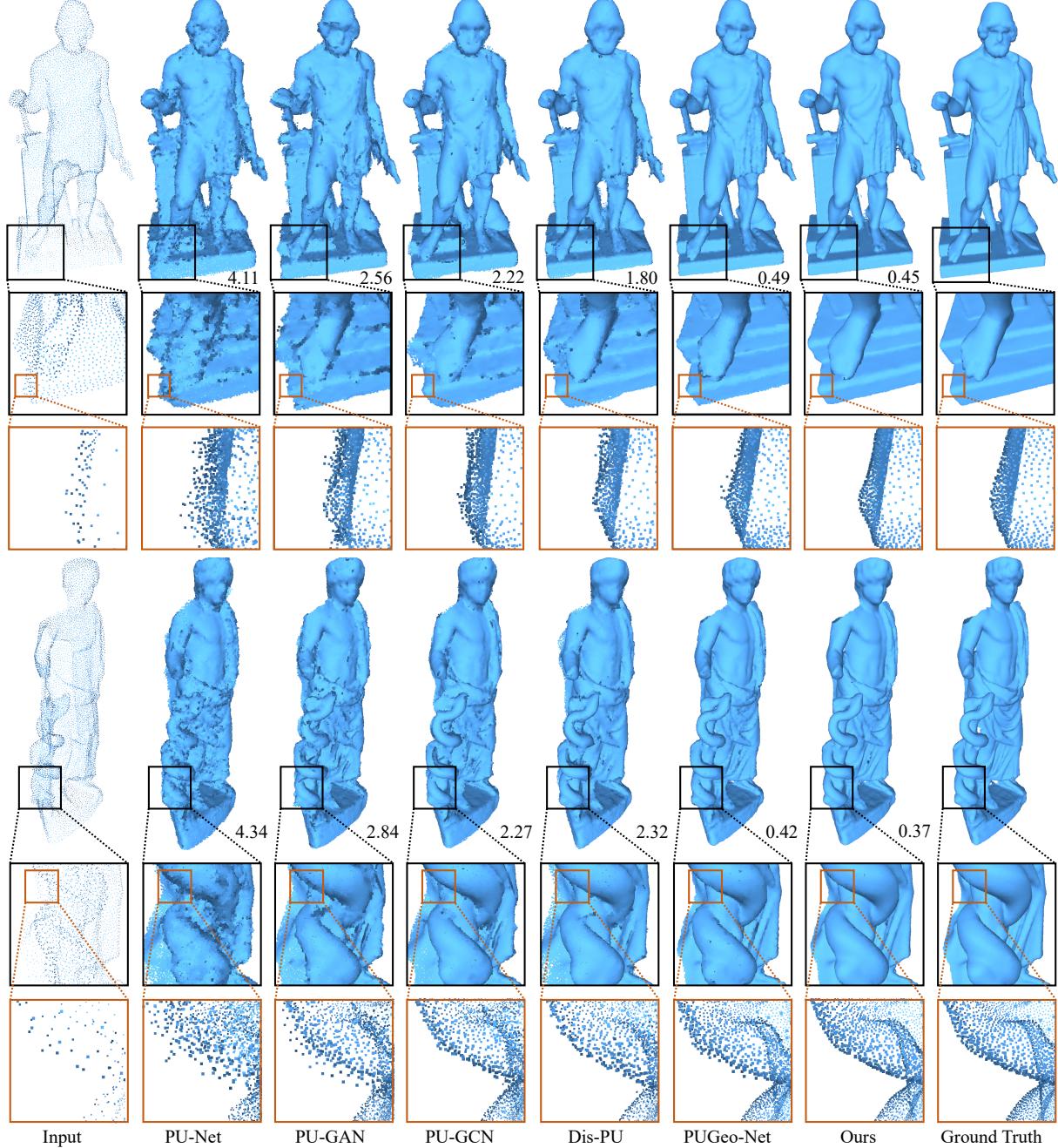


Figure 3. Results and comparison on the Sketchfab [1] dataset. The error metric  $CD \times 10^{-5}$  is also given in the bottom. For better visualization, we zoom-in some local parts of the results and choose the appropriate views to show the details.

