PU-Flow: a Point Cloud Upsampling Network with Normalizing Flows

Aihua Mao, Zihui Du, Junhui Hou, *Senior Member, IEEE,* Yaqi Duan, Yong-jin Liu, *Senior Member, IEEE,* and Ying He

Abstract—Point cloud upsampling aims to generate dense point clouds from given sparse ones, which is a challenging task due to the irregular and unordered nature of point sets. To address this issue, we present a novel deep learning-based model, called PU-Flow, which incorporates normalizing flows and weight prediction techniques to produce dense points uniformly distributed on the underlying surface. Specifically, we exploit the invertible characteristics of normalizing flows to transform points between Euclidean and latent spaces and formulate the upsampling process as ensemble of neighbouring points in a latent space, where the ensemble weights are adaptively learned from local geometric context. Extensive experiments show that our method is competitive and, in most test cases, it outperforms state-of-the-art methods in terms of reconstruction quality, proximity-to-surface accuracy, and computation efficiency. The source code will be publicly available at https://github.com/unknownue/puflow.

Index Terms—Point cloud analysis, upsampling, normalizing flows, weight prediction.

1 Introduction

Point clouds, as one of the most accessible 3D data formats, have been used in a wide range of scenarios, including geometric analysis, robotic object detection and autonomous driving. With compact storage and flexible organization in representing diverse 3D objects of complex structures and geometry, point clouds have attracted increasing research interest. However, raw points produced from LiDAR sensors or Depth cameras are often sparse, noisy and non-uniform due to hardware limitation of 3D scanning devices. Many 3D analysis tasks, such as robotic perception, point rendering, and surface reconstruction, highly depend on the quality of input point clouds.

Therefore, point cloud upsampling, the ability to generate dense points from sparse input, is required for point cloud analysis. Early optimization-based methods [1], [2], [3] use shape priors to guide point generation. These methods work on smooth and well-distributed points but have difficulty in processing more complex geometry.

In recent years, deep neural networks (DNNs) have brought new insights into point cloud upsampling in a data-driven manner. Yu et al. [4] first introduced PU-Net to extract embedding features from multi-scale patches and expands upsampled points by multi-branch *multi-layer perceptrons* (MLPs). As the first end-to-end point cloud upsampling network, PU-Net [4] demonstrates the feasibility of learning-based methods. Thereafter, many representative approaches, including EC-Net [5], MPU [6], PU-GAN [7], PUGeo-Net [8] and MAFU [9], have been proposed to further improve the quality of point generation. However, these methods generate points simply through coordinate reconstruction. They may overemphasize the coordinate similarity between the sparse input points and ground-truth while omitting the underlying distribution of the model surface. Besides, as the upsampling factor is binding to the point encoding/decoding process [4], [6], [7], numerous parameters are required to offer variation for duplicated features (to avoid clustering effect) and preserve uniformity.

In this study, we present a generative pipeline for point cloud upsampling. Similar to image super-resolution techniques [10], [11], we produce new points by weighted interpolation among local neighboring points. Particularly, there are two modules in our pipeline, the point transformer and the weight estimator. The point transformer formulates the transformation of point features between Euclidean space and latent space by leveraging normalizing flows (NFs). We propose to perform weighted interpolation in latent space, with the weights adaptively predicted by the weight estimator, as illustrated in Fig. 1.

NFs are known to be an invertible generative framework, which parametrizes a bijective mapping of a simple distribution into a more complex distribution. Thus, arbitrary manipulations in latent space reflects a bijective change in Euclidean space. By taking advantage of the invertibility of flows, we formulate the point encoding and decoding processes into a shared network. Through this way, we do not need a specific decoder for coordinate reconstruction like previous works, which helps to avoid the reconstruction error and reduce network parameters.

Previous works generally expand points by feature replication, which may lead to cluster phenomenon (i.e. non-uniformity) in practise. By contrast, our method upsamples points by adaptively interpolating local neighbors under a

- *A. Mao, Z. Du, Y. Duan are with School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. E-mails: ahmao@scut.edu.cn, {csusami,csduanyaqi}@mail.scut.edu.cn.*
- *J. Hou is with the Department of Computer Science, City University of Hong Kong, Hong Kong, and also with the City University of Hong Kong Shenzhen Research Institute, Shenzhen 518057, China. E-mail: jh.hou@cityu.edu.hk.*
- *Y.-J. Liu is with BNRist, MOE-Key Laboratory of Pervasive Computing, Department of Computer Science and Technology, Tsinghua University, Beijing, China. E-mail: liuyongjin@tsinghua.edu.cn.*
- *Y. He is with School of Computer Science and Engineering, Nanyang Technological University, Singapore. E-mail: yhe@ntu.edu.sg.*

Corresponding authors: Aihua Mao and Junhui Hou

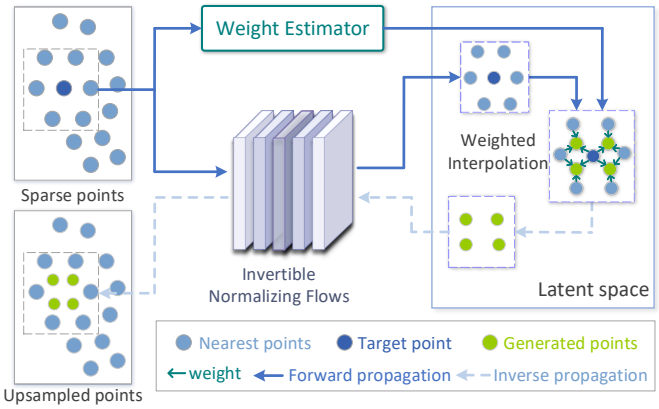


Fig. 1. A schematic illustration of our method. Input sparse patch (blue points) are first transformed into latent distribution $p_\vartheta(z)$ by forward propagation of normalizing flows. Then, we produce new latent points $\hat{z}$ (green points) by weighted interpolation of existing neighbors, where weights (green arrows) are predicted by a weight estimator. Finally, we transform $\hat{z}$ back into Euclidean space by inverse propagation and obtain upsampled patch. For simplicity, we only show this process for a single point.

prior distribution, where point variations are naturally introduced during the interpolation process. Therefore, there is no longer need to design extra modules to ensure point diversity, such as code assignment [6], [7] and multi-branch MLPs [4].

Our upsampling pipeline is designed to formulate the point transformation and weight estimation processes into two separated branches. On one hand, this design decouples the functionality of the point transformer and weight estimator and thus simplifies the optimization goal for each sub-network. On the other hand, it disentangles the task of expanding points from point decoding, such that the upsampling factor is not bind to the point transformation process.

In summary, the contributions of this paper are as follows:

- We innovatively formulate the 3D point cloud upsampling problem from the perspective of learned local interpolation in a latent space.
- We present a new upsampling pipeline, which cooperates the NFs and weight estimation techniques. By exploiting the invertibility of NFs, this pipeline ensures bijective mapping of point sets between coordinates and their latent representation.
- Through qualitative and quantitative evaluations on both synthetic and real-scanned datasets, we demonstrate the advantages of our PU-Flow over state-of-the-art works.

2 Related Works

2.1 Optimization-based upsampling methods

A number of optimization-based methods for point cloud upsampling or consolidation have been proposed over the past decade. Alexa et al. [1] computed a Voronor diagram on underlying surface by *moving least squares* methods, and generated points at vertices of the diagram. Lipman [2] introduced *locally optimal projection* (LOP) operator, a parametrization-free approach, for point resampling and surface reconstruction. Successively, Huang et al. [12] designed a weighted LOP operator to further consolidate the ability to handle sharp edges, outliers, and non-uniformity. Although the aforementioned methods can achieve good results, they are limited in processing smooth surfaces. Later, Huang et al. [3] developed an *edge-aware resampling* (EAR) method to progressively resample point set as well as approaching edge singularities, while its resampling effect is highly dependent on the accuracy of normal estimation. To complete large missing regions, Wu et al. [13] presented a consolidation method by introducing a *deep* representation for points.

In summary, optimization-based methods generally require geometric priors (e.g., normal) or assumption of smooth distribution, which restrict their application scope. In contrast, deep learning-based methods have more powerfull generalization ability without manual parameters tuning for different point sets.

2.2 Deep learning-based upsampling methods

In recent years, deep learning has been widely used in many fields of point clouds learning, including classification [14], [15], [16], [17], [18], segmentation [19], [20], [21], registration [22], [23], [24], denoising [25], [26], generation [27], completion [28], [29], [30], visualization [31], [32], etc. As the pioneer in applying neural networks to point cloud analysis, PointNet [14] and PointNet++ [15] propose to use shared MLP and symmetric functions as feature extractor.

Based on the architecture of PointNet++, Yu et al. [4] presented the first end-to-end deep learning framework, namely PU-Net, for point cloud upsampling. PU-Net [4] extracts different hierarchical features from multi-scale patches, and upsampling points are generated from multi-branch MLPs by coordinate reconstruction. It is optimized by a joint loss function including reconstruction and repulsion loss. PU-Net outperforms previous optimization-based methods, but the upsampling results still suffer from the cluster phenomenon and lack of fine-grained structure. Subsequently, Yu et al. [5] extended PU-Net with edge-aware loss function to consolidate edge smoothness. Yifan et al. [6] proposed a progressive upsampling network with dense connection and feature interpolation operator to bridge the upsampling unit on different levels. This network can adapt a large upsampling factor (e.g., 16×) by progressively feeding points to 2x upsampling unit. However, this mechanism requires step-by-step training for each unit, which is not flexible for tuning a large upsampling factor in practice. Subsequently, Li et al. [7] developed a generative adversarial network called PU-GAN, as well as a uniform metric to supervise upsampling quality. PU-GAN uses a self-attention unit to enhance feature integration and expand point features through the up–down–up pattern. Although PU-GAN [7] achieves impressive results on non-uniform data, there is no guarantee that the network can converge to the best performance stably in each training. Qian et al. [8] proposed PUGeo-Net to learn the intrinsic features of local geometry. They applied normal estimation to refine coordinate correction. Recently, Qian et al. [33] proposed PU-GCN, a graph-based convolutional approach.

