3D Point Cloud Analysis via Deep Learning

LI, Ruihui

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
July 2021

Thesis Assessment Committee

Professor XU Qiang (Chair)

Professor FU Chi Wing (Thesis Supervisor)

Professor HENG Pheng Ann (Committee Member)

Professor Zheng Jianmin (External Examiner)

Abstract

3D point clouds, as the default outputs from 3D scanning, are used in a wide range of applications. Though recent years have witnessed great progress on the 3D sensing technology, they are gaining more and more popularity as a compact representation for 3D data, and as an effective means for processing 3D geometry. Inspired by the wide impact of data-driven methods on various image-related tasks, a number of deep-learning-based networks have been proposed for point cloud processing and demonstrated superior performance. In this thesis, we present our efforts on 3D point cloud analysis via deep learning approaches from various aspects, especially considering the low quality of acquired 3D data and limited quantity of 3D datasets.

First, since raw point clouds produced from depth cameras and LiDAR sensors are often sparse, noisy, and non-uniform, we present a new point cloud upsampling framework, namely PU-GAN, that combines upsampling with data amendment capability. We formulate the network by leveraging the generative adversarial network to learn to synthesize points with a uniform distribution in the latent space and upsample points over

patches on object surfaces.

Second, after revisiting the point cloud upsampling task, we propose to disentangle the task based on its multi-objective nature and formulate two cascaded subnetworks, a dense generator, and a spatial refiner. The dense generator infers a coarse but dense output that roughly describes the underlying surface, while the spatial refiner further fine-tunes the coarse output by adjusting the location of each point. Compared with current upsampling methods, our disentangled refinement pipeline assigns lower requirements to each sub-network, so that both the dense generator and the spatial refiner could be more focused on their own sub-goals.

Third, considering that 3D datasets are typically smaller in quantity and diversity, we present PointAugment, a new auto-augmentation framework that automatically optimizes and augments point cloud samples to enrich the data diversity when we train a classification network. Different from existing auto-augmentation methods for 2D images, PointAugment is sample-aware and takes an adversarial learning strategy to jointly optimize an augmentor network and a classifier network, such that the augmentor can learn to produce augmented samples that best fit the classifier.

Lastly, we present SP-GAN, a new unsupervised generative model for direct synthesis of 3D point clouds. SP-GAN is able to synthesize diverse and high-quality shapes with fine details and promote controllability for part-aware shape generation and

manipulation, yet trainable without any parts annotations. The key insight in our design is to disentangle the complex 3D shape generation task into a global shape modeling and a local structure adjustment, to ease the learning process and enhance the shape generation quality. Beyond the existing generative models, our model enables various forms of structure-aware shape manipulations such as part editing and part-wise shape interpolation, and multi-shape part composition, etc.

To sum up, we systematically introduce four novel deep learning methods for enhancing the 3D point cloud representation and artificially enlarging the 3D datasets. Although significant progress has been made in 3D point cloud analysis in the past few years, there are still many important issues that have not been resolved, such as 3D large-scale scene understanding, shape representation learning, and the application to autonomous driving, etc.

摘要

三維點雲，作為三維掃描的默認輸出，被廣泛應用。近年來三維感測技術上取得了長足的進步，它們作為作為處理三維幾何形狀的有效手段越來越受到人們的重視。受各種圖像相關任務數據驅動方法的啟發，近年來已經提出了許多基於深度學習的網絡用於點雲分析，並表現出卓越的性能。本文中將介紹我們在三維點雲分析方面的研究成果，特別是考慮到採集的三維數據質量低下和三維數據集數量有限的情況。

首先，由於深度相機和 LiDAR 傳感器產生的原始點云通常稀疏，嘈雜且不均勻，因此我們提出了一個新的點雲上採樣框架 PU-GAN，該框架將上採樣與數據修正功能相結合。我們利用生成對抗網絡來構造方法，以學習合成在潛在空間中具有均勻分佈的點云，並在對象表面的局部面片上進行上採樣。

其次，在重新討論點雲上採樣任務後，我們建議根據任務的多目標性質對其進行分解，並製定兩個級聯的子網絡，即密集生成器和空間精簡器。密集生成器推斷出粗略但密集的輸出，該輸出粗略地描述了三維表面，而空間精簡器通過調整每個點的位置進一步微調了粗略輸出。與當前的上採樣方法相比，我們的分散式優化管線對每個子網分配的要求較低，因此密集生成器和空間優化器都可以更專注於自己的子目標。

第三，考慮到 3D 數據集的數量和多樣性通常較小，我們提出了 PointAugment，這是一個新的數據自動增強框架，當我們訓練分類網絡時，該框架會自動優化和增強點雲樣本以豐富數據多樣性。與現有的 2D 圖像自動增強方法不同，PointAugment 可以感知樣本，並採用對抗性學習策略來共同優化增強器網絡和分類器網絡，以便增強器可以學習生成最適合分類器的增強樣本。

最後，我們介紹 SP-GAN，這是一種直接合成 3D 點雲的無監督生成模型。SP-GAN 能夠合成具有精細細節的各種高質量形狀，並增強了可感知局部形狀的生成和操縱的可控性，而且無需任何局部標籤即可進行訓練。我們設計的關鍵是將復雜的 3D 形狀生成任務分解為全局形狀建模和局部結構調整，以簡化學習過程並提高形狀生成質量。我們的模型還支持各種形式的結構感知和形狀操縱，例如局部編輯和局部形狀插值等。

綜上所述，我們系統地介紹了四種不同的深度學習方法，以增強 3D 點雲表示和自動放大 3D 數據集。儘管在過去幾年中 3D 點雲分析取得了重大進展，但仍然存在許多尚未解決的重要問題，例如 3D 大規模場景理解，形狀表示學習以及在自動駕駛中的應用等。

Acknowledgments

Having my Ph.D. life at CUHK is an unforgettable and wonderful journey. These fantastic four years of study have been challenging but immensely rewarding for me. I would like to take this opportunity to express my sincere gratitude and appreciation to all those who support me in this memorable journey.

First and foremost, I want to express my deepest gratitude to my supervisor, Prof. Chi-Wing FU, for his incredible guidance during my career of Ph.D. study. He always shows me the right direction but encourages me to go for the best path. His passionate attitude and concentration towards scientific research has significantly influenced the shaping of my mind, particularly for his emphasis on the importance of pursuing highly-impact research, teamwork spirit, and communication skills. With his instruction, I got the courage and inspiration to explore many exciting questions across computer vision and computer graphics. I believe that this positive influence will benefit me lifelong.

I would like to greatly appreciate my thesis committee members Prof. Xu Qiang, Prof. HENG Pheng Ann, and Prof. Zheng Jianmin for all their time and efforts spent on assessing my the-

sis. Their valuable comments and suggestions help me to enhance my research works from various perspectives.

I am appreciate of the great research collaborators and colleagues. Particularly, I would like to express my sincere thanks to Dr. Xianzhi Li, who guides me to start my research journey and also help me how to conduct a high-quality research work. I'd thank many other collaborators and colleagues, including Hao Xu, Lequan Yu, Shufang Wang, Edward Hui, Xiao Tang, Mengyang Wu, Yang Tian, Zhiliang Zeng, Jingyu Hu and many others.

Finally, my deep and sincere gratitude goes to my family and my fiancee for their endless love, help and unparalleled support in all my pursuits. Thank you for always being there for me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

With the rapid development of autonomous driving and robotics industries, making machines understand the real three-dimensional world has become a guarantee for safe and efficient task execution. Point clouds, as a compact representation of 3D data and default outputs of 3D scanners, are widely explored by both traditional and deep-learning-based methods for many applications [1, 2, 3], *e.g.*, self-driving cars, robotics, rendering, and medical analysis. With the increased availability of 3D sensors, point clouds are gaining more and more popularity as an effective means for processing 3D geometry.

Early literatures focus on the processing of a small set of 3D point clouds via hand-crafted optimizers and filters, or various shape priors. However, these traditional techniques often do not generalize well to unseen data, since their capabilities are highly constrained by the pre-defined operations or priors.

On the other hand, due to its irregular and unordered format, however, current convolutional deep-learning methods cannot be directly used with point clouds. Most researchers convert such data to regular 3D voxel grids or collections of images, which renders data unnecessarily voluminous and causes quantization and other issues.

In recent few years, several deep neural networks [4, 5] have been designed to explore the possibility of directly consuming point clouds for geometry understanding and 3D structure recognition. In these works, the deep networks directly extract features from the raw 3D point coordinates and present impressive results for 3D object classification and semantic scene segmentation. Inspired by these pioneering works, a number of novel works [6, 7, 8, 9, 10, 11, 12, 13] have been proposed for assorted tasks on data-driven point cloud analysis. Powered by deep learning algorithms, data-driven methods on point clouds have significantly advanced the field of 3D understanding during the past few years.

Generally, the typical nature of deep neural networks (DNNs) is to learn to approximate the unknown underlying mapping function from the given data distribution. It is a shared knowledge that too little training data and noisy samples lead to a poor approximation. An over-constrained model will underfit the small training dataset, whereas an under-constrained model, in turn, will likely overfit the training data, both resulting in poor performance. Therefore, to robustly train a network often

relies on the quality and quantity of the training data, no matter for 2D images or 3D point clouds. Despite the promising results the deep-learning-based methods have achieved on 3D domain, there are some fundamental issues in this area.

- **Low quality of acquired 3D data**: A couple of varying artifacts, including occlusion, light reflection, surface materials, sensor resolutions, and viewing angles, hinder the acquisition of high-quality point clouds from real-world sensing. Especially when the target object is far from the sensing device or the underlying shape contains rich geometric details. Therefore, it is desirable but challenging to amend the raw data, such as upsampling, denoising, and completion, before it can be effectively used for 3D scene analysis, or general processing.

- **Limited quantity of 3D dataset**: Unlike 2D image benchmarks such as ImageNet [14] and MS COCO dataset [15], which have over millions of training samples and thousands of object categories, 3D datasets are typically much smaller in quantity, with relatively small amount of labels and limited diversity. The limited data quantity and diversity may cause overfitting problem and further affect the generalization ability of the network. Instead of manually collecting and labeling, it is more desirable to enlarge the 3D dataset by augmenting the available training samples or generating new 3D point clouds.

Overall, due to the complexity and costs of fine-grained 3D point cloud collections and annotations, the quality and quantity of existing point cloud datasets are inferior to that of the image datasets, resulting in the overfitting and poor generalization of existing methods on 3D analysis. These inherent drawbacks are still less considered in the point cloud domain and more efforts are required to be done to progress forward.

1.2 Overview of Thesis and Contributions

In this thesis, we focus on designing new and effective deep neural network methods for enhancing the point cloud representation and enlarging the 3D dataset automatically. I mainly present four major first-authored works I have done during my PhD study. These include two works on point cloud upsampling, aiming to produce dense and high-quality point clouds from sparse inputs, the first auto-augmentation framework for 3D point clouds, and a unsupervised 3D shape generation method that allows user to manipulate the GAN-generated shapes. In this section, we provide an overview of the topics discussed in this thesis.