upsampling effect of different methods with very large up-sampling factors. Notice that all the testing methods including our own method are trained with a  $16 \times$  supervision signal. We adjust the  $N$  in our method as  $256 \cdot I$  to obtain the  $256 \times$  result. For other methods, we apply the trained  $16 \times$  model for two times to obtain the  $256 \times$  results. The results are shown in Fig. 6. The visualization results of  $16 \times$  upsampling by different methods do not have too much difference, while the results of  $256 \times$  upsampling are quite dif-

ferent. After applying the model for two times, the results of PU-GAN, PU-GCN, Dis-PU and PUGeo-Net contain some flaws that are easy to observe. At the same time, our result still keeps in a good quality. Since Neural Points represents the surface in a continuous and resolution-free way, our result will not be affected by large sampling factors.

**Robustness to Real Captured Data.** We further compare our method with two most representative methods (Dis-PU [18] and PUGeo-Net [31]) on real captured data. Specif-

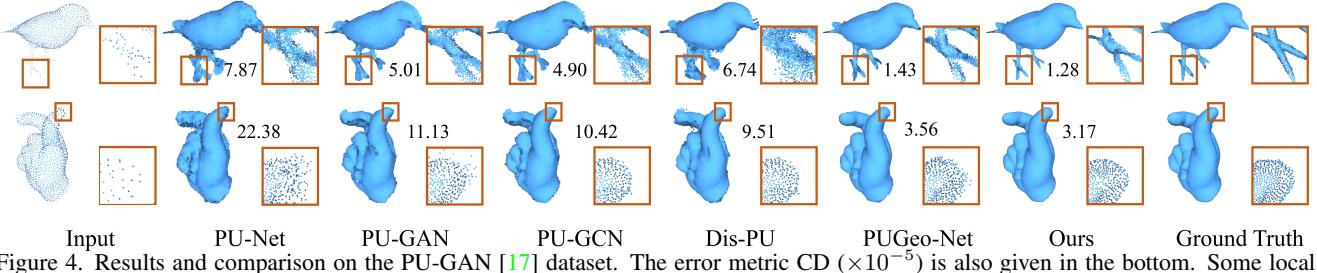


Figure 4. Results and comparison on the PU-GAN [17] dataset. The error metric CD ( $\times 10^{-5}$ ) is also given in the bottom. Some local parts are displayed for better comparison.

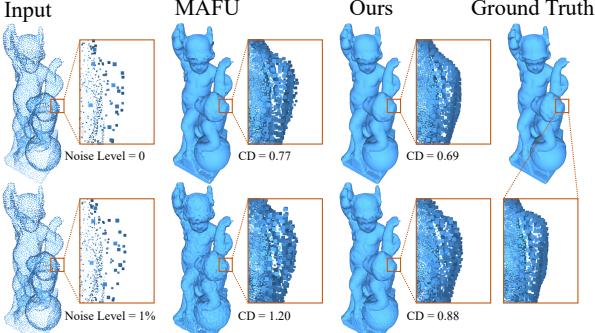


Figure 5. Comparison with MAFU & robustness test on noise.

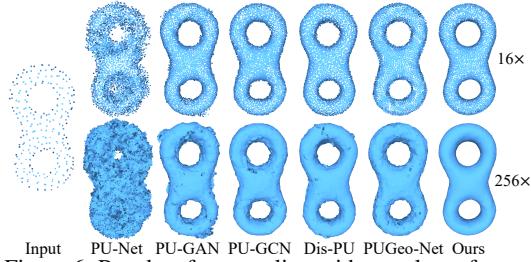


Figure 6. Results of upsampling with very large factors.

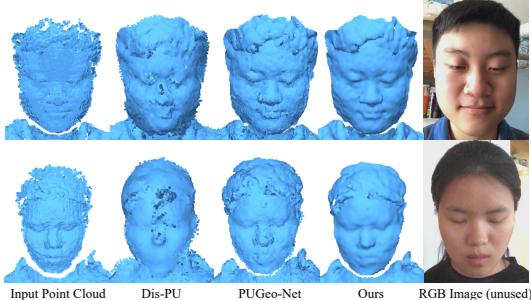


Figure 7. Comparisons on point cloud captured by depth camera. The RGB images are displayed for better visualization, which are not used in our method.

ically, we capture depth images of human face with the depth sensor equipped on iPhone X. Taking the point cloud converted from the depth image as input, we apply upsampling to the point clouds with different methods. As shown in Fig. 7, our method performs quite well, even the scanned point clouds contain lots of noise and bumpy local regions. In contrast, the results generated by Dis-PU [18] fail to preserve the geometry features and contain some artifacts. The

results of PUGeo-Net [31] preserve some geometric features of the input point cloud, but we can still observe some artifacts in their results. This test verifies the robustness and effectiveness of Neural Points to real captured data.