The introduced NodeShuffle module can be integrated into existing upsampling pipeline. Li et al. [34] introduced DisPU, which incorporates a dense generator to generate a rough point set and a spatial refiner module to employ offset refinement.

Our work is more related to MAFU [9], which extends the idea of linear interpolation to create new points and employs a flexible training strategy to enable a flexible upsampling factor. Specifically, based on the linear approximation theory, MAFU [9] interpolates the coordinates of neighbouring points and predicts point-wise offsets to reduce high-order approximation errors. By contrast, our method performs interpolation in a latent space instead of the Euclidean space and is not required to refine coordinates.

Compared to previous works, our method unify the point encoding and decoding processes into a shared framework. By decoupling the task of of point transformer and weight estimator, the point transformation process is independent on the upsampling factor.

2.3 Normalizing Flows in Point Cloud Learning

Compared with familiar generative models, such as VAE and GAN, NFs represent another generative family that has existed for a long time but only becomes popular in recent years. NFs have many types of implementation, such as planar flows [35], [36], autoregressive models [37], [38], coupling-based flows [39], [40], [41], and continuous flows [42], [43]. Dinh et al. [39] introduced a coupling method to enable highly expressive transformations for flows, and this idea is further improved in [40], [41], [44]. With coupling architecture, one can apply arbitrary complex convolutions in inference, and the calculation of the Jacobian is extensively simplified. In recent years, continuous normalizing flows (CNF) has become a promising innovation on flow models. Based on the Neural ODE solver [45], CNF can achieve competitive performance to discrete flows [41] but with fewer parameters.

The flow-based methods have been successfully adapted to a wide range of generative scenarios, such as image generation [46], [47], cross-domain learning [48], [49], video prediction [50], graph generation [51], and audio synthesis [52], [53]. It is natural to generalize this idea to point clouds generation tasks. Yang et al. [43] presented PointFlow, a CNF-based framework, to learn a two-level hierarchy of distributions of given shapes. As the first flow-based network in point cloud learning, PointFlow can be further extends to a variety of applications [54], [55]. Then, Pumarola et al. [48] proposed C-Flow to explore the potential of bridging different domains (e.g., image and point clouds) with NFs. Klokov et al. [56] proposed a discrete PointFlow network to alleviate the slow convergence and difficult training issue of PointFlow [43]. Postels et al. [57] introduced the mixture model of NFs and showed the improved generative ability to single NF model. These works mainly concentrate on improving the flow-based generative capability. However, there are few works paying attention to real-world applications in point cloud analysis.

In this study, we take advantage of the invertible capacity of NFs to transform point clouds between Euclidean and latent spaces. To the best of our knowledge, no prior work has applied NFs to point cloud upsampling tasks.

3 Proposed Method

3.1 Overview

Given a sparse point set $\mathcal{P} = \{p_i \in \mathbb{R}^D\}_{i=1}^{N}$, our goal is to predict a dense point set $\hat{\mathcal{X}} = \{\hat{x}_i \in \mathbb{R}^D\}_{i=1}^{RN}$, where N is the number of points and R is upsampling factor. In this study, we only consider the coordinate of point attributes with $D = 3$. The generated point set $\hat{\mathcal{X}}$ is expected to meet the following requirements:

- $\hat{\mathcal{X}}$ should retain the geometric details represented by $\mathcal{P}$, while $\mathcal{P}$ is not necessary to be a subset of $\hat{\mathcal{X}}$.
- $\hat{\mathcal{X}}$ should be complete and uniformly distributed in both local and global areas.

In this study, we propose to utilize NFs to model the mapping of the point distribution between Euclidean space and latent space, which enables us to formulate the point cloud upsampling as the problem of learning point interpolation in latent space, as illustrated in Fig. 1. Specifically, given an input sparse point set $\mathcal{P}$, we first convert it to latent variable $z = f(\mathcal{P})$ with an invertible transformation defined by NFs. Then, we interpolate points in z and obtain dense latent variable $\hat{z}^R$, where the interpolation weights are learned from the point-wise local neighbours. Finally we transform $\hat{z}^R$ to a dense point cloud $\hat{\mathcal{X}}$ by the inverse mapping $\hat{\mathcal{X}} = f^{-1}(\hat{z}^R)$.

3.2 Flow-based Upsampling Method

A normalizing flow is a series of invertible transformations of distribution. It is generally used to model an intractable, complex distribution by a simple prior distribution. Formally, let $z \in \mathbb{R}^{N\times D}$ be a latent variable of base distribution $p_\vartheta(z)$ with the known density, i.e., $z \sim p_\vartheta(z)$. Given a dataset of observations $\mathcal{P}$, we aim to learn an invertible transformation $f_\theta(\cdot)$ to parameterize mapping from $\mathcal{P}$ to tractable density $p_\vartheta(z)$:

$$z = f_\theta(\mathcal{P}; \mathcal{C}), \tag{1}$$

where $\mathcal{C} = \psi(\mathcal{P})$, and $\psi(\cdot)$ is an arbitrary function that extracts conditional features from $\mathcal{P}$. Here, we refer to f_θ as conditional normalizing flows, which is generally parameterized by a neural network with parameters θ. Note that, f_θ is required to be a bijective transformation, which indicates that the dimension of points remains unchanged during distribution transforms.

By exploring the geometric structure in local context of each point, we apply weighted interpolation over the k-nearest neighbors, producing the upsampled latent variables $\hat{z}^R \in \mathbb{R}^{RN\times D}$

$$\hat{z}_i^R = I_\theta(z_i, \mathcal{N}(p_i)), \tag{2}$$

where $\mathcal{N}(p_i)$ denotes the k-nearest neighbors of latent point p_i and I_θ represents the interpolation function. Given conditional features $\mathcal{C}$ and latent points $\hat{z}^R$, the inverse mapping $g_\theta(\cdot) = f_\theta^{-1}(\cdot)$ implicitly defines the point decoding process

$$\hat{\mathcal{X}} = g_\theta(\hat{z}^R; \mathcal{C}), \tag{3}$$

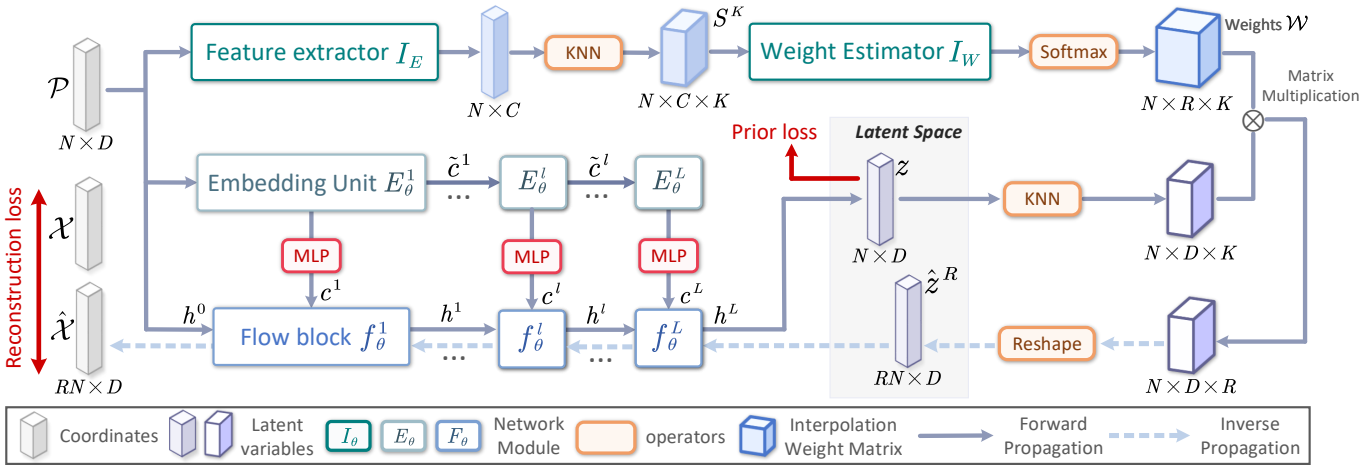


Fig. 2. Network architecture of the proposed method. Given a sparse patch $\mathcal{P}$ with N points, our method transforms $\mathcal{P}$ into latent distribution z by a sequence of flow blocks f_θ^l (forward propagation). Flow block f_θ^l is conditioning on features c^l, which is generated by point embedding unit E_θ^l. Then interpolation module I_θ predicts neighbour weights $\mathcal{W}$ by analyzing local context of k-nearest graph for each point. By interpolating latent variable z, we get dense latent distribution $\hat{z}^R$, with upsampling factor R. Upsampled patch is generated by converting $\hat{z}^R$ to Euclidean space by inverse flow F_θ^{-1} (inverse propagation). The network is trained end-to-end by minimizing prior loss $\mathcal{L}_{\text{prior}}$ of $\mathcal{P}$ and reconstruction loss $\mathcal{L}_{\text{rec}}$.

where $\hat{\mathcal{X}}$ is an upsampled estimation of $\mathcal{P}$. In contrast to decoding point by MLPs, utilizing the inverse mapping f_θ^{-1} can take advantage of the invertibility of NFs and help to reduce the reconstruction error and the number of network parameters.