1.2.1 Point Cloud Upsampling

Raw point clouds produced from range scans are often sparse, noisy, and non-uniform. Hence, it is desirable to amend such an imperfect input to provide a more faithful underlying geometric

structures, before it can be effectively used for downstream applications. To this end, we focus on the point cloud upsampling tasks, aiming to generate a denser and uniform set of points from a sparser set of points. In Chapter 3, we presented the first generative point cloud upsampling framework, namely PU-GAN, that combines upsampling with data amendment capability. It is formulated based on a generative adversarial network (GAN), to learn a rich variety of point distributions from the latent space and upsample points over patches on object surfaces.

In Chapter 4, after revisiting the point cloud upsampling task, we formulate an improved method as Dis-PU. We propose to disentangle the task based on its multi-objective nature and formulate two cascaded sub-networks, a dense generator, and a spatial refiner. The dense generator infers a coarse but dense output that roughly describes the underlying surface, while the spatial refiner further fine-tunes the coarse output by adjusting the location of each point.

We conducted extensive experiments to evaluate the effectiveness of both PU-GAN and Dis-PU, and demonstrate that our methods outperform state-of-the-arts on both real-scanned and synthetic inputs.

1.2.2 Point Cloud Auto-Augmentation

In Chapter 5, we present the first method for 3D point cloud augmentation, as an effective way to improve the network gener-

alization, particularly considering the often smaller size of point clouds datasets. Different from traditional augmentation techniques with manually design, our PointAugment is a new auto-augmentation framework for 3D point clouds, and demonstrate its effectiveness to enhance shape classification. The key idea here is to build a deep neural network model that learns to optimize and augment point cloud samples to enrich the data diversity when we train a classification network. It takes an adversarial learning strategy to jointly optimize an augmentor network and a classifier network, such that the augmentor can learn to produce augmented samples that best fit the classifier.

As the first attempt to explore auto-augmentation for 3D point clouds, we show by replacing conventional data augmentation with our PointAugment, clear improvements shown in shape classification and retrieval can be achieved.

1.2.3 Point Cloud Generation

Lastly, we focus on a very challenging task: unsupervised point cloud generation with shape manipulation. In Chapter 6, we present SP-GAN, a new unsupervised sphere-guided generative model for direct synthesis of 3D shapes in the form of point clouds. It characterizes the shape distribution from exist repositories and learns to directly synthesis new shapes and manipulate them in an unsupervised manner, enabling to directly enlarge the 3D dataset.

In SP-GAN, we incorporate a global prior (uniform points on a sphere) to spatially guide the generative process and attach a local prior (a random latent code) to each sphere point to provide local details. Compared with existing models, SP-GAN is able to synthesize diverse and high-quality shapes with fine details and promote controllability for part-aware shape generation and manipulation, yet trainable without any parts annotations. Experimental results, which include both visual and quantitative evaluations, demonstrate that our model is able to synthesize diverse point clouds with fine details and less noise, as compared with the state-of-the-art models.

1.3 Thesis Organization

In the following of this thesis, we first review existing deep neural networks for point cloud processing and related topics in Chapter 2. We then introduce our first two efforts, i.e., PU-GAN and Dis-PU, for point cloud upsampling in Chapter 3 and 4, respectively. In Chapter 5, we present the first auto-augmentation framework in 3D point cloud domain. In Chapter 6, a novel unsupervised model is introduced for point cloud generation and manipulation. Chapter 7 concludes this thesis and also discusses several potential research directions. Lastly, Chapter 8 lists my publications during the Ph.D. study.

□ **End of chapter.**

Chapter 2

Related Work

In this chapter, we first review existing deep neural networks for point cloud analysis (Section 2.1). Next, we elaborate on various methods for point cloud upsampling (Section 2.2). Then, we review the works of data augmentation (Section 2.3). Lastly, we present recent generation networks for 3D shapes (Section 2.4).

2.1 Point Cloud Analysis

The advent of fast 3D point cloud acquisition and rapid advances in 3D sensing encourage the development of deep feature learning on point sets. In this section, we briefly review existing deep neural networks for point cloud analysis. Please refer to [16] for a detailed survey of deep learning on 3D point clouds.

Since the lack of point order and regular structure impede the extension of powerful CNN to 3D point clouds. a lot of existing works convert point clouds to other 3D representations, such as the volumetric grids [17, 18, 19, 20] and geometric graphs [21, 22]

for processing. However, the converting procedure may cause the inevitable information loss.

The pioneering work of Qi *et al.* [4] presented PointNet, the first neural network that operates directly on unordered point cloud data. It is composed of per-point multi-layer perceptrons, a symmetric pooling operation and several fully connected layers. Later on, its variant PointNet++ [5] uses a hierarchical feature learning architecture to capture both local and global geometry context. Subsequently, many other networks were proposed for better feature embedding, *e.g.*, pointwise convolution [23], SpiderCNN [24], KCNet [7], SPLATNet [25], PointCNN [26], DGCNN with EdgeConv block [6], *etc.*. Several other works proposed to transfer points into a regular space for allowing traditional convolutional neural networks to work on, like self-organizing map [8], or conducting the $\mathcal{X}$-transformation [26], and parameterized embedding [24]. Besides the these supervised approaches, some self-supervised and unsupervised networks [27, 28, 29, 30, 31, 32] were introduced recently to alleviate the workload of tedious manual labeling.

On the other hand, inspired by the success of PointNet [33], various deep-learning methods were introduced for assorted point cloud processing tasks, including shape classification [26, 5, 13], segmentation [34, 35, 36], object detection [37, 38, 39], point set generation [40, 41], completion [42, 43, 44], registration [45, 46, 47], and denoising [48, 49], etc.

2.2 Point Cloud Upsampling

3D point clouds, captured by 3D scanning devices or depth sensors, are often sparse, non-uniform, and noisy. Given such a sparse set, the goal of point cloud upsampling is to produce a denser point set to describe the underlying object surface by learning the local structures from training samples. The existing works can be roughly divided into two categories: optimization-based and deep-learning-based.

2.2.1 Optimization-based upsampling methods

To generate new points from the inputs, optimization-based methods typically rely on hand-crafted priors. Alexa *et al.* [50] introduced an early work that inserts new points at the vertices of the Voronoi diagram, which is computed based on the moving-least-squares surface. Later, a locally optimal projection operator (LOP) was developed by Lipman *et al.* [51], where points are resampled based on the L_1 norm. However, the LOP-based methods require surface smoothness, giving rise to performance struggling around sharp edges and corners. To upsample point cloud in an edge-aware manner, Huang *et al.* [52] proposed to first upsample points away from the edges then progressively move points towards the edge singularities, whereas the performance of EAR heavily depends on the given normal values and parameter tuning. Later, Wu *et al.* [53] introduced a point-set consolidation method by augmenting surface points into deep

points that lie on the meso-skeleton of the shape and the optimization is conducted under global geometry constraints. Overall, optimization-based methods may fail when the prior assumptions are not satisfied and be often sensitive to noise and outliers.

2.2.2 Deep-learning-based upsampling methods

Inspired by the success of PointNet [33] and its promising results, taking advantages of deep learning techniques to model the complex geometries has been brought to attention thus far. For the point cloud upsampling task, Yu *et al.* [54] introduce PU-Net, the first attempt based on deep learning to learn to extract multi-scale features and expand a point set via a multi-branch convolution in the feature space. Later, they introduce EC-Net [55], which utilizes an edge-aware technique to process point clouds and simultaneously recovers 3D point coordinates and point-to-edge distances by a joint loss. It achieves edge-aware point cloud upsampling, which further enhances the surface reconstruction quality. Soon after that, Wang *et al.* [56] proposed MPU, a network that progressively upsamples point patches in subsequent upsampling units. However, MPU is computationally expensive due to its progressive nature, and it requires more data to supervise the middle output of the network. Qian *et al.* [57] proposed PUGeoNet to first generate samples in a 2D domain and then use a linear transform to lift up the samples to 3D. PU-GCN [58] is built upon Graph Convolutional

Networks (GCNs), and multiple designed upsampling modules can be incorporated into other upsampling pipelines.

In this thesis, we present a new method PU-GAN by leveraging the generative adversarial network to learn to synthesize points with a uniform distribution in the latent space. Then, to better meet the multi-objective nature of the task, we propose a new approach Dis-PU to disentangle the task into two cascaded networks and demonstrate substantial improvements over the prior works.

2.3 Point Cloud Augmentation

Training data plays a very important role for deep neural networks to learn to perform tasks. However, training data usually has limited quantity, compared with the complexity of our real world, so data augmentation is often needed as a regularization to expand the training set and maximize the knowledge that a network can learn from the training data. However, data augmentation is less considered for 3D point cloud analysis despite many studies designing various augmentation techniques for 2D images. Actually, regularization is essential for point clouds since lack of generality is more likely to occur in point cloud due to small datasets.

2.3.1 Data Augmentation for Images

Various methods have been proposed in the image domain in addition to conventional methods, such as random rotation, flip and crop. Instead of randomly transforming the training data samples [59, 60], some works attempted to generate augmented samples from the original data by using image combination [61], generative adversarial network (GAN) [62, 63], Bayesian optimization [64], and image interpolation in the latent space [65, 66, 67]. However, these methods may produce unreliable samples that are different from those in the original data. On the other hand, some image DA techniques [61, 59, 60] apply pixel-by-pixel interpolation for images with regular structures; however, they cannot handle order-invariant point clouds.

Another approach aims to find an optimal combination of predefined transformation functions to augment the training samples, instead of applying the transformation functions based on a manual design or by complete randomness. AutoAugment [68] suggests a reinforcement learning strategy to find the best set of augmentation functions by alternatively training a proxy task and a policy controller, then applying the learned augmentation function to the input data. Soon after, two other works, FastAugment [69] and PBA [70], explore advanced hyperparameter optimization methods to more efficiently find the best transformations for the augmentation. Tang *et al.* [71] and Zhang *et al.* [72] suggested to learn augmentation policies on

target tasks using an adversarial strategy. They tend to directly maximize the loss of augmented samples to improve the generalization of image classification networks.

2.3.2 Data Augmentation for Point Cloud

Data augmentation has not been extensively explored in the point cloud domain. as randomly scaling, rotation, and jittering. In existing points processing networks, data augmentation mainly include random rotation about the gravity axis, random scaling, and random jittering [33, 5]. These handcrafted rules are fixed throughout the training process, so we may not obtain the best samples to effectively train the network. In this thesis, we present PointAugment, the first auto-augmentation framework for point clouds, aiming to maximize the network learning with 3D points.

2.4 Point Cloud Generation

In this thesis, we focus on 3D shape generation, specifically in the context of generating new and diverse shapes that are not necessarily in the given shape repository but yet look realistic when compared with the existing shapes in the repository. With advances in direct 3D point cloud processing using deep neural networks, as inspired by pioneering works such as [4, 5, 73], *etc.*, several new approaches were proposed recently to generate 3D shapes in the form of point clouds. These methods can be

roughly classified into three types, *i.e.*, autoregressive-based, flow-based, and GAN-based.