Figure 8. Result on LiDAR data from KITTI dataset.

**Robustness to LiDAR-scanned Data.** We show the test result on LiDAR data from KITTI [8] in Fig. 8, and we can also achieve reasonable upsampling result.

## 5. Conclusion&Limitation

We proposed Neural Points, a novel point cloud representation for arbitrary upsampling, where each point represents a local continuous geometric shape via neural fields instead of only a position or a local plane in the 3D space. Neural Points can involve more shape information and thus have a stronger representation ability than the traditional point cloud. We trained Neural Points with surface containing rich geometric details, such that the trained model has enough expression ability for various shapes. Specifically, we extracted deep local features and constructed neural fields through the local isomorphism between the 2D parametric domain and the 3D local patch. The final global continuous surface is obtained by integrating the neural fields. The powerful representation ability, robustness and generalization ability of Neural Points have been verified by extensive experiments. The great performance on the upsampling task further verified its nice properties.

Currently, we only utilize the local geometry shape to train Neural Points, which might limit its applications to more broad areas. In the future, we plan to utilize other modalities like corresponding color images and textures, or global semantic structure to further improve its representation ability.

**Acknowledgement** This research was supported by National Natural Science Foundation of China (No. 62122071), the Youth Innovation Promotion Association CAS (No. 2018495), “the Fundamental Research Funds for the Central Universities”(No. WK3470000021).

## References

- [1] Sketchfab. <https://sketchfab.com>. 5, 7
- [2] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Cláudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 9(1):3–15, 2003. 2
- [3] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8628–8638, 2021. 2
- [4] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5939–5948, 2019. 2
- [5] Julian Chibane, Thimo Alldieck, and Gerard Pons-Moll. Implicit functions in feature space for 3d shape reconstruction and completion. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6968–6979, 2020. 2
- [6] Massimiliano Corsini, Paolo Cignoni, and Roberto Scopigno. Efficient and flexible sampling with blue noise properties of triangular meshes. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 18(6):914–924, 2012. 5
- [7] Wanquan Feng, Juyong Zhang, Hongrui Cai, Haofei Xu, Junhui Hou, and Hujun Bao. Recurrent multi-view alignment network for unsupervised surface registration. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10297–10307, 2021. 2
- [8] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *Int. J. Robotics Res.*, 32(11):1231–1237, 2013. 5, 8
- [9] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014. 2
- [10] Robert Held, Ankit Gupta, Brian Curless, and Maneesh Agrawala. 3d puppetry: a kinect-based interface for 3d animation. In *ACM Symposium on User Interface Software and Technology (UIST)*, pages 423–434, 2012. 1
- [11] Hugues Hoppe, Tony DeRose, Tom Duchamp, John Alan McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 71–78, 1992. 1
- [12] Hui Huang, Dan Li, Hao Zhang, Uri M. Ascher, and Daniel Cohen-Or. Consolidation of unorganized point clouds for surface reconstruction. *ACM Transactions on Graphics (TOG)*, 28(5):176, 2009. 2
- [13] Hui Huang, Shihao Wu, Minglun Gong, Daniel Cohen-Or, Uri M. Ascher, and Hao (Richard) Zhang. Edge-aware point set resampling. *ACM Transactions on Graphics (TOG)*, 32(1):9:1–9:12, 2013. 2
- [14] Michael M. Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG)*, 32(3):29:1–29:13, 2013. 1
- [15] Artem Komarichev, Zichun Zhong, and Jing Hua. A-CNN: annularly convolutional neural networks on point clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7421–7430, 2019. 2
- [16] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12697–12705, 2019. 1
- [17] Ruihui Li, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. PU-GAN: A point cloud upsampling adversarial network. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7202–7211, 2019. 1, 2, 5, 6, 8
- [18] Ruihui Li, Xianzhi Li, Pheng-Ann Heng, and Chi-Wing Fu. Point cloud upsampling via disentangled refinement. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 344–353, 2021. 1, 2, 3, 5, 6, 7, 8
- [19] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhuan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 828–838, 2018. 2
- [20] Yaron Lipman, Daniel Cohen-Or, David Levin, and Hillel Tal-Ezer. Parameterization-free projection for geometry reconstruction. *ACM Transactions on Graphics (TOG)*, 26(3):22, 2007. 2
- [21] Luqing Luo, Lulu Tang, Wanyi Zhou, Shizheng Wang, and Zhi-Xin Yang. PU-EVA: an edge-vector based approximation solution for flexible-scale point cloud upsampling. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16188–16197, 2021. 2
- [22] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4460–4470, 2019. 2
- [23] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*, pages 405–421, 2020. 2, 4
- [24] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011. 1
- [25] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3504–3515, 2020. 2
- [26] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning con-