As the single-layer flow model has limited non-linear capabilities, in practice, the flow network f_θ is composed of a sequence of L invertible layers. Let h^l be the output of the l-th flow layers, then h^{l+1} is defined as

$$h^{l+1} = f_\theta^{l+1}(h^l; \mathcal{C}^l), \tag{4}$$

where f_θ^l is the l-th flow layer, $h^0 = \mathcal{P}$, $h^L = z$, and $\mathcal{C}^l$ is the corresponding conditional features at the l-th layer. With the change of variable formula [40] and the chain rule, the probability density of the given input $\mathcal{P}$ can be computed as

$$\begin{aligned} \log p\left(\mathcal{P} \mid \mathcal{C}, \theta\right) &= \log p_\vartheta\left(f_\theta\left(\mathcal{P}; \mathcal{C}\right)\right) + \log \left| \det \frac{\partial f_\theta}{\partial \mathcal{P}}(\mathcal{P}; \mathcal{C}) \right| \\ &= \log p_\vartheta\left(f_\theta\left(\mathcal{P}; \mathcal{C}\right)\right) + \sum_{l=1}^{L} \log \left| \det \frac{\partial f_\theta^l}{\partial h^l}(h^l; \mathcal{C}^l) \right|, \end{aligned} \tag{5}$$

where the term $\left|\det \frac{\partial f_\theta}{\partial \mathcal{P}}(\mathcal{P}; \mathcal{C})\right|$ is the Jacobian determinant of transformation f_θ, measuring the volume changing [39] caused by f_θ. Generally, f_θ is trained by maximum likelihood principle with gradient descent techniques.

4 Network Architecture

According to the idea in Section 3.2, the overall architecture of PU-Flow depicted in Fig. 2 includes an invertible flow module F_θ (Section 4.2) and a point interpolation module I_θ (Section 4.3). Furthermore, to enhance the transformation capability of the flow layer, we introduce a hierarchical point embedding module E_θ (Section 4.1) to augment conditional features (i.e. $\mathcal{C}$ in Section 3.2) for F_θ in both forward and inverse propagation.

4.1 Hierarchical Point Embedding

Dimensional bottleneck. NFs are designed to ensure analytical invertibility. This fact poses a challenge that each flow component must output the same dimensionality as the input data (the dimension of raw point clouds is only $D = 3$). This constraint conflicts with the widely adopted intelligence of deep learning that learns features with a higher dimension than that of input data, resulting in limited transform capability of each flow block. This issue can be referred to as the *dimensional bottleneck* problem.

To alleviate this limitation, previous works, such as RealNVP [40] and Glow [41], propose to increase flows depth or use a multi-scale architecture coupled with squeezing operation. Simply increasing the depth of flows requires a large amount of parameters, leading to slow training speed and decreased training stability. Meanwhile, squeezing operator exchanges feature channel with spatial dimension, which is mainly designed for image manipulation. However, it is non-trivial to adopt squeezing to point cloud processing due to the unorder nature of a point set.

Hierarchical Embedding. Based on the above analysis, we propose a parallel sequence of embedding units to augment additional point-wise features for flow block f_θ^l as follows:

$$\tilde{c}^l = E_\theta^l(\tilde{c}^{l-1}), \tag{6}$$

where $\tilde{c}^l \in \mathbb{R}^{N \times C}$ denotes the high-level features outputted by l-th unit E_θ^l, and $\tilde{c}^0 = \mathcal{P}$. As E_θ is not a component of the NFs, it does not need to be invertible and can be arbitrary flexible architectures. One can consider the hierarchical embedding as a pattern of feature fusion.

In this study, the point embedding unit E_θ^l is constructed by a stack of densely connected graph convolutional layers [16], where the neighbour size of the graph is fixed to 16. It utilizes dense connections to enable richer contextual vision of multi-scale features. A more detailed description of unit E_θ^l can be found in supplementary material.

We further employ a simple MLP layer to obtain a specialized point-wise conditional features $\tilde{c}^l$ for each in-

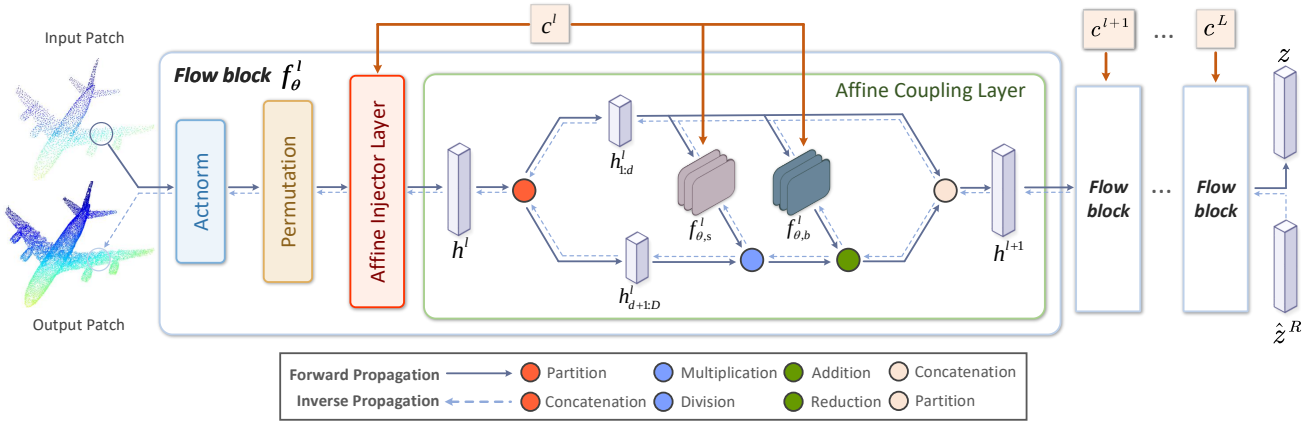


Fig. 3. Illustration of discrete flow block f_θ^l and conditional affine coupling layer. Each flow block consists of 4 flow components. The affine coupling/injector layer are conditioning on embedding features c^l from E_θ^l (Section 4.1) in both forward and inverse propagation.

dividual flow block (i.e. $c^l = \text{MLP}(\tilde{c}^l)$), as shown in Figure 2. Note that the embedding features $\mathcal{C} = \{c^l\}_{l=1}^L$ are shared in both forward and inverse propagation and only need to be computed once. We investigate the impact of embedding unit E_θ^l in Section 5.5.

4.2 Flow Module

During the forward propagation, the flow module F_θ accepts input patch $\mathcal{P} = h^0$, perform transformation through a sequence of flow blocks, and output latent variables $z = h^L$ from final block f_θ^L. Through exact log-likelihood training, F_θ learns a conditional mapping from $\mathcal{P}$ to latent distribution $p_\vartheta(z)$.

To be specific, F_θ is composed of L blocks. Each flow block consists of four flow layers, including actnorm [41], permutation layer [41], affine injector layer [47] and affine coupling layer [40]. We carefully design these layers to satisfy the invertible requirement (see supplementary material for detailed formulation). Among these layers, the affine coupling/injector layers merge features c^l from point embedding unit E_θ^l and employ distribution transformation to intermediate point representation h^l, as shown in Fig. 3.

During the inverse propagation, interpolated latent variable $\hat{z}^R = h^L$ is fed into last flow block f_θ^L as input, and upsampled estimation $\hat{\mathcal{X}}$ is generated through inverse flow pass F_θ^{-1}. We duplicate features in c^l to match the number of points between c^l and h^l during conditioning.

It is worth pointing out that the flow block f_θ^l can also be implemented using the continuous flow block (i.e. ODE-Net [45]).

4.3 Interpolation Module

Given a sparse point set $\mathcal{P}$ as input, we first enrich point-wise features s_i by a feature extractor I_E:

$$\mathcal{S} = I_E(\mathcal{P}). \tag{7}$$

where $\mathcal{S} = \left\{s_i \in \mathbb{R}^C\right\}_{i=1}^N$. See supplementary material for detailed implementation of I_E.

To expand (upsample) latent points, we need to interpolate R points for each latent point z_i. Therefore, we gather a set of neighbors as point-wise local context $\mathcal{S}_i^K = \left\{s_i^k\right\}_{k=1}^K$ by kNN algorithm, where s_i^k is the k-th nearest neighbor of s_i in $\mathcal{S}$. Then, we predict R groups of weights for each latent point z_i by a weight estimator I_W. This process can take the form:

$$\mathcal{W}_i = I_W(\mathcal{S}_i^K), \tag{8}$$

where matrix $\mathcal{W}_i = \left[w_i^{1,1}, w_i^{1,2}, \ldots, w_i^{r,k}, \ldots, w_i^{R,K}\right] \in \mathbb{R}^{R\times K}$ denotes the predicted weights for K nearest neighbors of the i-th point. The element $w_i^{r,k}$ indicates the weight of the k-th latent point z_i^k among kNN in the r-th interpolation result. I_W is simply parameterized as MLPs.

Before interpolation, we apply a softmax function to the generated weight

$$\tilde{w}_i^{r,k} = \frac{e^{w_i^{r,k}}}{\sum_{k=1}^K e^{w_i^{r,k}}}, \tag{9}$$

such that we obtain normalized weights that satisfy $\tilde{w}_i^{r,k} \geq 0$ and $\sum_{k=1}^K \tilde{w}_i^{r,k} = 1$.

Finally, interpolation can be formulated as matrix multiplication performed on latent variable z_i

$$\hat{z}_i^r = \sum_{z^k \in \mathcal{N}(p_i)} \tilde{w}_i^{r,k} z^k, \tag{10}$$

where $\hat{z}_i^r \in \mathbb{R}^D$ is the interpolated point of the r-th result, and $\mathcal{N}(p_i)$ is the set of latent variables in kNN field of point p_i. Note that the neighbor relationship is constructed in Euclidean space to keep consistent neighborship with conditional features c in E_θ.