Autoregressive-based generative approach models the joint 3D spatial distribution of points in a point cloud. Specifically, Sun *et al.* [74] propose PointGrow to estimate the point coordinate distributions from the training shapes. In the generative phase, points are sampled one-by-one based on the estimated probability distributions given the previously-generated points. However, due to the iterative intrinsic of autoregressive models, the model cannot scale well with the size of the point cloud.

Flow-based generative approach learns to model the distribution of points in a shape mainly by an invertible parameterized transformation of the points. In the generative phase, the approach samples points from a given generic prior (*e.g.*, a Gaussian distribution) and moves them to the target shape using the learned parameterized transformation. For example, Yang *et al.* [41] generate point clouds from a standard 3D Gaussian prior based on continuous normalizing flows. Klokov *et al.* [75] relieve the computation by formulating a model using discrete normalizing flows with affine coupling layers [76]. To better characterize the data distribution, Kim *et al.* [77] propose to learn a conditional distribution of the perturbed training shapes. Cai *et al.* [78] model the gradient of the log-density field of shapes and generate point clouds by moving the sampled points towards the high-likelihood regions.

GAN-based generative approach explores adversarial learning to train the shape generation model with the help of a discriminator. Achlioptas *et al.* [79] introduce the first set of deep generative models to produce point clouds from a Gaussian noise vector, including an r-GAN that operates on a raw point cloud input and an l-GAN that operates on the bottleneck latent variables of a pre-trained autoencoder. To overcome the redundancy and structural irregularity of point samples, Ramasinghe *et al.* [80] propose Spectral-GANs to synthesize shapes using a spherical-harmonics-based representation. Shu *et al.* [81] propose tree-GAN to perform graph convolutions in a tree and Gal *et al.* [82] recently extend it into a multi-rooted version. Hui *et al.* [83] design a progressive deconvolution network to generate 3D point clouds, while Arshad *et al.* [84] create a conditional generative adversarial network to produce dense colored point clouds in a progressive manner.

Compared with the above GAN-based approaches, which attempt to synthesize point clouds from only a single latent code, in this thesis, we incorporate a fixed prior shape to guide the generative process in the form of a disentanglement model. Our model not only produces high-quality outputs with fine details but also promotes controllability for structure-aware shape generation and manipulation.

□ **End of chapter.**

Chapter 3

PU-GAN: a Point Cloud Upsampling Adversarial Network

3.1 Introduction

Point clouds, as a compact representation of 3D data, are widely explored by both traditional and deep-learning-based methods for many applications [1, 2, 3], *e.g.*, self-driving cars, robotics, rendering, and medical analysis, etc. However, raw point clouds produced by 3D scanning are often locally sparse and non-uniform, possibly with small holes on the object surface. This is evidenced in various public benchmark datasets, such as KITTI [85], SUN RGB-D [86], and ScanNet [87]. Obviously, we need to amend such raw data, before we can effectively use it for rendering, analysis, or general processing.

The goal of point cloud upsampling is not limited to generat-

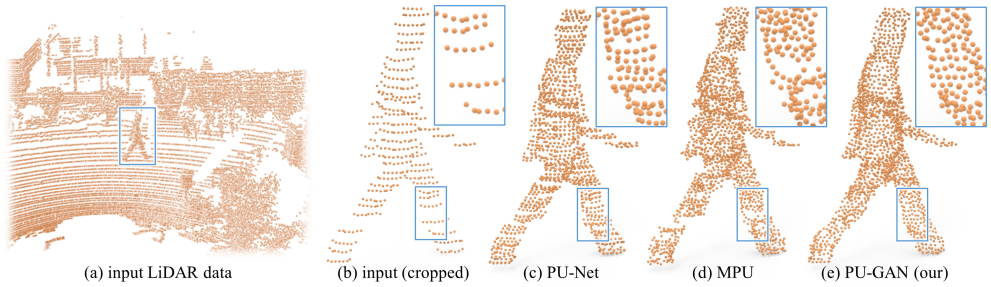


Figure 3.1: Upsampling (b) a point set cropped from (a) the real-scanned KITTI dataset [85] by using (c) PU-Net [54], (d) MPU [56], and (e) our PU-GAN. Note the distribution non-uniformity and sparsity in the input.

ing a dense point set from the sparse input. Very importantly, the generated points should also faithfully locate on the underlying surface and cover the surface with a uniform distribution. As an inference-based problem, these goals are very demanding, due to the limited information available in the sparse input. Besides being sparse, the input points can be non-uniform and noisy, and they may not well represent fine structures (if any) on the underlying surface.

Early methods [50, 51, 88, 52, 53] for the problem are optimization-based, where various shape priors are used to constrain the point cloud generation. Recently, deep neural networks brought the promise of data-driven approaches to the problem. Network architectures, including PU-Net [54], EC-Net [55], and very recently, MPU [56], have demonstrated the advantage of upsampling point clouds through learning. However, these networks may not produce plausible results from low-quality inputs that are particularly sparse and non-uniform; in Figure 3.1 (c)-(e),

we show a typical example that demonstrates the advantage of our method, which unlike the other networks, it combines completion and uniformity together with upsampling.

In this chapter, we present a new point cloud upsampling framework, namely PU-GAN, that combines upsampling with data amendment capability. The key contribution is an adversarial network to enable us to train a generator network to learn to produce a rich variety of point distributions from the latent space, and also a discriminator network to help implicitly evaluate the point sets produced from the generator against the latent point distribution. Particularly, the adversarial learning strategy of the GAN framework can regularize the predictions from a global perspective and implicitly penalize outputs that deviate from the target.

However, successfully training a working GAN framework is known to be challenging [89, 90], particularly the difficulty to balance between the generator and discriminator and to avoid the tendency of poor convergence. Thus, we first improve the point generation quality, or equivalently the feature expansion capability, of the generator, by constructing an up-down-up unit to expand point features by upsampling also their differences for self-correction. Besides, we formulate a self-attention unit to enhance the feature integration quality. Lastly, we further design a compound loss to train the network end-to-end with adversarial, uniform, and reconstruction terms to enhance the distribution uniformity of the results and encourage the discriminator

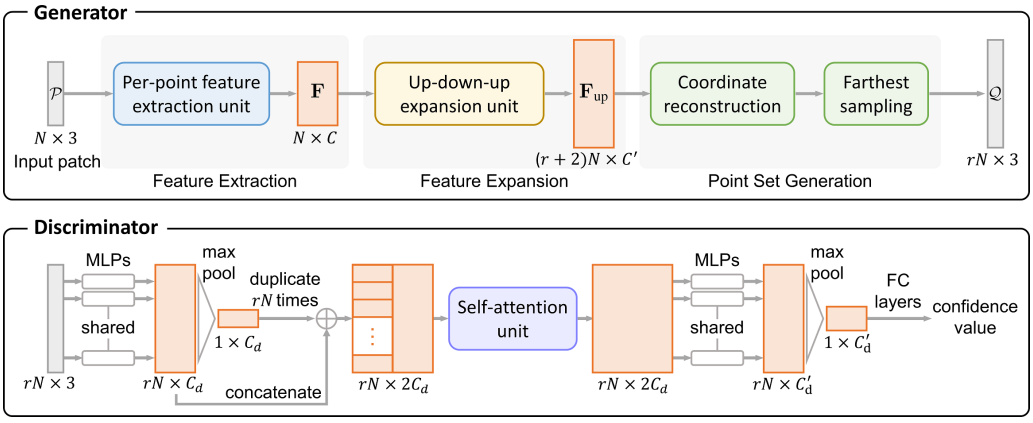


Figure 3.2: Overview of PU-GAN's generator and discriminator architecture. Note that N is the number of points in input $\mathcal{P}$; r is the upsampling rate; and C, C', C_d, and C'_d are the number of feature channels that are 480, 128, 64, and 256, respectively, in our implementation.

to learn more latent patterns in the target distribution.

To evaluate the upsampling quality of PU-GAN, we employ four metrics to assess its performance against the state-of-the-arts on a variety of synthetic and real-scanned data. Extensive experimental results show that our method outperforms others in terms of distribution uniformity, proximity-to-surface, and 3D reconstruction quality.

3.2 Method

3.2.1 Overview

Given an unordered sparse point set $\mathcal{P} = \{p_i\}_{i=1}^{N}$ of N points, we aim to generate a dense point set $\mathcal{Q} = \{q_i\}_{i=1}^{rN}$ of rN points, where r is the upsampling rate. While output $\mathcal{Q}$ is not necessarily a superset of $\mathcal{P}$, we want it to satisfy two requirements: (i) $\mathcal{Q}$

should describe the *same underlying geometry* of a latent target object as $\mathcal{P}$, so points in $\mathcal{Q}$ should lie on and cover the target object surface; and (ii) points in $\mathcal{Q}$ should be *uniformly-distributed* on the target object surface, even for sparse and non-uniform input $\mathcal{P}$.

Figure 3.2 shows the overall network architecture of PU-GAN, where the generator produces dense output $\mathcal{Q}$ from sparse input $\mathcal{P}$, and the discriminator aims to find the fake generated ones. Following, we first elaborate on the architecture of the generator and discriminator (Section 5.3.1). We then present two building blocks in our architecture: the up-down-up expansion unit (Sections 3.2.3) and the self-attention unit (Section 3.2.4). Lastly, we present the patch-based training strategy with the compound loss (Section 3.2.5).

3.2.2 Network Architecture

Generator

As shown on top of Figure 3.2, our generator network has three components to successively process input $\mathcal{P}$:

The feature extraction component aims to extract features $\mathbf{F}$ from input $\mathcal{P}$ of $N \times d$, where d is the number of dimensions in the input point attributes, *i.e.*, coordinates, color, normal, *etc.* Here, we focus on the simplest case with $d = 3$, considering only the 3D coordinates, and adopt the recent feature extraction method in [56], where dense connections are introduced to

integrate features across different layers.

The feature expansion component expands $\mathbf{F}$ to produce the expanded features $\mathbf{F}_{\text{up}}$; here, we design the *up-down-up expansion unit* (see Figure 3.2 (top)) to enhance the feature variations in $\mathbf{F}_{\text{up}}$, enabling the generator to produce more diverse point distributions; see Section 3.2.3 for details.

The point set generation component first regresses a set of 3D coordinates from $\mathbf{F}_{\text{up}}$ via a set of multilayer perceptrons (MLPs). Since the feature expansion process is still local, meaning that the features in $\mathbf{F}_{\text{up}}$ (or equivalently, points in the latent space) are intrinsically close to the inputs, we thus include a farthest sampling step in the network to retain only rN points that are further away from one another; see the green boxes in Figure 4.2. To allow this selection, when we expand $\mathbf{F}$ to $\mathbf{F}_{\text{up}}$, we actually generate $(r+2)N$ features in $\mathbf{F}_{\text{up}}$. This strategy further enhances the point set distribution uniformity, particularly from a global perspective.