- tinuous signed distance functions for shape representation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 165–174, 2019. 1, 2
- [27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, volume 32, pages 8024–8035, 2019. 5
- [28] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017. 2
- [29] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pages 5099–5108, 2017. 2
- [30] Guocheng Qian, Abdulellah Abualshour, Guohao Li, Ali K. Thabet, and Bernard Ghanem. PU-GCN: point cloud upsampling using graph convolutional networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11683–11692, 2021. 2, 5, 6
- [31] Yue Qian, Junhui Hou, Sam Kwong, and Ying He. Pugeonet: A geometry-centric network for 3d point cloud upsampling. In *European Conference on Computer Vision (ECCV)*, volume 12364, pages 752–769, 2020. 2, 3, 5, 6, 7, 8
- [32] Yue Qian, Junhui Hou, Sam Kwong, and Ying He. Deep magnification-flexible upsampling over 3d point clouds. *IEEE Trans. Image Process.*, 30:8354–8367, 2021. 2, 6
- [33] Gernot Riegler, Ali Osman Ulusoy, Horst Bischof, and Andreas Geiger. Octnetfusion: Learning depth fusion from data. In *IEEE International Conference on 3D Vision (3DV)*, pages 57–66, 2017. 1
- [34] José Miguel Santana, Jochen Wendel, Agustín Trujillo, José Pablo Suárez, Alexander Simons, and Andreas Koch. Multimodal location based services—semantic 3d city data as virtual and augmented reality. In *Progress in location-based services (LBS)*, pages 329–353. 2017. 1
- [35] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 1119–1130, 2019. 2
- [36] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark E. Campbell, and Kilian Q. Weinberger. Pseudolidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8445–8453, 2019. 1
- [37] 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):146:1–146:12, 2019. 2, 4
- [38] Yifan Wang, Shihao Wu, Hui Huang, Daniel Cohen-Or, and Olga Sorkine-Hornung. Patch-based progressive 3d point set upsampling. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5958–5967, 2019. 1, 2
- [39] Shihao Wu, Hui Huang, Minglun Gong, Matthias Zwicker, and Daniel Cohen-Or. Deep points consolidation. *ACM Transactions on Graphics (TOG)*, 34(6):176:1–176:13, 2015. 2
- [40] Shun Yao, Fei Yang, Yongmei Cheng, and Mikhail G Mozeirov. 3d shapes local geometry codes learning with sdf. In *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 2110–2117, 2021. 2
- [41] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Ronen Basri, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020. 1, 2
- [42] 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 (TVCG)*, 2021. 2
- [43] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-net: Point cloud upsampling network. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2790–2799, 2018. 1, 2, 3, 6
- [44] Yifan Zhao, Le Hui, and Jin Xie. Sppu-net: Self-supervised point cloud upsampling via differentiable rendering. In Heng Tao Shen, Yueling Zhuang, John R. Smith, Yang Yang, Pablo Cesar, Florian Metze, and Balakrishnan Prabhakaran, editors, *ACM Multimedia Conference*, pages 2214–2223, 2021. 2