After interpolation, we flatten interpolated dense points $\hat{z}^R$ and feed them into the inverse propagation pass of flow module F_θ, as shown in the dotted path of Fig. 2.

4.4 Training Objects

Let $\mathcal{P} \in \mathbb{R}^{N\times D}$ and $\mathcal{X} \in \mathbb{R}^{RN\times D}$ be the sets of sparse input and ground-truth dense points, respectively, with upsampling factor R. We design a joint loss function to train PU-Flow in an end-to-end manner. This objective function consists of two components: the reconstruction loss to encourage the generated points $\hat{\mathcal{X}}$ and reference $\mathcal{X}$ to share the same distribution, and the prior loss to optimize the

TABLE 1
Quantitative comparisons of state-of-the-art methods on PU1K ($N = 2048$).
We highlight the best and second best results in bold and underline, respectively.

Ratio	$R = 4$						$R = 16$					
Methods	Network Size	CD (10^{-4})	EMD (10^{-2})	HD (10^{-2})	P2F (10^{-3})	JSD (10^{-2})	Network Size	CD (10^{-4})	EMD (10^{-2})	HD (10^{-2})	P2F (10^{-3})	JSD (10^{-2})
EAR [3]	-	9.198	4.572	0.770	2.029	11.31	-	9.021	5.134	0.896	2.712	12.16
PU-Net [4]	10.1 MB	6.654	3.872	6.927	5.747	8.847	24.5 MB	6.767	4.470	1.156	4.625	11.66
MPU [6]	23.1 MB	4.401	3.100	0.340	1.237	5.204	92.5 MB	4.203	3.421	0.454	1.456	6.105
PU-GAN [7]	9.5 MB	4.239	3.036	0.540	1.580	5.382	9.5 MB	4.365	3.826	0.724	2.164	7.827
PUGeo-Net [8]	26.6 MB	3.695	2.809	0.325	1.189	4.267	26.7 MB	2.791	3.208	0.386	1.359	4.896
PU-GCN [33]	1.8 MB	3.847	2.862	0.374	1.385	4.509	1.8 MB	2.875	3.243	0.471	1.507	5.244
Dis-PU [34]	13.2 MB	3.941	2.897	0.362	1.336	4.568	13.2 MB	2.987	3.178	0.430	1.469	5.157
MAFU [9]	4.7 MB	3.624	2.776	0.450	**1.139**	4.365	4.7 MB	2.745	3.219	0.487	**1.303**	5.087
Ours (discrete)	3.4 MB	3.613	2.861	**0.310**	1.324	4.311	3.4 MB	2.675	3.217	**0.367**	1.416	4.864
Ours (continuous)	3.3 MB	**3.563**	**2.741**	0.321	1.315	**4.021**	3.3 MB	**2.548**	**3.114**	0.373	1.350	**4.728**

transformation capability of flow module F_θ by maximizing the likelihood of observation $\mathcal{P}$.

Reconstruction loss. We employ Earth Mover's distance (EMD) loss to measure the similarity between $\mathcal{X}$ and $\hat{\mathcal{X}}$:

$$\mathcal{L}_{\text{rec}} = \mathcal{L}_{\text{EMD}}(\hat{\mathcal{X}}, \mathcal{X}) = \min_{\phi:\hat{\mathcal{X}}\to\mathcal{X}} \sum_{x_i\in\hat{\mathcal{X}}} \|x_i - \phi(x_i)\|_2, \quad (11)$$

where $\phi : \hat{\mathcal{X}} \to \mathcal{X}$ is a bijective mapping.

Prior likelihood. Eq. (5) allows us to train the flow layers by minimizing the negative log-likelihood (NLL) with input patch $\mathcal{P}$:

$$\mathcal{L}_{\text{prior}}(\mathcal{P}) = \mathcal{L}(\mathcal{P}, \mathcal{C}; \theta) = -\log p(\mathcal{P} \mid \mathcal{C}, \theta), \quad (12)$$

where $\mathcal{C} = E_\theta(\mathcal{P})$. Optimizing prior likelihood of $\mathcal{P}$ encourages the encoded shape representation to gain high probability under the predefined prior $p_\vartheta(z)$, which is modeled by the flow module F_θ.

In our experiment, the prior $p_\vartheta(z)$ is simply set as standard Gaussian distribution $\mathcal{N}(0, \mathbf{I})$. In addition, $p_\vartheta(z)$ can also be set to Gaussian with learnable mean and variance, but we do not observe an obvious influence to model performance.

Total loss. Combining the preceding formulas, we train PU-Flow with respect to parameters θ by minimizing

$$\mathcal{L}(\theta) = \alpha\mathcal{L}_{\text{rec}} + \beta\mathcal{L}_{\text{prior}}, \quad (13)$$

where α and β are hyper-parameters that balance the terms.

5 Experiments

5.1 Experimental setup

Datasets. For quantitative comparison, we train and evaluate our method on following datasets:

- **PU1K.** This dataset consists of models from PU-GAN [7] and ShapeNetCore [58] of various categories, which are used in PU-GCN [33]. PU1K contains 1020 meshes for training and 127 meshes for evaluation.
- **PUGeo-Net dataset.** This dataset includes elaborate statues from Sketchfab [59], provided by PUGeo-Net [8]. It contains 90 high-resolution meshes for training and 13 for testing, with complex geometry and high-frequency details.
- **PU36.** To achieve more generalized evaluation results of more categories, we constructed a new dataset for evaluation, containing 36 models collected from Sketchfab [59]. Please refer to supplementary material for gallery of all shapes.
- **PU-GAN dataset.** This dataset [7] includes 120 models for training and 27 for testing. The testing set contains a variety of basic shapes.
- **FAMOUSTHINGI.** This dataset includes models chosen from Thingi10k [60] and PCPNet [61] datasets, with a total of 37 shapes for evaluation. We use FAMOUSTHINGI [62] dataset to evaluate the quality of surface reconstruction results.

In the experiments, the results on PU1K, PUGeo-Net, PU36 and FAMOUSTHINGI datasets are trained and evaluated on uniform data, while the results on PU-GAN dataset are trained and evaluated on non-uniform data. The results on the PU1K, PU36 and PU-GAN datasets are evaluated by the same evaluation script as PU-GCN [33]. The results on the PUGeo-Net dataset is evaluated by the same evaluation script as PUGeo-Net [8].

Methods under comparison. We compare our model with a representative optimization-based method and six state-of-the-art deep learning-based methods, including EAR [3], PU-Net [4], MPU [6], PU-GAN [7], PUGeo-Net [8], PU-GCN [33], Dis-PU [34] and MAFU [9]. For fair comparison, we use the public released code and retrain the models for all deep learning-based methods on each datasets for evaluation. Note that as PU1K and PU-GAN datasets do not contain the normal information of points, we retrained PUGeo-Net and MAFU without using their normal generation modules.

Implementation details. We implemented PU-Flow with PyTorch framework. The corresponding source code will be published later, including both discrete and continuous implementations. The training settings, detailed network architectures, hyper-parameters and evaluation practices are provided in supplementary material.

5.2 Comparisons on upsampled points

Evaluation metrics. We employ five evaluation metrics, including (i) Chamfer Distance (CD), (ii) Earth Mover's distance (EMD), (iii) Point-to-surface distance (P2F), (iv) Hausdorff distance (HD) and (v) Jensen-Shannon divergence (JSD). All metrics are estimated on the whole point set after merging from upsampled patches. The lower the values are, the better the upsampling quality is.

Fig. 4. Visual Comparisons of various methods (b-f). We visualized the P2F errors by colors for each point. The images are best viewed on screen when zoomed in. See also the supplementary material for more comparisons and visual results.

Quantitative comparison on uniform inputs. Table 1 shows the comparison results evaluated on PU1K dataset. We also compare the network size as an intuitional metric to evaluate the memory efficiency of networks. We can see that both our discrete and continuous models can achieve the lowest values on most evaluation metrics.

To be specific, deep learning-based methods significantly outperform EAR on all metrics. This reveals the superiority of the deep learning technique compared with the optimization-based methods. For PU-GAN, it fails to maintain a good estimation of the HD and P2F metrics. For PU-GCN, we use the pretrained model provided by [33] and achieve consistent results in [33]. The results in Table 1 show the performance of PU-GCN is inferior to other methods. We also observe that PUGeo-Net and Dis-PU have competitive performance on all metrics, but both of them require relatively more network parameters. Since PUGeo-Net can not be trained without normal supervision on PU1K training set, we use the pretrained model of PUGeo-Net instead.

In contrast to the aforementioned works, our method disentangles the point expanding and decoding processes, which extensively reduces the number of parameters but still preserves reasonable results. Our method achieves the best balance between network size and generation quality. Table 2 shows the results of different methods evaluated on PUGeo-Net dataset and PU36 dataset. Still, our method maintains advantages on most metrics.

Quantitative comparison on non-uniform inputs. Table 3 shows the results evaluated on the PU-GAN dataset. From

TABLE 2
Quantitative comparisons of state-of-the-art methods on PUGeo-Net dataset and PU36 ($N = 5000, R = 4$). We highlight the best and second best results in bold and underline, respectively.