Discriminator

The goal of the discriminator is to distinguish whether its input (a set of rN points) is produced by the generator. To do so, we first adopt the basic network architecture in [42] to extract the global features, since it efficiently combines the local and global information, and ensures a lightweight network. To improve the feature learning, we add a self-attention unit (see

Section 3.2.4) after the feature concatenation; see the middle part in Figure 4.2 (bottom). Compared with the basic MLPs, this attention unit helps enhance the feature integration and improve the subsequent feature extraction capability. Next, we apply a set of MLPs and a max pooling to produce the global features, and further regress the final confidence value via a set of fully connected layers. If this confidence value is close to 1, the discriminator predicts that the input likely comes from a target distribution with high confidence, and otherwise from the generator.

3.2.3 Up-down-up Expansion Unit

To upsample a point set, PU-Net [54] duplicates the point features and uses separate MLPs to process each copy independently. However, the expanded features would be too similar to the inputs, so affecting the upsampling quality. Rather than a single-step expansion, the very recent MPU method [56] breaks a 16× upsampling network into four successive 2× upsampling subnets to progressively upsample in multiple steps. Though details are better preserved in the upsampled results, the training process is complex and requires more subsets for a higher upsampling rate.

In this work, we construct an up-down-up expansion unit to expand the point features. To do so, we first upsample the point features (after MLPs) to generate $\mathbf{F}'_{\text{up}}$ and downsample it back

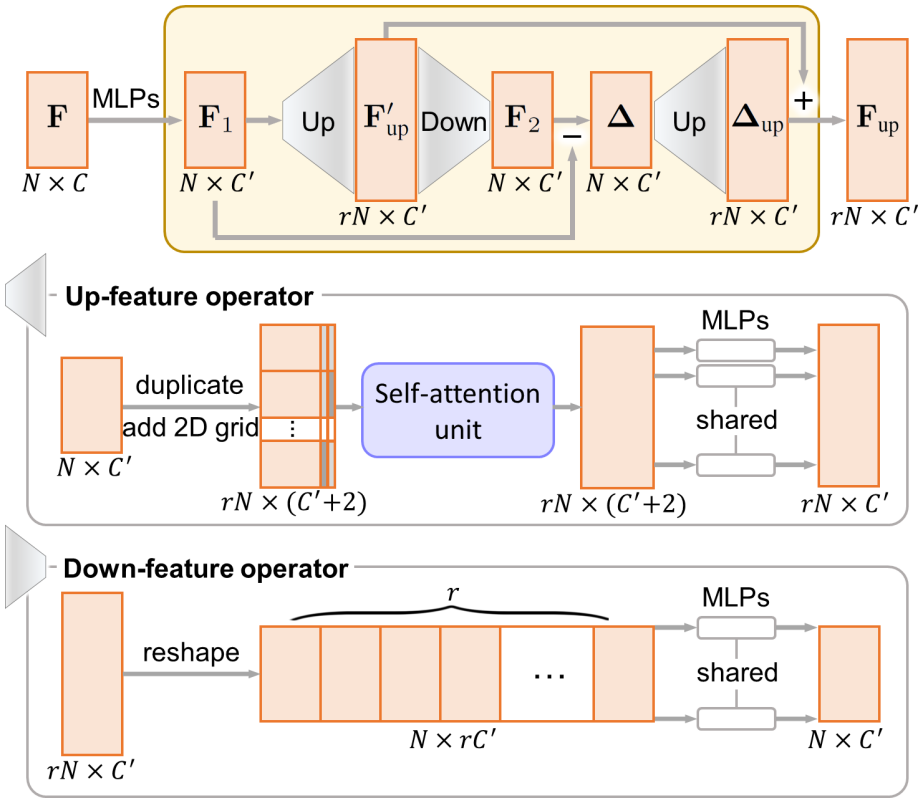


Figure 3.3: The up-down-up expansion unit (top), up-feature operator (middle), and down-feature operator (bottom).

(see Figure 3.3 (top)); then, we compute the difference (denoted as Δ) between the features before the upsampling and after the downsampling. By also upsampling Δ to Δ_{up}, we add Δ_{up} to $\mathbf{F}'_{\text{up}}$ to self-correct the expanded features; see again Figure 3.3 (top) for the whole procedure. Such a strategy not only avoids tedious multi-step training, but also facilitates the production of fine-grained features.

Next, we discuss the up-feature and down-feature operators in the up-down-up expansion unit (see also Figure 3.3):

Up-feature operator. To upsample the point features r times,

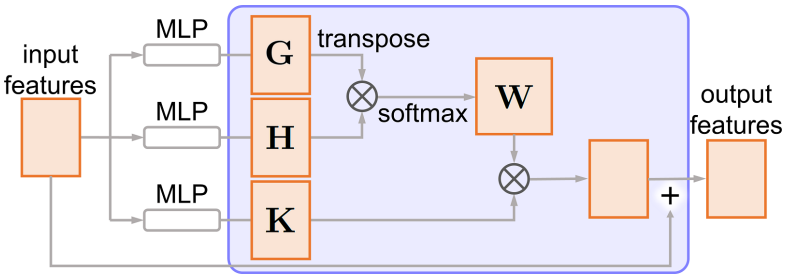


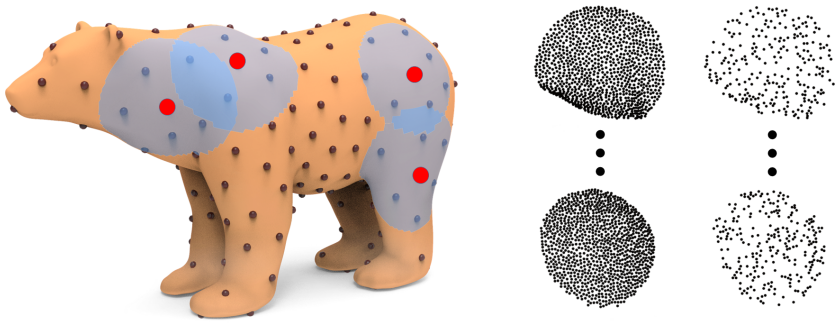
Figure 3.4: Illustration of the self-attention unit.

we should increase the variations among the duplicated features. This is equivalent to pushing the new points away from the input ones. After we duplicate the input feature map (of N feature vectors and C' channels) r times, we adopt the 2D grid mechanism in FoldingNet [27] to generate a unique 2D vector per feature-map copy, and append such vector to each point feature vector in the same feature-map copy; see Figure 3.3 (middle). Further, we use the self-attention unit (see Section 3.2.4) followed by a set of MLPs to produce the output upsampled features.

Down-feature operator. To downsample the expanded features, we reshape the features and then use a set of MLPs to regress the original features; see Figure 3.3 (bottom).

3.2.4 Self-attention Unit

To introduce long-range context dependencies to enhance feature integration after concatenation, we adopt the self-attention unit [91] in the generator (see Figure 3.3 (middle)), as well as in the discriminator (see Figure 4.2 (bottom)). Figure 3.4 presents



(a) Example patches (b) $\hat{\mathcal{Q}}$ (c) $\mathcal{P}$

Figure 3.5: (a) Seed points (black dots) and patches (blue disks) on a 3D mesh in training data. (b) & (c) Example $\hat{\mathcal{Q}}$ and $\mathcal{P}$ on patches.

its architecture. Specifically, we transform the input features into $\mathbf{G}$ and $\mathbf{H}$ via two separate MLPs, and then generate the attention weights $\mathbf{W}$ from $\mathbf{G}$ and $\mathbf{H}$ by

$$\mathbf{W} = f_{\text{softmax}}(\mathbf{G}^T\mathbf{H}) \ , \tag{3.1}$$

where f_{softmax} denotes the softmax function. After that, we obtain the weighted features $\mathbf{W}^T\mathbf{K}$, where $\mathbf{K}$ is the set of features extracted from the input via another MLP. Lastly, we generate the output features, which is the sum of the input features and the weighted features.

3.2.5 Patch-based End-to-end Training

Training data preparation

We trained our network to upsample local groups of points over patches on object surface. Specifically, for each 3D mesh (normalized in a unit sphere) in training set (see Section 3.3.1), we use randomly find 200 seed positions on each mesh surface,

geodesically grow a patch from each seed (each patch occupies ∼5% of the object surface), and then normalize each patch within a unit sphere; see Figure 3.5(a). On each patch, we further use Poisson disk sampling [92] to generate $\hat{\mathcal{Q}}$, which is a target point set of rN points on the patch. During the training, we generate the network input $\mathcal{P}$ by randomly selecting N points on-the-fly from $\hat{\mathcal{Q}}$.

Loss functions

We now present the compound loss designed for training PU-GAN in an end-to-end fashion.

Adversarial loss. To train the generator network G and discriminator network D in an adversarial manner, we use the least-squared loss [93] as our adversarial loss:

$$\mathcal{L}_{\text{gan}}(G) = \frac{1}{2}[D(\mathcal{Q}) - 1]^2 \tag{3.2}$$

$$\text{and } \mathcal{L}_{\text{gan}}(D) = \frac{1}{2}[D(\mathcal{Q})^2 + (D(\hat{\mathcal{Q}}) - 1)^2], \tag{3.3}$$

where $D(\mathcal{Q})$ is the confidence value predicted by D from generator output $\mathcal{Q}$. During the network training, G aims to generate $\mathcal{Q}$ to fool D by minimizing $\mathcal{L}_{\text{gan}}(G)$, while D aims to minimize $\mathcal{L}_{\text{gan}}(D)$ to learn to distinguish $\mathcal{Q}$ from $\hat{\mathcal{Q}}$.

Uniform loss. The problem of learning to generate point sets in 3D is complex with an immense space for exploration during the network training. Particularly, we aim for uniform distributions; using the adversarial loss alone is hard for the

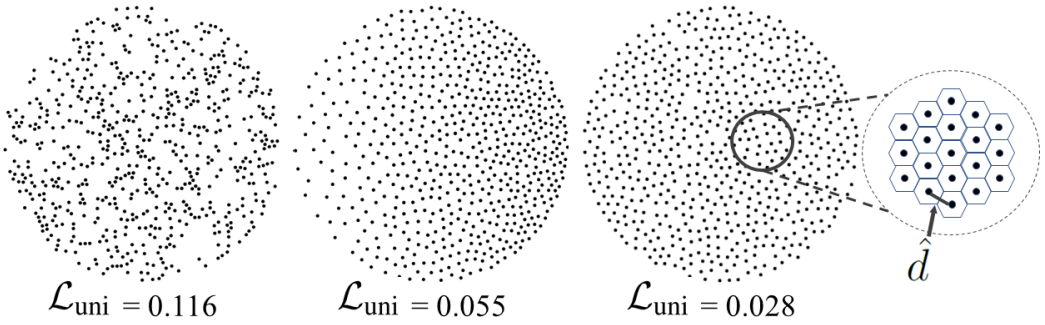


Figure 3.6: Example point sets with same number of points (625) but different point distribution patterns; $\mathcal{L}_{\text{uni}}$ is computed with p=1%.

network to converge well. Thus, we formulate a uniform loss to evaluate $\mathcal{Q}$ from the generator, aiming to improve the generative ability of the generator.