Dataset	PUGeo-Net dataset					PU36				
Methods	CD (10^{-2})	EMD (10^{-2})	HD (10^{-2})	P2F (10^{-3})	JSD (10^{-2})	CD (10^{-4})	EMD (10^{-2})	HD (10^{-3})	P2F (10^{-3})	JSD (10^{-2})
EAR [3]	0.660	2.745	1.990	1.489	1.192	1.695	2.323	1.571	1.843	2.319
PU-Net [4]	0.658	2.419	1.003	1.532	0.950	3.245	2.706	4.505	9.704	3.960
MPU [6]	0.573	1.491	1.073	0.808	0.614	1.049	1.632	1.040	1.507	0.895
PU-GAN [7]	0.586	1.957	1.422	0.889	0.579	1.516	2.146	3.624	2.292	1.926
PUGeo-Net [8]	0.558	1.479	**0.934**	0.617	0.444	1.016	1.557	**0.909**	1.514	0.832
PU-GCN [33]	0.568	1.538	1.093	0.754	0.542	1.082	1.641	1.729	1.682	1.014
Dis-PU [34]	0.585	1.719	1.318	0.819	0.615	1.235	1.823	2.568	2.083	1.265
MAFU [9]	0.556	1.480	1.077	0.653	0.453	1.060	1.558	1.024	1.576	0.974
Ours (discrete)	**0.551**	**1.441**	0.951	**0.582**	**0.422**	0.979	**1.520**	1.011	**1.434**	**0.749**
Ours (continuous)	0.553	1.464	0.963	0.590	0.433	**0.973**	1.532	0.991	1.463	0.781

TABLE 3
Quantitative comparisons of state-of-the-art methods with non-uniform inputs on the PU-GAN dataset ($N = 2048$, $R = 4$). We highlight the best and second best results in bold and underline, respectively.

Methods	CD (10^{-4})	EMD (10^{-2})	HD (10^{-3})	P2F (10^{-3})	JSD (10^{-2})
EAR [3]	5.543	4.562	9.042	7.038	6.872
PU-Net [4]	4.642	4.035	8.249	8.348	9.267
MPU [6]	3.410	3.439	4.758	2.935	4.898
PU-GAN [7]	**2.985**	**2.787**	5.284	2.818	**3.811**
PUGeo-Net [8]	3.286	3.298	5.693	2.589	4.500
PU-GCN [33]	3.476	3.433	4.821	2.809	5.459
Dis-PU [34]	3.288	3.084	5.425	**2.460**	4.479
MAFU [9]	3.192	3.247	**4.472**	2.671	4.268
Ours (discrete)	3.103	3.143	4.631	2.724	4.149

TABLE 4
Quantitative comparisons of mesh reconstruction results on the FAMOUSTHINGI dataset [62]. We highlight the best and second best results in bold and underline, respectively.

Methods	CD (10^{-2})	NW (%)	NR (degree)
EAR [3]	0.684	4.203	16.67
PU-Net [4]	1.071	11.285	29.06
MPU [6]	0.409	0.886	10.29
PU-GAN [7]	0.453	3.533	12.96
PUGeo-Net [8]	**0.393**	0.849	9.75
PU-GCN [33]	0.421	2.464	12.15
Dis-PU [34]	0.453	2.796	11.98
MAFU [9]	0.407	0.854	9.87
Ours (discrete)	0.394	**0.744**	**9.69**
Ours (continuous)	0.398	0.806	10.01
Reference	0.326	0.397	5.22

Table 3, we observe that PU-GAN [7] achieves the best results on CD, EMD and JSD metrics, but fails to preserve the superiority in terms of HD and P2F metrics. Among other non-GAN-based methods, our method achieves competitive performance to PUGeo-Net [8] and MAFU [9] and outperforms PU-GCN [33] and Dis-PU [34].

Qualitative comparison. We visualize the 4x upsampling results of different methods with 5K inputs points in Fig. 4. We use color to reveal the P2F error for each point.

Compared with other methods, our method achieves minimum average errors. It can better preserve the smoothness of local regions and produce a reliable shape, while other methods tend to produce more noisy points between some complex adjacent regions, as shown in Fig. 4 (b), (c), and (d).

5.3 Comparisons on reconstructed surface

To further demonstrate the generation quality of our method, we compare the surface reconstruction results from upsampled points ($N = 2500$, $R = 4$) with state-of-the-art methods.

Specifically, we employ DSE-meshing [62], a cutting edge method for mesh reconstruction from point clouds, as mesh generator. We use DSE-meshing instead of traditional methods, such as screened Poisson Sampling Reconstruction [64] and ball-pivoting surface reconstruction [65], for the following reasons: (i) Traditional methods generally require additional information (e.g. normal data) and careful parameter selection to obtain satisfactory results. Since DSE-meshing is an end-to-end method, we can achieve more fair comparisons by eliminating the influence of normal accuracy and manual parameters tuning. (ii) The quality of upsampled points have significant impact to DSE-meshing. Thus, we can employ the quantitative comparison between reconstructed mesh to evaluate the upsampling quality of various methods.

Evaluation metrics. We consider three metrics to evaluate mesh quality: (i) Chamfer Distance (CD), (ii) the percentage of non-watertight edges (NW), (iii) normal reconstruction error in degrees (NR). CD measures the distance between point sets sampled on reconstructed and ground truth surface. NW counts the number of triangle edges that are only shared by one triangle. NR measures the angle difference of normals between reconstructed and ground truth surface. For these metrics, the lower the values are, the better the mesh quality is.

Quantitative comparison. Table 4 summarizes the comparison results evaluated on FAMOUSTHINGI dataset. We also include the mesh reconstructed from ground truth points (denoted as Reference). From Table 4, we can observe the similar trend with Table 2. Obvious performance gap still exists between the best one and reference. In particular, both our discrete and continuous model yield lowest reconstruction error (CD and NR) and produce least non-manifold edges (NW).

Qualitative comparison. Fig. 5 visualizes the reconstructed mesh between representative works. We can observe notable artifacts in the region with large curvature, especially for

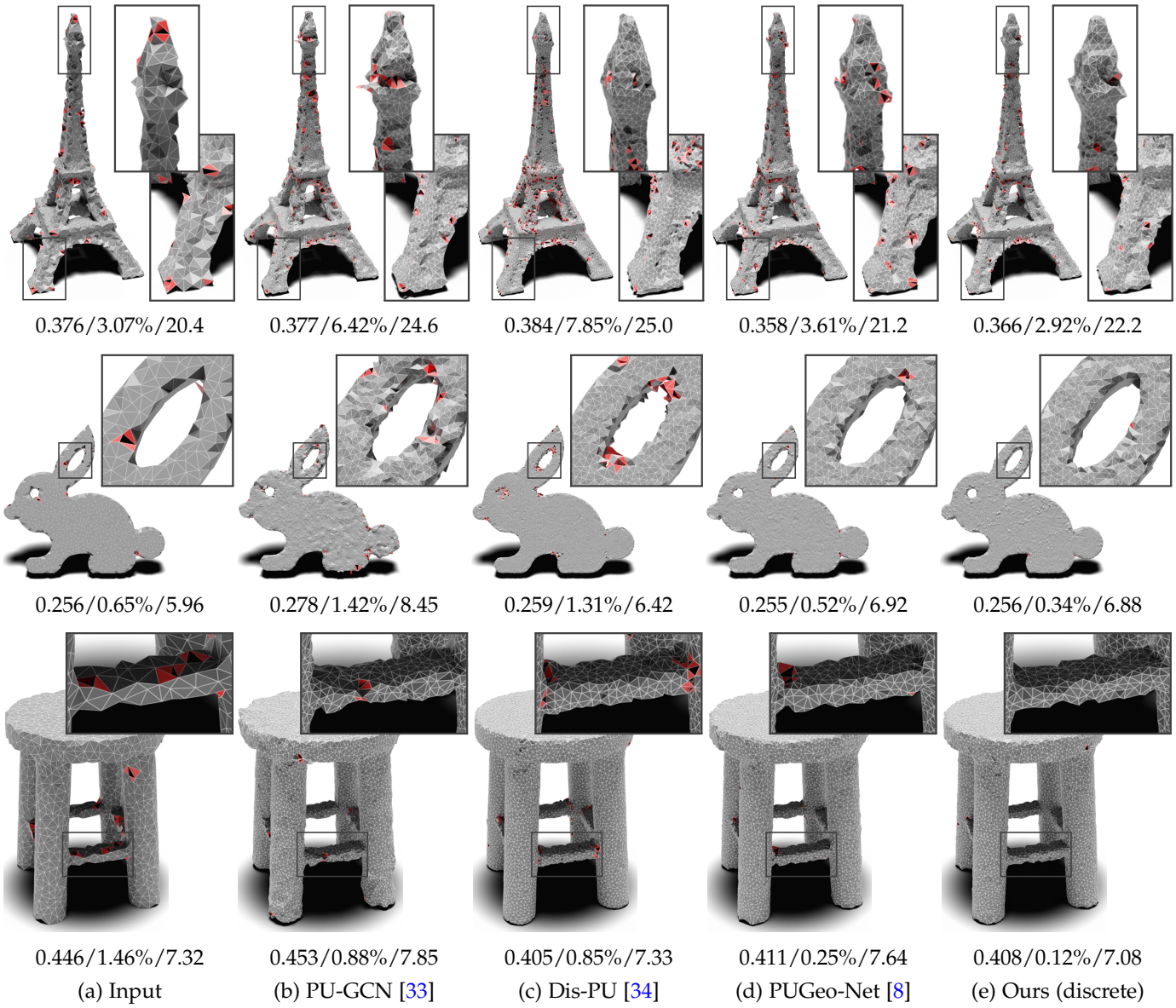


Fig. 5. Visual Comparisons of reconstructed surfaces from upsampled points of various methods (b-e). The first column shows the mesh reconstructed from sparse input points. To visualize the artifacts of mesh surface, we mark the non-manifold triangles in red. We display the Chamfer distance (multiplied by 100), the percentage of non-watertight edges (NW) and normal reconstruction error (NR) below each object. See supplementary material for more visual results.

geometrically complex surface (e.g. tower in Fig. 5). The uniformity and outliers of upsampled points affect the quality of mesh significantly. PUGeo-Net and our method both achieve the most promising reconstruction quality. However, our method produce less non-manifold (bad) triangles across most objects, demonstrating its better point distribution.

5.4 Upsampling Real-world Data

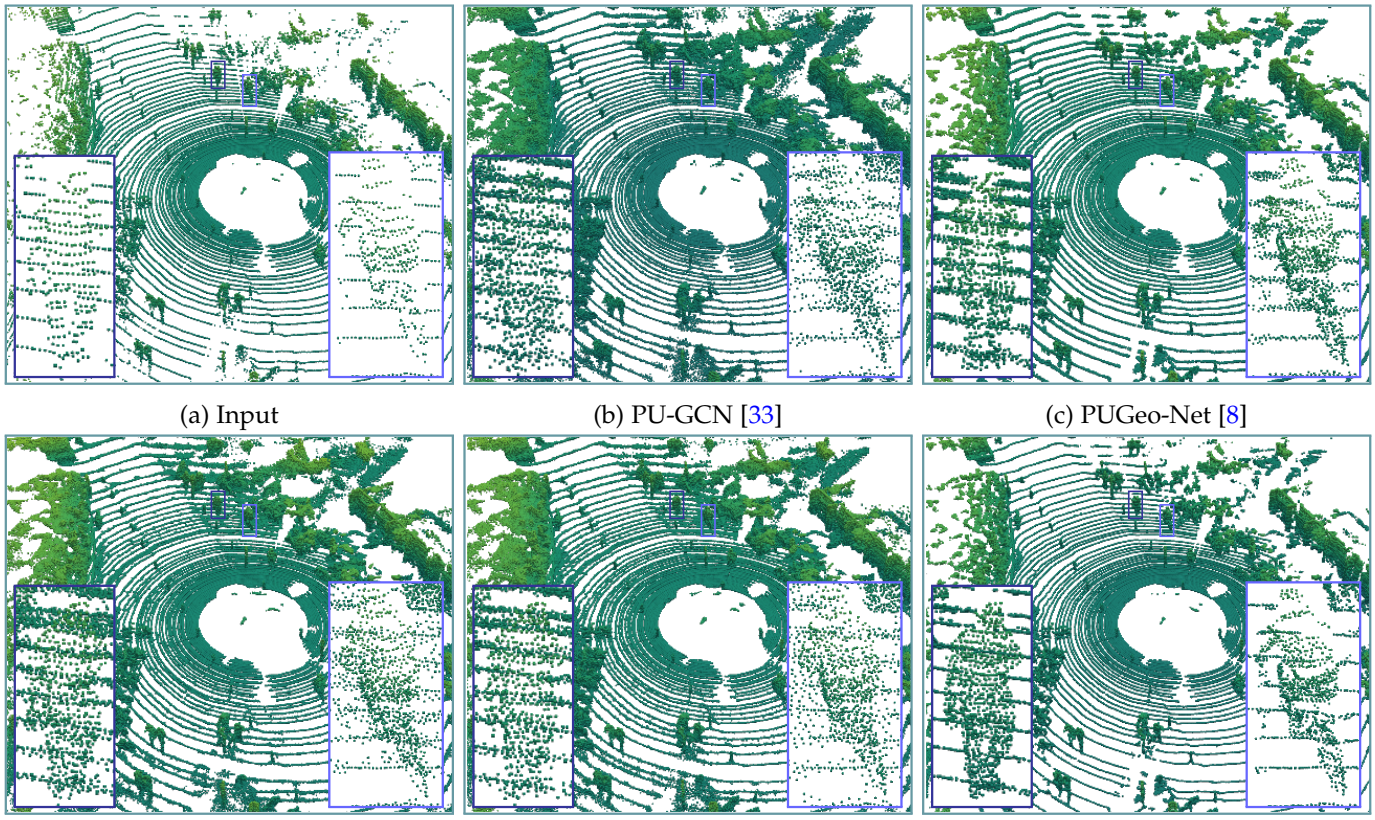
To confirm the robustness on more complicated and unseen data, we evaluate our method on two real-world point clouds datasets: KITTI [63] and ScanObjectNN [66].

KITTI. This dataset captures the point clouds of driving scenes with its data produced by LiDAR sensors. The raw input data severely suffer from sparsity, occlusion, and non-uniformity. For example, some vehicles, people, and plants are only described with few points and the density of points vary across distance to center. As shown in Fig. 6, we observe other methods suffer from sparsity and non-uniformity issues and thus produce more outliers. Our method can generate dense points with more fine-grained details, resulting in better object visibility when compared with other methods.

ScanObjectNN. This dataset comprises point clouds of scanned indoor scenes, where objects are divided into 15 categories. The raw input objects are cluttered with background and suffer from the partial occlusion and the scan line distribution pattern. As shown in Fig. 8, our method improves the visibility quality and makes objects more distinguishable from the background.

5.5 Ablation Study

In this section, we quantitatively evaluate the contribution of network design of PU-Flow. We use the discrete model in-



(a) Input (b) PU-GCN [33] (c) PUGeo-Net [8]

(d) PU-GAN [7] (e) Dis-PU [34] (f) Ours (discrete)

Fig. 6. Visual comparisons of upsampling results ($R = 4$) on the KITTI [63] dataset. See supplementary material for more visual results.

TABLE 5
Ablation study of flow architectures.

Ablation		CD (10^{-4})	P2F (10^{-3})
Vanilla pipeline		1.17	1.89
Embedding Unit E_θ	None	1.31	2.36
	PointNet [14]	1.14	1.84
	EdgeConv [16]	1.03	1.57
	PAConv [67]	1.09	1.64
Point Decoder	MLP (w/ $\mathcal{L}_{\text{prior}}$)	1.35	2.16
	MLP (w/o $\mathcal{L}_{\text{prior}}$)	1.12	1.91
	xyz-interpolation	1.38	1.84
Graph used in $E_\theta + I_\theta$	static + dynamic	2.05	2.86
	dynamic + static	1.02	1.45
	dynamic + dynamic	1.95	2.52
Full pipeline (Point decoder: F_θ^{-1}, Graph: static + static)		**0.98**	**1.43**

stead of continuous model for evaluation, because they have very close performance. The benchmarks are evaluated on PU36 dataset with input points $N = 5000$ and upsampling factor $R = 4$.

Flow architecture. We first construct a vanilla model (denoted as vanilla pipeline in Table 5), which implements the basic idea of weighted interpolation for upsampling. As shown in Figure 7, this model generates weights and high-level point abstraction from shared point-wise semantic features. Compared to our full pipeline, the vanilla model has relatively low performance, demonstrating that it has difficulty to generate appropriate weights for latent features.

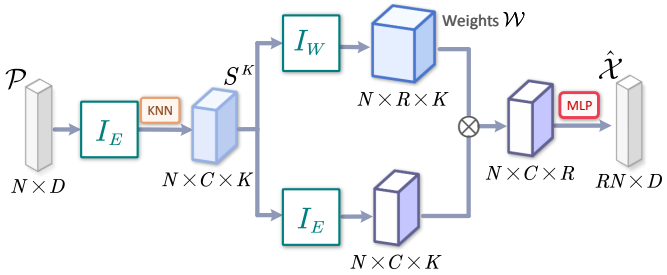


Fig. 7. Network architecture of the vanilla model.

Figure 9 shows the upsampled results by the vanilla model and our method, where the vanilla model fails to preserve a smooth distribution on surface and produces more jitters.

We investigate the impact of point embedding unit E_θ, as shown in Table 5. The vanilla model does not use features from E_θ (the None row). In this way, the flow module F_θ uses independent transformation for each point and thus our method suffers from underfitting. By integrating features from modern feature extractor, F_θ reveals promising transform capability, thus leading to substantial performance boost.

To validate the effectiveness of generating points by inverse propagation F_θ^{-1}, we replace it with MLPs used in previous works [4], [6], [7], in which $\mathcal{L}_{\text{prior}}$ is not needed. The results in Table 5 show that using F_θ^{-1} for coordinate reconstruction can better preserve intricate structures than MLPs, which demonstrates the feasibility of F_θ^{-1}.

Fig. 8. Upsampling results by PU-Flow in different categories (chair, computer monitor). For better visualization, the color of upsampled points (2nd column) are assigned by the color of the closest input point (1st column). See the supplementary material for more visual results.

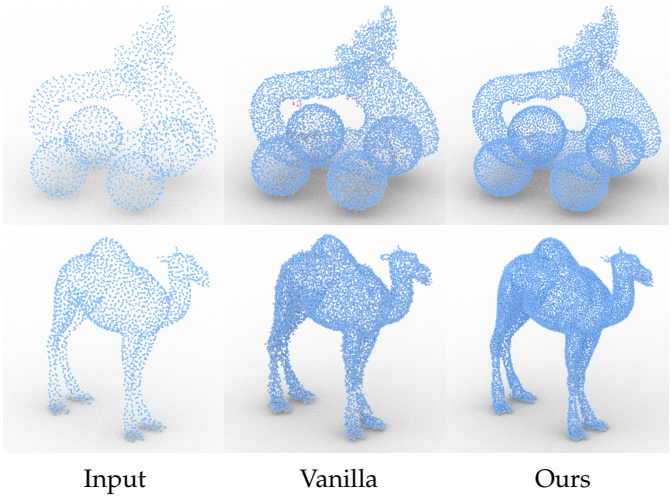


Fig. 9. Visual comparisons of generated points between the vanilla model and our method ($N = 2048$, $R = 4$).