To evaluate a point set's uniformity, the NUC metric in PU-Net [54] crops equal-sized disks on the object surface and counts the number of points in the disks. So, the metric neglects the local clutter of points. Figure 3.6 shows three patches of points with very different point distributions; since they contain the same number of points, the NUC metric cannot distinguish their distribution uniformity.

Our method evaluates $\mathcal{Q}$ (a patch of rN points) during the network training by first using the farthest sampling to pick M seed points in $\mathcal{Q}$ and using a ball query of radius r_d to crop a point subset (denoted as S_j, $j = 1..M$) in $\mathcal{Q}$ at each seed. Here, we use a small r_d, so S_j roughly lies on a small local disk of area πr_d^2 on the underlying surface. On the other hand, since we form patches by geodesics and normalize them in a unit sphere, patch area is $\sim\pi 1^2$. So, the expected percentage of points in

S_j (denoted as p) is $\pi r_d^2/\pi 1^2 = r_d^2$, and the expected number of points in S_j (denoted as $\hat{n}$) is rNp. Hence, we follow the chi-squared model to measure the deviation of $|S_j|$ from $\hat{n}$, and define

$$U_{\text{imbalance}}(S_j) = \frac{(|S_j| - \hat{n})^2}{\hat{n}} \ . \tag{3.4}$$

To account for local point clutter, for each point in S_j, we find its distance to the nearest neighbor, denoted as $d_{j,k}$ for the k-th point in S_j. If S_j has a uniform distribution, the expected point-to-neighbor distance $\hat{d}$ should be roughly $\sqrt{\frac{2\pi r_d^2}{|S_j|\sqrt{3}}}$, which is derived based on the assumption that S_j is flat and neighboring points are hexagonal; see Figure 3.6 (right) for an illustration. Again, we follow the chi-squared model to measure the deviation of $d_{j,k}$ from $\hat{d}$, and define

$$U_{\text{clutter}}(S_j) = \sum_{k=1}^{|S_j|} \frac{(d_{j,k} - \hat{d})^2}{\hat{d}} \ . \tag{3.5}$$

Here, U_{clutter} accounts for the local distribution uniformity, while $U_{\text{imbalance}}$ accounts for the nonlocal uniformity to encourage better point coverage. Putting them together, we formulate the uniform loss as

$$\mathcal{L}_{\text{uni}} = \sum_{j=1}^{M} U_{\text{imbalance}}(S_j) \cdot U_{\text{clutter}}(S_j) \ . \tag{3.6}$$

Figure 3.6 shows three example point sets with the same number of points but different point distribution patterns. Using $\mathcal{L}_{\text{uni}}$, we can distinguish the point uniformity among them.

Reconstruction loss. Both the adversarial and uniform losses do not encourage the generated points to lie on the target surface. Thus, we include a reconstruction loss using the Earth Mover's distance (EMD) [94]:

$$\mathcal{L}_{\text{rec}} = \min_{\phi:\mathcal{Q}\to\hat{\mathcal{Q}}} \sum_{q_i\in\mathcal{Q}} \|q_i - \phi(q_i)\|_2, \tag{3.7}$$

where $\phi : \mathcal{Q} \to \hat{\mathcal{Q}}$ is the bijection mapping.

Compound loss. Overall, we train PU-GAN end-to-end by minimizing $\mathcal{L}_G$ for generator and $\mathcal{L}_D$ for discriminator:

$$\mathcal{L}_G = \lambda_{\text{gan}}\mathcal{L}_{\text{gan}}(G) + \lambda_{\text{rec}}\mathcal{L}_{\text{rec}} + \lambda_{\text{uni}}\mathcal{L}_{\text{uni}}, \tag{3.8}$$

$$\text{and } \mathcal{L}_D = \mathcal{L}_{\text{gan}}(D), \tag{3.9}$$

where λ_{gan}, λ_{rec}, and λ_{uni} are weights. During the training process, G and D are optimized alternatively.

3.3 Experiments

3.3.1 Datasets and Implementation Details

We collected 147 3D models from the released datasets of PU-Net [54] and MPU [56], as well as from the Visionair repository [95], covering a rich variety of objects, ranging from simple and smooth models (*e.g.*, Icosahedron) to complex and high-detailed objects (*e.g.*, Statue). Among them, we randomly select 120 models for training and use the rest for testing.

In the training phase, we cropped 200 patches per training model (see Section 3.2.5) and produced $24,000$ patches in total.

By default, we set $N = 256$, $r = 4$, and $M = 50$. Moreover, for the uniform loss, we cropped one set of S_j with radius $r_d = \sqrt{p}$ for each $p \in \{ 0.4\%, 0.6\%, 0.8\%, 1.0\%, 1.2\% \}$, compute Eq. (3.6) five times for each set, and then sum up the results as the $\mathcal{L}_{\text{uni}}$ term in Eq. (3.8).

To avoid overfitting in training, we augment the network inputs by random rotation, scaling, and point perturbation with Gaussian noise. We trained the network for 100 epochs using the Adam algorithm [96] with a two time-scale update rule (TTUR) [97]. We set the learning rates of generator and discriminator as 0.001 and 0.0001, respectively; after 50k iterations, we gradually reduce both rates by a decay rate of 0.7 per 50k iterations until 10^{-6}. The batch size is 28, and λ_{rec}, λ_{uni}, and λ_{gan} are empirically set as 100, 10, and 0.5, respectively. We implemented our network using TensorFlow and trained it on NVidia Titan Xp GPU.

During the testing, given a point set, we follow the patch-based strategies in MPU [56] and EC-Net [55] to use the farthest sampling to pick seeds and extract a local patch of N points per seed. Then, we feed the patches to the generator and combine the upsampled results as the final output.

3.3.2 Evaluation Metrics

We employ four evaluation metrics: (i) uniformity using $\mathcal{L}_{\text{uni}}$ in Eq. (3.6), (ii) point-to-surface (P2F) distance using the testing

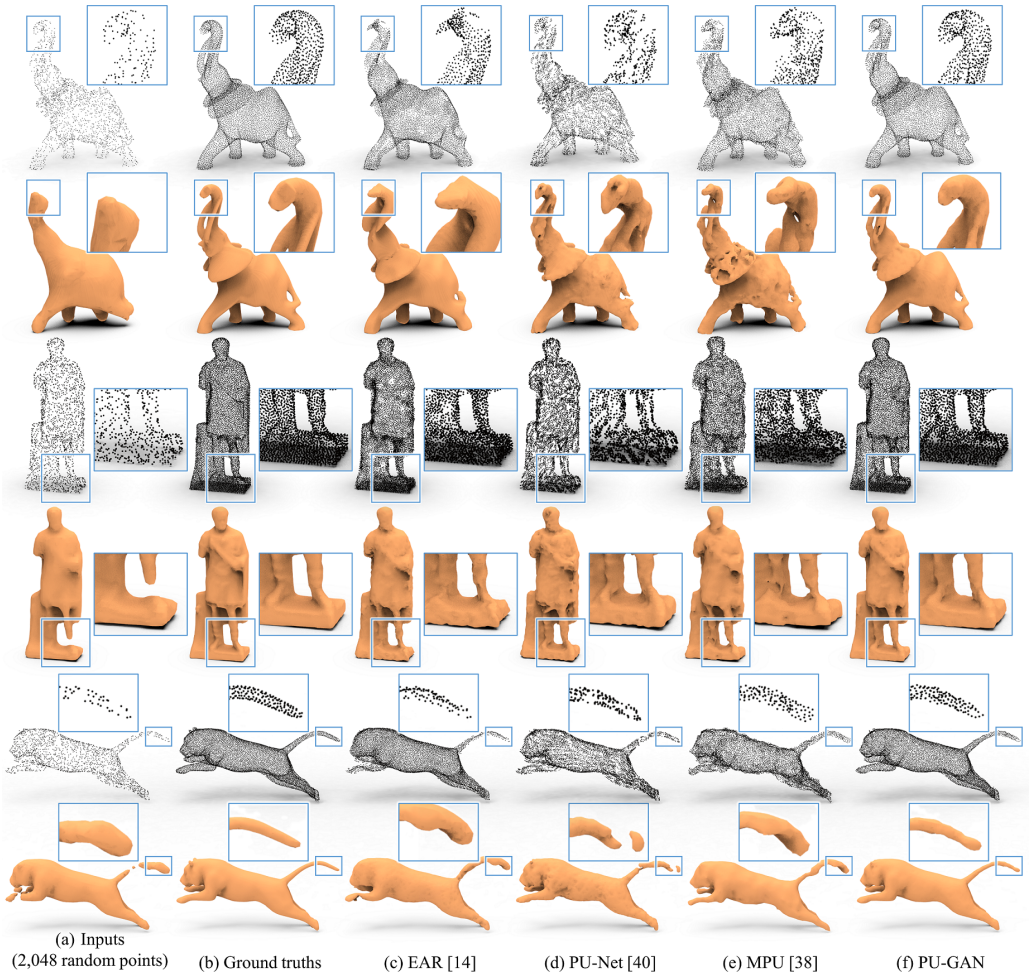


Figure 3.7: Comparing point set upsampling (x4) and surface reconstruction results produced with different methods (c-f) from inputs (a).

models, (iii) Chamfer distance (CD), and (iv) Hausdorff distance (HD) [98]. For quantitative evaluation, we use Poisson disk sampling to sample 8,192 points as the ground truth (*e.g.*, see Figure 3.7(b)) on each of the testing models, and randomly select 2,048 points as testing input. To evaluate the uniformity of the test results (using Eq. (3.6)), we randomly pick $M = 1,000$ seeds on each result, and instead of using ball query to crop S_j, we

Table 3.1: Quantitative comparisons with the state-of-the-arts.

Methods	Uniformity for different p (10^{-3})					P2F	CD	HD
	0.4%	0.6%	0.8%	1.0%	1.2%	(10^{-3})	(10^{-3})	(10^{-3})
EAR [52]	16.84	20.27	23.98	26.15	29.18	5.82	0.52	7.37
PU-Net [54]	29.74	31.33	33.86	36.94	40.43	6.84	0.72	8.94
MPU [56]	7.51	7.41	8.35	9.62	11.13	3.96	0.49	6.11
PU-GAN	**3.38**	**3.49**	**3.44**	**3.91**	**4.64**	**2.33**	**0.28**	**4.64**

use the actual mesh (testing model) to geodesically find S_j at each seed of different p for higher-quality evaluation. The lower the metric values are, the better the upsampling results are.

3.3.3 Qualitative and Quantitative Comparisons

We qualitatively and quantitatively compared PU-GAN with three state-of-the-art point set upsampling methods: EAR [52], PU-Net [54], and MPU [56]. For EAR, we employed the released demo code and generated the best results by exhaustively fine-tuning every associated parameter. For PU-Net and MPU, we used their public code and retrained their networks using our training data.

Table 6.1 shows the quantitative comparison results. Our PU-GAN achieves the lowest values consistently for all the evaluation metrics. Particularly, the uniformity of our results stays the lowest for all different p, indicating that the points generated by PU-GAN are much more uniform than those produced by others over varying scales.