Fig. 10 shows the effect of the number of flow blocks L used in PU-Flow. When L is relatively low, our method achieves better performance as L increase. When $L \geq 8$, the performance gain becomes unobvious, with the cost of computational overhead and increasing network parameters. The best performance of the full pipeline is achieved when setting $L = 6$ to $L = 8$.

Interpolation module. Fig. 11 shows the impact of the

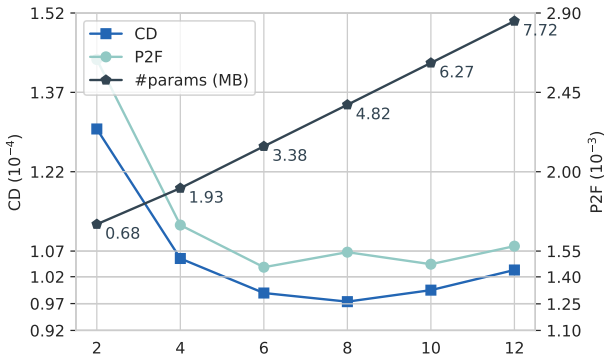


Fig. 10. Ablation study of number of flow block from $L = 2$ to $L = 12$.

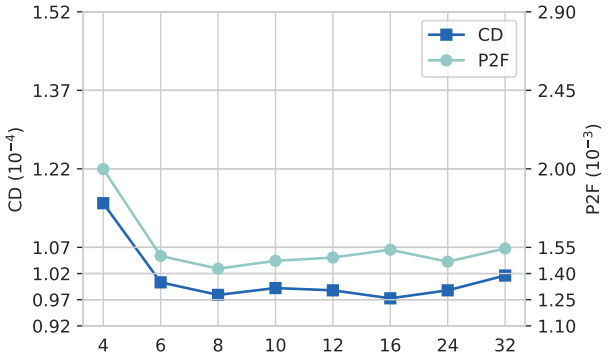


Fig. 11. Ablation study of number of interpolation neighbors for $K \in [4, 32]$ to $K = 32$.

number of neighbours K participating in interpolation. A larger value of K means a broader area for generation. We observe that both a small (e.g. $K \leq 4$) and large (e.g. $K \geq 32$) value of K can lead to degraded performance. Otherwise, our method is not sensitive to K assignment. In this study, we set $K = 16$ by default.

Furthermore, we try to use the weights from I_θ to directly interpolate points in xyz coordinates (the xyz-interpolation row in Table 5). It turns out that these weights are not feasible in Euclidean space, demonstrating that I_θ adaptively learns weights specific to latent point under prior distribution.

We also investigate the impact of space for point embedding and interpolation (i.e. the graph type used in E_θ and I_θ), as shown in Table 5. Employing interpolation in latent space means that the k-nearest-neighbour graph is dynamically constructed (denoted as dynamic graph) in KNN operator of Fig. 2. We can see a significant performance drop when using dynamic graph for interpolation. We hypothesize the potential reason is as follows: F_θ and I_θ are independent branch of upsampling pipeline. Using dynamic graph in I_θ does not ensure consistent neighbour relationship between upsampled latents $\hat{z}^R$ and conditional features c, resulting into inconsistent features conditioning during inverse propagation. In contrast, employing interpolation in Euclidean space (denoted as static graph) would not cause this issue, and thus achieve competitive performance. Besides, the graph type used in E_θ has relative little impact on performance.

Flow components. Fig. 12 shows the ablation study evaluated on each flow component. We observe that our method cannot yield reasonable results without affine cou-

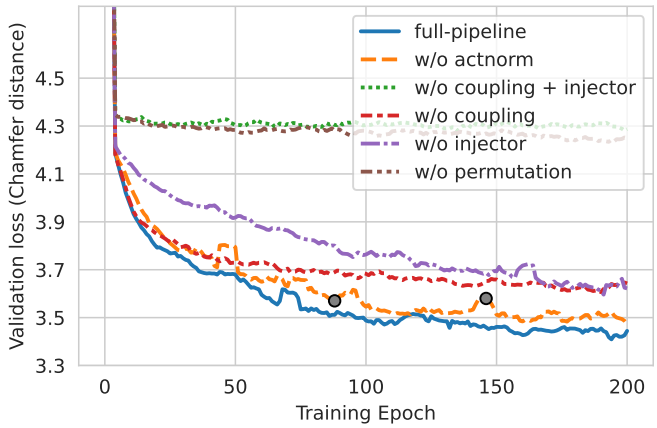


Fig. 12. Ablation study of flow components. We show the loss curves of training PU-Flow under different settings of flow components. The grey point indicate the infinite value encountered during training.

pling/injector layers. This demonstrates that the flow module F_θ requires features extracted from E_θ to encode point to latent representation. The affine coupling and injector layers can be used as drop-in replacement to each other. The permutation layers are necessary components to ensure that all channels are fully processed by coupling layer. When the actnorm layer is removed, the training process becomes more sensitive to gradient explode and we may encounter invalid loss value (e.g. infinite). Thus, the actnorm layer makes significant contribution to training stability.

6 Conclusion

We presented a novel point cloud upsampling framework, called PU-Flow, that leverages NFs and weight prediction technique. With the invertible capability of NFs, we can transform the point clouds between Euclidean space and latent distribution in a lossless manner. We formulate upsampling as local ensemble of latent variables, with the interpolation weights adaptively learned from the local neighborhood context. Given sparse point cloud as input, our method can produce a dense output that provides a reasonable prediction of the underlying surface and also contains fine details. Quantitative evaluation demonstrates that PU-Flow outperforms existing state-of-the-art works in terms of quality and efficiency.

While we have demonstrated high quality results of PU-Flow, our approach is still subject to a few limitations that can be addressed in follow-up work. First, the dimension bottleneck problem, described in Section 4.1, limits the expressive ability of latent points in flow transformation. Second, due to the invertible requirement, some commonly adopted network designs, such as skip connection, code assignment, up-down-up expansion pattern, etc. are not feasible for flow architecture. Third, our method upsamples points at the *patch* level, and thus has a limited ability in inferring global shape and large holes. Lastly, from Table 3, it can be seen that the capability of our method in handling non-uniform inputs is still a little limited, compared with state-of-the-art methods. Considering the impressive performance of PU-GAN in such a scenario, we can incorporate our method into a GAN-based architecture to further improve its performance on non-uniform data.

In the future, we will extend PU-Flow to simultaneously generate normals and a higher resolution of geometry for sparse input. Furthermore, we will investigate the propagation pipeline of PU-Flow to point cloud compression tasks, which proposes a high requirement of detail reconstruction and efficient storage, and denoise task, which is sensitive to noisy point distribution.

Acknowledgment

We thank the anonymous reviewers for their constructive comments that help us to improve the paper. This work was supported in part by the NSF of Guangdong Province (2019A1515010833, 2022A1515011573), in part by the Natural Science Foundation of China (61725204, 61871342), and in part by the Hong Kong Research Grants Council under Grants 11202320 and 11218121, and in part by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (RG20/20) and Tier 2 (MOE-T2EP20220-0005).

References

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, "Computing and rendering point set surfaces," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 1, pp. 3–15, 2003.

[2] Y. Lipman, D. Cohen-Or, and D. Levin, "Parameterization-free projection for geometry reconstruction," *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, pp. 22–28, 2007.

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

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

[5] L. Yu, X. Li, D. Cohen-Or, C.-W. Fu, and P.-A. Heng, "Ec-net: an edge-aware point set consolidation network," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 386–402.

[6] Y. Wang, S. Wu, H. Huang, D. Cohen-Or, and O. Sorkine-Hornung, "Patch-based progressive 3d point set upsampling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 5951–5960.

[7] R. Li, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "Pu-gan: a point cloud upsampling adversarial network," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 7203–7212.

[8] Y. Qian, J. Hou, S. Kwong, and Y. He, "Pugeo-net: A geometry-centric network for 3d point cloud upsampling," in *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2020, pp. 752–769.

[9] Y. Qian, J. Hou, and S. Kwong, "Deep magnification-flexible upsampling over 3d point clouds," *IEEE Transactions on Image Processing*, vol. 30, pp. 8354–8367, 2021.

[10] X. Hu, H. Mu, X. Zhang, Z. Wang, T. Tan, and J. Sun, "Meta-sr: A magnification-arbitrary network for super-resolution," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 1575–1584.

[11] Y. Chen, S. Liu, and X. Wang, "Learning continuous image representation with local implicit image function," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 8628–8638.

[12] H. Huang, D. Li, H. Zhang, U. Ascher, and D. Cohen-Or, "Consolidation of unorganized point clouds for surface reconstruction," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 5, pp. 1–7, 2009.

[13] S. Wu, , H. Huang, M. Gong, M. Zwicker, and D. Cohen-Or, "Deep points consolidation," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, pp. 1–13, 2015.

[14] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 652–660.

[15] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5105–5114.

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

[17] J. Li, B. M. Chen, and G. H. Lee, "So-net: Self-organizing network for point cloud analysis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 9397–9406.

[18] Y. Liu, B. Fan, S. Xiang, and C. Pan, "Relation-shape convolutional neural network for point cloud analysis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8895–8904.

[19] G. Te, W. Hu, A. Zheng, and Z. Guo, "Rgcnn: Regularized graph cnn for point cloud segmentation," in *Proceedings of the 26th ACM international conference on Multimedia*, 2018, pp. 746–754.

[20] L. Wang, Y. Huang, Y. Hou, S. Zhang, and J. Shan, "Graph attention convolution for point cloud semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10 296–10 305.