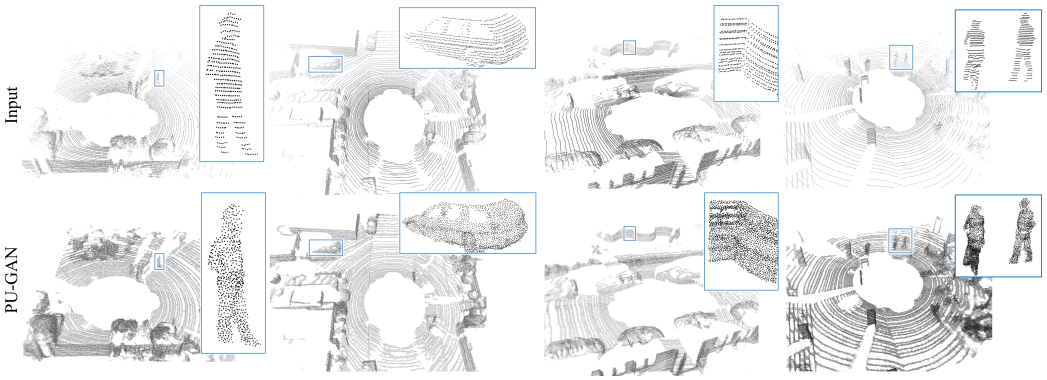


Figure 3.8: Using PU-GAN to upsample real-scanned point cloud data acquired by LiDAR.

Besides quantitative results, we show point set upsampling and surface reconstruction (using [99]) results for various models in Figure 3.7. Comparing the results produced by (f) our method and by (c-e) others, against (b) the ground truth points that are uniformly-sampled on the original testing models, we can see that other methods tend to produce more noisy and nonuniform point sets, thus leading to more artifacts in the reconstructed surfaces. See, particularly, the blown-up views, showing that PU-GAN can produce more fine-grained details in the upsampled results, *e.g.*, elephant's nose and tiger's tail.

3.3.4 Upsampling Real-scanned Data

Figure 3.8 shows upsampling results on LiDAR point clouds (downloaded from KITTI dataset [85]) produced by PU-GAN. From the first row, we can see the sparsity and non-uniformity of the inputs. Our PU-GAN can still fill some holes and output more uniform points in the results.

Table 3.2: Quantitative comparisons: removing each specific component from the full PU-GAN pipeline (top five rows) vs. baseline GAN model (6-th row) vs. full PU-GAN pipeline (last row).

	Uniformity for different p (10^{-3})					P2F	CD	HD
	0.4%	0.6%	0.8%	1.0%	1.2%	(10^{-3})	(10^{-3})	(10^{-3})
Discriminator	7.02	7.31	8.36	9.70	11.17	4.61	0.57	7.25
$\mathcal{L}_{\text{uni}}$	5.15	5.71	6.13	6.82	7.32	3.99	0.51	6.22
self-attention	4.19	4.66	4.87	5.52	6.47	3.97	0.46	6.01
up-down-up	3.89	4.16	4.63	5.14	5.89	3.02	0.35	5.15
farthest samp.	3.56	3.76	3.78	4.15	4.97	2.72	0.31	4.96
baseline GAN	8.12	8.18	8.76	9.89	11.32	4.79	0.59	7.31
full pipeline	**3.38**	**3.49**	**3.44**	**3.91**	**4.64**	**2.33**	**0.28**	**4.64**

3.3.5 Ablation Study and Baseline Comparison

To evaluate the components in PU-GAN, including the GAN framework (*i.e.*, remove the discriminator and keep only the generator), up-down-up expansion unit, self-attention unit, uniform loss, and farthest sampling, we removed each of them from the network and generated upsampling results for the testing models. Table 3.2 shows the evaluation results. Our full pipeline performs the best, and removing any component reduces the overall performance, meaning that each component contributes. Particularly, removing the GAN framework (*i.e.*, removing the discriminator) causes the largest performance drop, thereby showing the effectiveness of adversarial learning in the framework.

Further, we designed a GAN baseline for comparison by removing $\mathcal{L}_{\text{uni}}$, self-attention unit, up-down-up expansion unit,

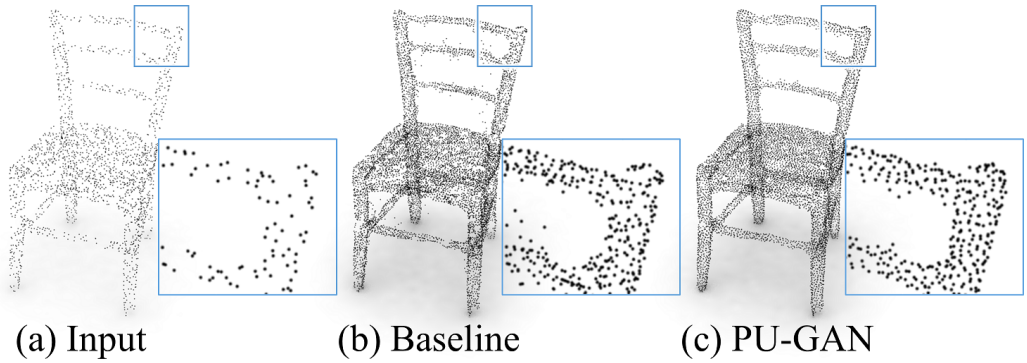


Figure 3.9: Visual comparison of (c) PU-GAN against (b) the baseline GAN model, given (a) an input of 2,048 non-uniform points.

and farthest sampling altogether. Quantitative results shown in the second last row of Table 3.2 and a visual comparison example shown in Figure 3.9 demonstrate that simply applying a basic GAN framework is insufficient for the upsampling problem; our adaptations to the GAN model plus the various formulations contribute to realizing a working GAN model.

3.3.6 Other Experiments

Upsampling point sets of varying noise levels. Figure 4.6 shows the results of using PU-GAN to upsample point sets of increasing Gaussian noise levels, indicating the robustness of PU-GAN to noise and sparsity.

Upsampling point sets of varying sizes. Figure 3.11 shows the results of upsampling input point sets of different sizes; our method is stable even for input with only 512 points.

Applications. Besides surface reconstruction and rendering (see Figure 3.7), point cloud upsampling is also helpful for object

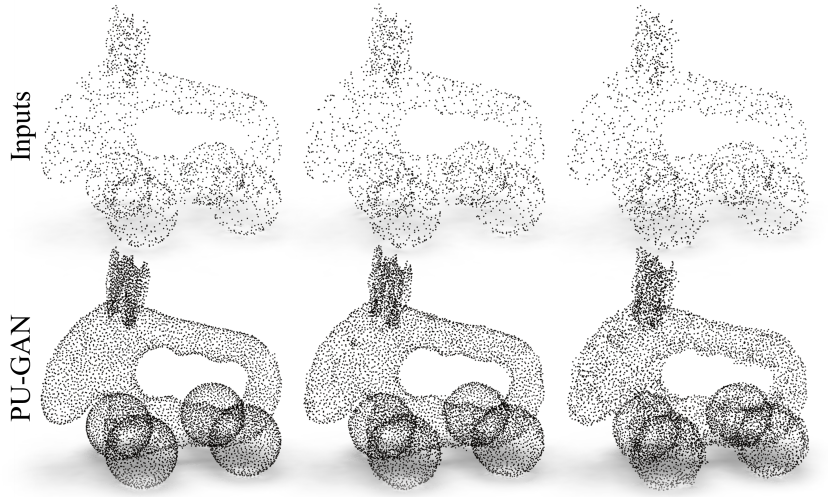


Figure 3.10: Upsampling results by applying PU-GAN to inputs with noise level of 0, 0.5%, and 1%.

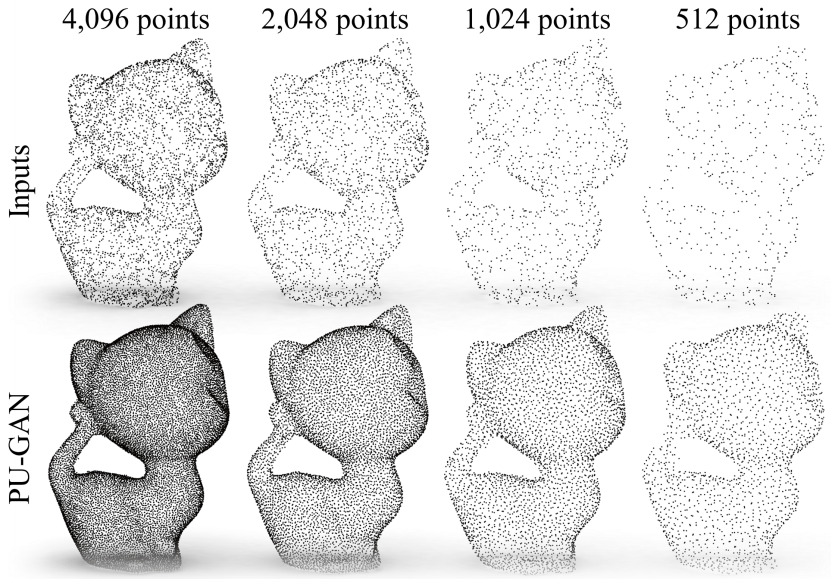


Figure 3.11: Upsampling point sets (top row) of varying sizes.

recognition. To demonstrate this, we trained PointNet (vanilla version) [4] on ModelNet40 dataset [18] for classification under two cases: in case (i), we directly trained PointNet to take sparse training set (512 points) as inputs for classifying objects in the sparse testing set; and in case (ii), we upsampled the point

clouds in both sparse training and testing sets to 2,048 points using PU-GAN and then trained another PointNet with the upsampled training set for classifying the upsampled data in the testing set. Results show that, the overall classification accuracy increased from 82.4% (case (i)) to 83.8% (case (ii)), indicating that PU-GAN helps improve the classification performance.

3.4 Summary

In this chapter, we presented, PU-GAN, a novel GAN-based point cloud upsampling network, that combines upsampling with data amendment capabilities. Such adversarial network enables the generator to produce a uniformly-distributed point set, and the discriminator to implicitly penalize outputs that deviate from the expected target. To facilitate the GAN framework, we introduced an up-down-up unit for feature expansion with error feedback and self-correction, as well as a self-attention unit for better feature fusion. Further, we designed a compound loss to guide the learning of the generator and discriminator. We demonstrated the effectiveness of our PU-GAN via extensive experiments, showing that it outperforms the state-of-the-art methods for various metrics, and presented the upsampling performance on real-scanned LiDAR inputs.

□ **End of chapter.**

Chapter 4

Point Cloud Upsampling via Disentangled Refinement

4.1 Introduction

The previous chapter describes a novel deep network architecture for point clouds upsampling called PU-GAN. It leverages the generative adversarial network to learn to synthesize points with a uniform distribution in the latent space. Benefited from the novel design, PU-GAN demonstrated superior performance, compared with traditional methods [50, 51, 52] and previous learning-based methods [54, 55, 56]. Later on, several advanced models [57, 58] are proposed to improve the performance.