[21] S.-M. Hu, J.-X. Cai, and Y.-K. Lai, "Semantic labeling and instance segmentation of 3d point clouds using patch context analysis and multiscale processing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 7, pp. 2485–2498, 2020.

[22] G. Elbaz, T. Avraham, and A. Fischer, "3d point cloud registration for localization using a deep neural network auto-encoder," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4631–4640.

[23] Y. Aoki, H. Goforth, R. A. Srivatsan, and S. Lucey, "Pointnetlk: Robust & efficient point cloud registration using pointnet," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 7163–7172.

[24] Z. Yan, Z. Yi, R. Hu, N. J. Mitra, D. Cohen-Or, and H. Huang, "Consistent two-flow network for tele-registration of point clouds," *IEEE Transactions on Visualization and Computer Graphics*, doi:10.1109/TVCG.2021.3086113, 2021.

[25] D. Zhang, X. Lu, H. Qin, and Y. He, "Pointfilter: Point cloud filtering via encoder-decoder modeling," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 3, pp. 2015–2027, 2020.

[26] Z. Xu and A. Foi, "Anisotropic denoising of 3d point clouds by aggregation of multiple surface-adaptive estimates," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 6, pp. 2851–2868, 2021.

[27] Y. Li and G. Baciu, "Sg-gan: Adversarial self-attention gcn for point cloud topological parts generation," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2021.

[28] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert, "Pcn: Point completion network," in *2018 International Conference on 3D Vision (3DV)*, 2018, pp. 728–737.

[29] L. P. Tchapmi, V. Kosaraju, H. Rezatofighi, I. Reid, and S. Savarese, "Topnet: Structural point cloud decoder," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 383–392.

[30] Z. Huang, Y. Yu, J. Xu, F. Ni, and X. Le, "Pf-net: Point fractal network for 3d point cloud completion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 7662–7670.

[31] Z. Chen, W. Zeng, Z. Yang, L. Yu, C.-W. Fu, and H. Qu, "Lassonet: Deep lasso-selection of 3d point clouds," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 195–204, 2019.

[32] A. Bletterer, F. Payan, and M. Antonini, "A local graph-based structure for processing gigantic aggregated 3d point clouds," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 8, pp. 2822–2833, 2020.

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

[34] R. Li, X. Li, P.-A. Heng, and C.-W. Fu, "Point cloud upsampling via disentangled refinement," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 344–353.

[35] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International Conference on Machine Learning*. PMLR, 2015, pp. 1530–1538.

[36] R. van den Berg, L. Hasenclever, J. Tomczak, and M. Welling, "Sylvester normalizing flows for variational inference," in *proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018, pp. 393–402.

[37] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, "Improved variational inference with inverse autoregressive flow," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 29, 2016, pp. 4743–4751.

[38] J. Oliva, A. Dubey, M. Zaheer, B. Poczos, R. Salakhutdinov, E. Xing, and J. Schneider, "Transformation autoregressive networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3898–3907.

[39] L. Dinh, D. Krueger, and Y. Bengio, "Nice: Non-linear independent components estimation," *arXiv preprint arXiv:1410.8516*, 2014.

[40] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," in *International Conference on Learning Representations (ICLR)*, 2017, pp. 1–12.

[41] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018, pp. 10 215–10 224.

[42] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, "Ffjord: Free-form continuous dynamics for scalable reversible generative models," in *International Conference on Learning Representations (ICLR)*, 2019, pp. 1–10.

[43] G. Yang, X. Huang, Z. Hao, M.-Y. Liu, S. Belongie, and B. Hariharan, "Pointflow: 3d point cloud generation with continuous normalizing flows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 4541–4550.

[44] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, "Neural spline flows," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019, pp. 7511–7522.

[45] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 6571–6583.

[46] L. Ardizzone, C. Lüth, J. Kruse, C. Rother, and U. Köthe, "Guided image generation with conditional invertible neural networks," *arXiv preprint arXiv:1907.02392*, 2019.

[47] A. Lugmayr, M. Danelljan, L. Van Gool, and R. Timofte, "Srflow: Learning the super-resolution space with normalizing flow," in *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2020, pp. 715–732.

[48] A. Pumarola, S. Popov, F. Moreno-Noguer, and V. Ferrari, "C-flow: Conditional generative flow models for images and 3d point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 7949–7958.

[49] A. Grover, C. Chute, R. Shu, Z. Cao, and S. Ermon, "Alignflow: Learning from multiple domains via normalizing flows," in *DGS@ ICLR*, 2019, pp. 1–10.

[50] M. Kumar, M. Babaeizadeh, D. Erhan, C. Finn, S. Levine, L. Dinh, and D. Kingma, "Videoflow: A flow-based generative model for video," in *International Conference on Learning Representations (ICLR)*, 2020, pp. 1–13.

[51] J. Liu, A. Kumar, J. Ba, J. Kiros, and K. Swersky, "Graph normalizing flows," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 13 578–13 588.

[52] R. Prenger, R. Valle, and B. Catanzaro, "Waveglow: A flow-based generative network for speech synthesis," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 3617–3621.

[53] S. Kim, S.-G. Lee, J. Song, J. Kim, and S. Yoon, "Flowavenet: A generative flow for raw audio," in *International Conference on Machine Learning*, 2019, pp. 3370–3378.

[54] D. Rempe, T. Birdal, Y. Zhao, Z. Gojcic, S. Sridhar, and L. J. Guibas, "Caspr: Learning canonical spatiotemporal point cloud representations," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 13 688–13 701.

[55] R. Abdal, P. Zhu, N. J. Mitra, and P. Wonka, "Styleflow: Attribute-conditioned exploration of stylegan-generated images using conditional continuous normalizing flows," *ACM Transactions on Graphics (TOG)*, vol. 40, no. 3, pp. 1–21, 2021.

[56] R. Klokov, E. Boyer, and J. Verbeek, "Discrete point flow networks for efficient point cloud generation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020, pp. 694–710.

[57] J. Postels, M. Liu, R. Spezialetti, L. Van Gool, and F. Tombari, "Go with the flows: Mixtures of normalizing flows for point cloud generation and reconstruction," in *2021 International Conference on 3D Vision (3DV)*, 2021, pp. 1249–1258.
[58] 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," Stanford University — Princeton University — Toyota Technological Institute at Chicago, Tech. Rep. arXiv:1512.03012 [cs.GR], 2015.
[59] Sketchfab. https://sketchfab.com/.
[60] Q. Zhou and A. Jacobson, "Thingi10k: A dataset of 10,000 3d-printing models," *arXiv preprint arXiv:1605.04797*, 2016.
[61] P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra, "PCPNet: Learning local shape properties from raw point clouds," *Computer Graphics Forum*, vol. 37, no. 2, pp. 75–85, 2018.
[62] M.-J. Rakotosaona, P. Guerrero, N. Aigerman, N. J. Mitra, and M. Ovsjanikov, "Learning delaunay surface elements for mesh reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 22–31.
[63] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, vol. 32, no. 11, pp. 1231–1237, 2013.
[64] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 3, pp. 1–13, 2013.
[65] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349–359, 1999.
[66] M. A. Uy, Q.-H. Pham, B.-S. Hua, D. T. Nguyen, and S.-K. Yeung, "Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1588–1597.
[67] M. Xu, R. Ding, H. Zhao, and X. Qi, "Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 3173–3182.
[68] C. Winkler, D. Worrall, E. Hoogeboom, and M. Welling, "Learning likelihoods with conditional normalizing flows," *arXiv preprint arXiv:1912.00042*, 2019.
[69] M. Corsini, P. Cignoni, and R. Scopigno, "Efficient and flexible sampling with blue noise properties of triangular meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 6, pp. 914–924, 2012.
[70] Cgal. https://www.cgal.org/.

Aihua Mao is a professor with the School of Computer Science and Engineering, South China University of Technology (SCUT), China. He received the PhD degree from the Hong Kong Polytechnic University in 2009, the M.Sc degree from Sun Yat-Sen University in 2005 and the B.Eng degree from Hunan University in 2002. His research interests include 3D vision and computer graphics.

Zihui Du received the B.S. degree in computer science from TianGong University, Tianjin, China in 2018. He is currently pursuing the M.S. degree in computer science with South China University of Technology, Guangzhou, China. His current research interests include machine learning, 3D point cloud generation and normalizing flows.

Junhui Hou Junhui Hou received the B.Eng. degree in information engineering (Talented Students Program) from the South China University of Technology, Guangzhou, China, in 2009, the M.Eng. degree in signal and information processing from Northwestern Polytechnical University, Xian, China, in 2012, and the Ph.D. degree in electrical and electronic engineering from the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, in 2016. Since Jan. 2017, he has been an Assistant Professor with the Department of Computer Science, City University of Hong Kong. His research interests fall into the general areas of visual computing, such as image/video/3D geometry data representation, processing and analysis, semi/un-supervised data modeling, and data compression and adaptive transmission.

Yaqi Duan received the B.S. degree in computer science from TianGong University, Tianjin, China in 2020. He is currently pursuing the M.S. degree in computer science with South China University of Technology, Guangzhou, China. His research focuses on deep learning, invertible networks and point cloud processing.

Yong-Jin Liu is a professor with the Department of Computer Science and Technology, Tsinghua University, China. He received the BEng degree from Tianjin University, China, in 1998, and the PhD degree from the Hong Kong University of Science and Technology, Hong Kong, China, in 2004. His research interests include computer graphics and computer-aided design.

Ying He is an associate professor in the School of Computer Engineering, Nanyang Technological University, Singapore. He received his Bachelor (1997) and Master (2000) degrees in Electrical Engineering from Tsinghua University, and PhD (2006) in Computer Science from Stony Brook University. He is interested in the problems that require geometric computing and analysis.