The general approach taken in these learning-based methods is that they first design an upsampling module to expand the number of points in the feature space, then formulate losses to constrain the output points for the distribution uniformity and proximity-to-surface. In other words, the designed upsampling module is expected not only to infer and generate dense points

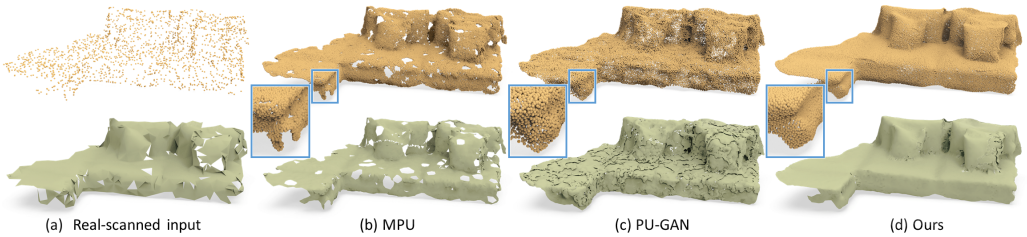


Figure 4.1: In the top row, we show (a) a sparse real-scanned point set from [100], followed by upsampled results (16×) produced by (b) MPU [56], (c) PU-GAN [101], and (d) our method. In the bottom row, we show the associated reconstructed 3D meshes produced by the ball-pivoting algorithm [102]. Clearly, our method outperforms others on the local uniformity, contributing a better surface reconstruction.

from the sparse input, but also to produce points that are uniform, clean, and faithfully located on the underlying surface. However, it is very hard for a network to meet all the requirements at the same time. Therefore, the dense points produced by existing works still tend to be non-uniform or retain excessive noise (see the top results in Figures 4.1 (b) & (c)), thus resulting in low-quality reconstructed meshes (results in the bottom row).

After revisiting the point cloud upsampling task, we propose to disentangle the task into two sub-goals: (i) to first generate a *coarse but dense point set* with less details to roughly describe the underlying surface, and then (ii) to *refine the coarse points* to better cover the underlying surface for distribution uniformity and proximity-to-surface. To do so, we formulate an end-to-end disentangled refinement framework, which consists of two cascaded sub-networks, *a dense generator* and *a spatial refiner*, which are designed to aim for sub-goals (i) and (ii), respectively.

Particularly, we design the spatial refiner with a pair of local and global refinement units to evolve the coarse feature map inside the refiner to take into account both the local and global geometric structures. Further, inspired by the residual-learning strategy [103], we formulate the spatial refiner to regress a per-point offset vector for fine-tuning the coarse outputs by adjusting the location of each point. Compared with directly predicting the final refined 3D point coordinates, regressing a small residual is much easier for the network.

Compared with current upsampling methods [54, 56, 101], our disentangled refinement pipeline assigns lower requirements to each sub-network, so that both the dense generator and the spatial refiner could be more focused on their own sub-goals. In addition, the cascading scheme allows the two sub-networks to complement each other during network learning, thus leading to a substantial performance boost; see Figure 4.1(d). Extensive experimental results demonstrate that our method outperforms others on both real-scanned and synthetic inputs.

4.2 Method

4.2.1 Overview

Essentially, 3D scanning is a sampling problem in the 3D physical space, while upsampling is a prediction problem, aiming to infer more samples on the original surface, given the sparse

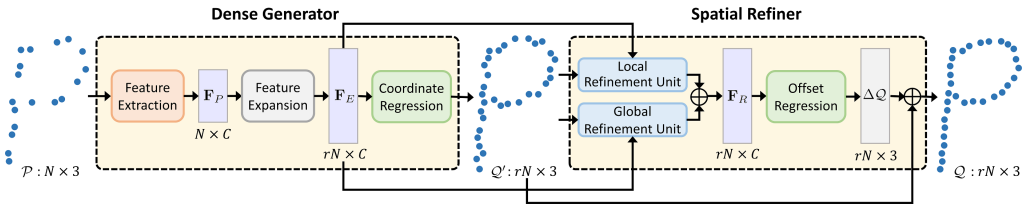


Figure 4.2: An illustration of our framework. Given sparse input $\mathcal{P}$ of N points, the dense generator extracts feature map $\mathbf{F}_P$ from the input, generates the expanded feature map $\mathbf{F}_E$, then produces a coarse but dense point set $\mathcal{Q}'$ of rN points, where r is the upsampling rate. Next, the spatial refiner consumes both $\mathcal{Q}'$ and the associated $\mathbf{F}_E$ to obtain the refined feature map $\mathbf{F}_R$ via a pair of local and global refinement units. We then regress offsets $\Delta\mathcal{Q}$ from $\mathbf{F}_R$, and output the final refined dense points $\mathcal{Q}$ by $\mathcal{Q}' + \Delta\mathcal{Q}$.

samples obtained in the scanning. Given a point set $\mathcal{P}$ of N points, which is typically sparse, non-uniform, and noisy, the point cloud sampling task aims to generate a dense point set, say $\mathcal{Q}$ of rN points, for a given upsampling rate r. These upsampled points should (i) faithfully describe the underlying object surface and (ii) cover the surface with a uniform distribution. This upsampling task is very challenging, since we need to infer new knowledge from the sparse input, in which the information about the original geometry is not completely represented.

Different from the existing approaches that try to meet the various goals in upsampling all in a single network, we propose to disentangle the upsampling task into two sub-goals, where we first generate a coarse but dense point set and then refine these points over the underlying surface to improve the distribution uniformity. Before giving the details of our method, we first discuss our key insights:

- First, we propose an end-to-end disentangled refinement framework with two cascaded sub-networks: one to generate dense points that roughly locate on the underlying surface and the other to aim for proximity-to-surface and distribution uniformity. Thus, each sub-network can better focus on its specific sub-goal, while complementing each other in the upsampling task.
- Second, with the help from the spatial refiner sub-network, the dense generator sub-network does not need a complicated structure. Hence, having a simple yet effective structure can enable it to expand features with higher flexibility and increase the upsampling rate without introducing extra network parameters.
- Third, we design the spatial refiner with both local and global refinements, such that we can leverage their complementary strengths to locally improve the distribution uniformity and proximity-to-surface, and globally explore similar structures on the surface.

Figure 4.2 shows the overall framework of our method. Given a sparse point set $\mathcal{P}$, we first feed it into our *dense generator* to generate the dense output $\mathcal{Q}'$ with rN points. At present, $\mathcal{Q}'$ may still be non-uniform and noisy like $\mathcal{P}$, as illustrated in the figure's toy example. Next, we feed it into our *spatial refiner* to further regress a per-point offset vector $\Delta\mathcal{Q}$, which is used to adjust the location of each point in $\mathcal{Q}'$, such that the refined

dense points $\mathcal{Q}$ can faithfully locate on the underlying surface, while being more uniform. In the following, we first present the details of the dense generator and spatial refiner in Sections 4.2.2 and 4.2.3, respectively. Then, we give the details of the patch-based end-to-end network training in Section 6.3.3.

4.2.2 Dense Generator

Given $\mathcal{P} \in \mathbb{R}^{N\times 3}$, our dense generator produces the upsampled coarse points $\mathcal{Q}' \in \mathbb{R}^{rN\times 3}$. Similar to existing upsampling approaches [54, 56, 101], we also expand the number of points in the feature space.

Specifically, as shown in the left-side of Figure 4.2, we first employ a feature extraction unit to embed the feature map $\mathbf{F}_P \in \mathbb{R}^{N\times C}$ from $\mathcal{P}$, where C is the number of feature channels. Here, we follow [56] to use the same feature extraction unit by considering the efficiency and effectiveness. Please refer to [56] for the details of this unit. Next, we feed $\mathbf{F}_P$ into a feature expansion unit to generate the expanded feature map $\mathbf{F}_E \in \mathbb{R}^{rN\times C}$. As discussed in Section 4.2.1, with the help from the cascaded spatial refiner, the dense generator only needs to generate a dense point set to roughly locate on the underlying surface. Hence, in the feature expansion unit, we adopt the commonly-used expansion operation by duplicating $\mathbf{F}_P$ with r copies and concatenating with a regular 2D grid to obtain $\mathbf{F}_E$. Although such operation may introduce redundant information

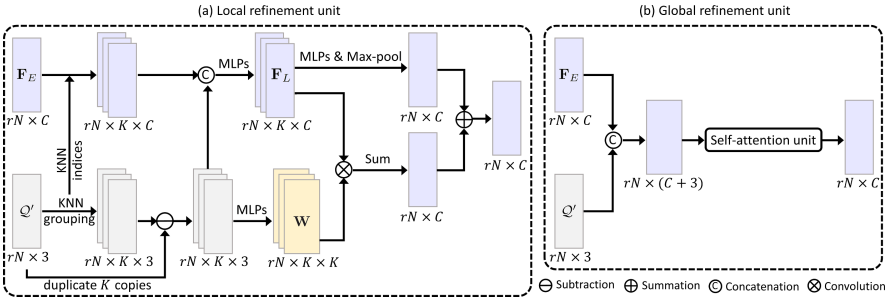


Figure 4.3: The architecture of (a) the local refinement unit and (b) the global refinement unit.

or extra noise, these problems could be rectified by the subsequent spatial refiner. Lastly, $\mathcal{Q}'$ is generated by regressing the point coordinates from $\mathbf{F}_E$ via multi-layer perceptrons (MLPs).

4.2.3 Spatial Refiner

Considering that $\mathcal{Q}'$ may still be noisy and non-uniform, we thus design a spatial refiner to further fine-tune the spatial location of each point in $\mathcal{Q}'$ and generate a high-quality dense point set $\mathcal{Q}$, which lies on the underlying surface and also distributes uniformly.

To do so, as shown in the right-side of Figure 4.2, we first feed the coarse $\mathcal{Q}'$ and the associated coarse feature map $\mathbf{F}_E$ into both local and global refinement units, which are detailed later. Next, we sum the two outputs generated by the two refinement units to obtain the refined feature map $\mathbf{F}_R \in \mathbb{R}^{rN \times C}$. Then, instead of directly regressing the refined point coordinates, we adopt residual learning to regress the per-point offset $\Delta\mathcal{Q}$. The reason behind is that, compared with the relative offset, the absolute

point coordinates are more diverse and have a wide distribution in 3D space. Hence, it is difficult for the network to synthesize points without introducing extra noise, while still preserving the uniformity and shape structures. Lastly, the final output $\mathcal{Q}$ is obtained by $\mathcal{Q}' + \Delta\mathcal{Q}$.

Local refinement unit aims to evolve $\mathbf{F}_E$ by considering the local geometric structures. Figure 4.3(a) shows the detailed architecture. Specifically, we first employ KNN grouping on $\mathcal{Q}'$ to search K-nearest neighbors and group the associated neighbor points together to obtain a stacked $rN \times K \times 3$ point volume. At the same time, we employ the same nearest neighbor indices to group $\mathbf{F}_E$ into an $rN \times K \times C$ feature volume. Then, we duplicate $\mathcal{Q}'$ with K copies and apply a subtraction operation on the duplicated and the grouped point volumes, which helps encode local information. We then concatenate the subtracted point volume with the grouped feature volume and apply MLPs to obtain the encoded local feature volume $\mathbf{F}_L \in \mathbb{R}^{rN \times K \times C}$.

Next, to obtain the local point feature over $\mathbf{F}_L$, a common routine is to apply MLPs followed by a max-pooling along the K-dimension. However, to account for the relative importance among the K neighbors, we further regress a spatial weight $\mathbf{W}$ (see the light yellow volume in Figure 4.3 (a)) from the subtracted point volume. Then, we modify $\mathbf{F}_L$ via a convolution with $\mathbf{W}$, followed by a summation along the K-dimension to obtain the weighted $rN \times C$ feature map. Lastly, we sum the

weighted feature map as the final refined local features.

Global refinement unit aims to refine $\mathbf{F}_E$ by considering the overall shape structure. As shown in Figure 4.3(b), instead of feeding only $\mathbf{F}_E$ to the refinement unit, we concatenate $\mathbf{F}_E$ and $\mathcal{Q}'$ together as the input to avoid losing the overall shape structure. Next, we adopt the widely-used self-attention unit [91] to obtain the refined global feature map, since this unit regresses attention weights among all the rN points, thus introducing long-range context dependencies. For brevity, we will not describe the details of this attention unit; please refer to [91] if needed.

4.2.4 Patch-based End-to-end Training

Since point cloud upsampling is a low-level task that requires us to focus more on the local geometric structures, we thus adopt the patch-based training strategy, as all the existing upsampling approaches did. During training, for each input sparse point set $\mathcal{P}$ and its associated target dense point set $\hat{\mathcal{Q}}$, our framework predicts both $\mathcal{Q}'$ and $\mathcal{Q}$. Hence, we formulate our objective function to encourage the geometric consistency between $\mathcal{Q}'\&\hat{\mathcal{Q}}$, and between $\mathcal{Q}\&\hat{\mathcal{Q}}$:

$$\mathcal{L} = \mathcal{L}_{\mathrm{CD}}(\mathcal{Q}', \hat{\mathcal{Q}}) + \lambda \mathcal{L}_{\mathrm{CD}}(\mathcal{Q}, \hat{\mathcal{Q}}), \tag{4.1}$$

where $\mathcal{L}_{\mathrm{CD}}(\cdot)$ means the Chamfer Distance (CD) [104] to measure the average closest point distance between two point sets. The parameter λ controls the relative importance of each term.

In the early stage of network training, we set a small λ, so that the network focuses more on the training of the dense generator to produce a more reliable $\mathcal{Q}'$. As the training progresses, we gradually increase λ to let our spatial refiner to be fully trained.

Note that, we also tried to combine Eq. (4.1) with the repulsion loss [54] to encourage the distribution uniformity. However, we found that this repulsion loss does not contribute too much in our work, because our method can already generate a relatively uniform dense point set benefited from the disentangled refinement scheme, even without any losses to constrain the uniformity distribution.

4.3 Experiments

4.3.1 Experimental Settings

Datasets. We employ both synthetic and real-scanned datasets in our experiments. For the synthetic dataset, we use the benchmark dataset provided by [101] with 120 training and 27 testing objects. For each training object, we follow [101] to crop 200 overlapped patches, thus resulting in totally 24,000 training surface patches. On each surface patch, we uniformly sample rN points as target $\hat{\mathcal{Q}}$, and then randomly downsample N points from $\hat{\mathcal{Q}}$ as the training input $\mathcal{P}$. For each testing shape, we follow MPU [56] and PU-GAN [101] to sample $\sim$20,000 uniform points using Poisson disk sampling as $\hat{\mathcal{Q}}$ for quantitative evalu-

ation, and generate 1,024 non-uniform points for testing.

For real-scanned dataset, we use ScanObjectNN [100], which contains 2,902 point cloud objects in 15 categories. Each object has 2,048 points. Since no target dense points are provided in ScanObjectNN, we just use this dataset for testing. Hence, during testing, we have both synthetic and real-scanned point clouds. For each testing point cloud with 2,048 points, we use farthest sampling to pick seeds and extract a local patch of N points per seed. We then feed these patches to the network for testing and combine the upsampled results as the final output.

Evaluation metrics. For quantitative evaluation, we consider three widely-used evaluation metrics: (i) Chamfer distance (CD), (ii) Hausdorff distance (HD), and (iii) Point-to-surface (P2F) distance using the original testing objects. A lower evaluation metric indicates a better performance.

Comparison methods. To demonstrate the effectiveness of our method, we compare it with three state-of-the-art point cloud upsampling methods, including PU-Net [54], MPU [56], and PU-GAN [101]. We use their released public code and follow the same setting in the original papers to re-train their networks using our prepared training data. Note that, for the recent work PUGeoNet [57], we cannot provide the comparison results without available code so far.

Implementation details. In experiments, we set $N = 256$. We train our network with a batch size of 28 for 400 epochs

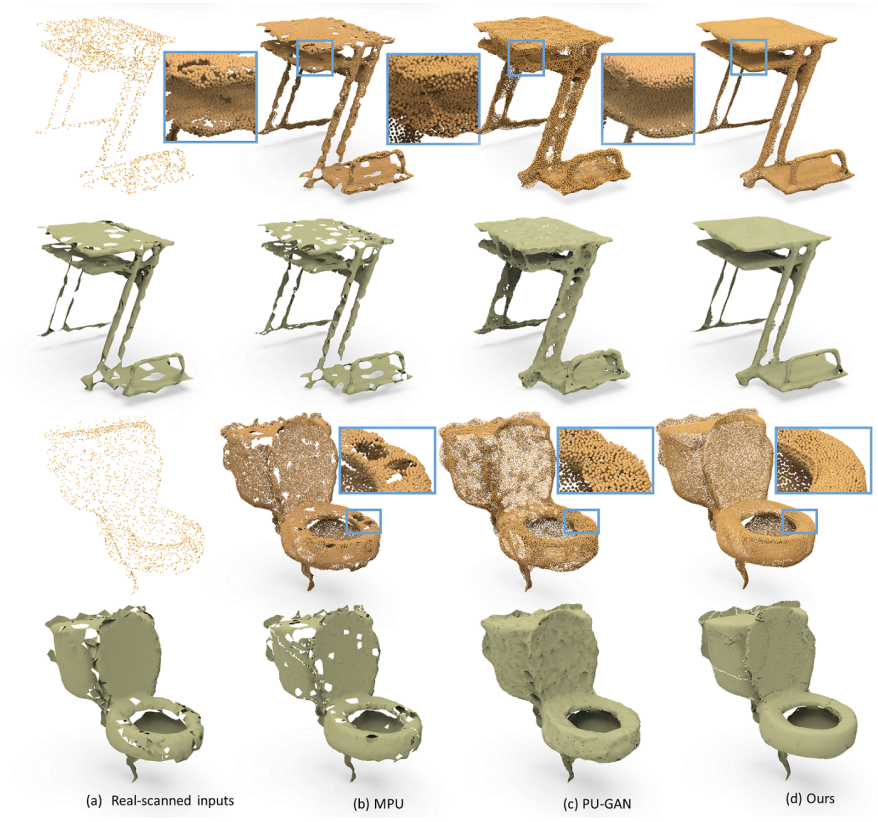


Figure 4.4: Comparing point set upsampling (16×) results and reconstructed 3D meshes using different methods (b-d) from real-scanned sparse inputs (a), while the bottom row shows the reconstructed meshes.

on the TensorFlow platform. For each patch, we apply random scaling, rotation, and point perturbation to avoid over-fitting. The Adam optimizer is used with the learning rate of 0.001, which is linearly decreased by a decay rate of 0.7 per 40 epochs until 10^{-6}. The parameter λ in Eq. (4.1) is linearly increased from 0.01 to 1.0 as the training progresses.

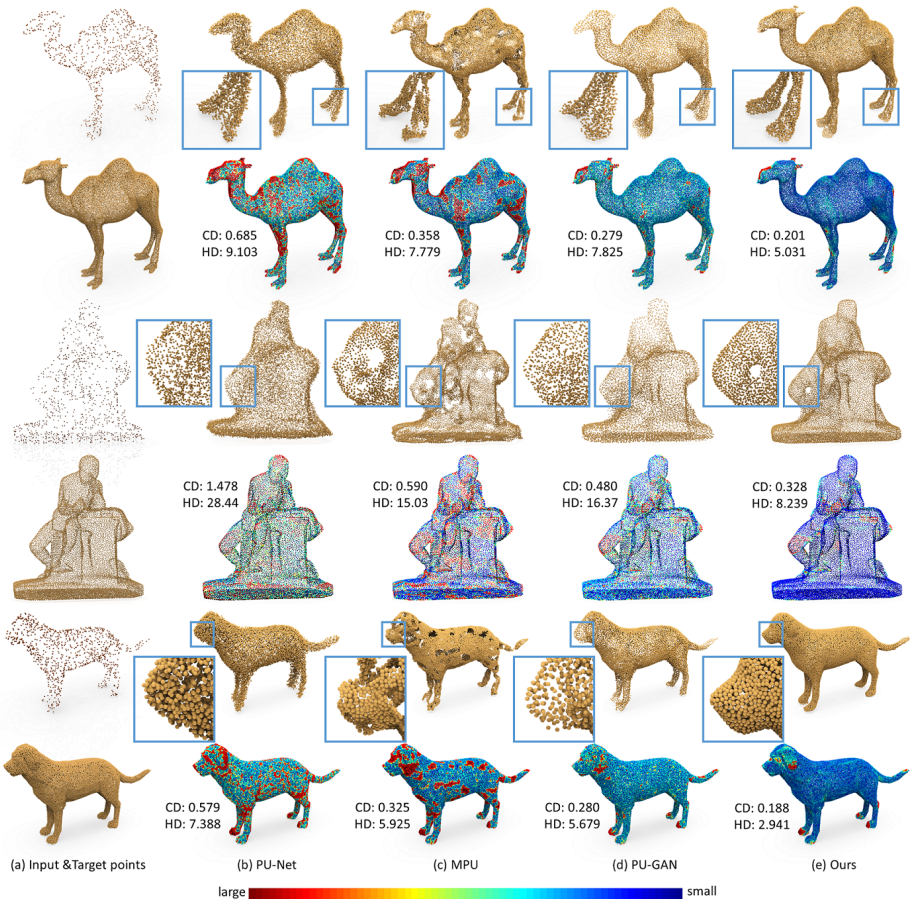


Figure 4.5: Comparing point set upsampling (x16) results from synthetic sparse inputs (a) using different methods (b-e). We also show the associated error maps, where the colors reveal the nearest distance for each target point to the predicted point set generated by each method.

4.3.2 Results on Real-scanned Dataset

First, we compared our method with state-of-the-arts on real-scanned test inputs. Besides the results shown earlier in Figure 4.1, we further show more visual comparison results in Figure 4.4 (see page 8), where we set $r = 16$. For each object, the top row shows the upsampled points by each method, and the bottom row shows the associated reconstructed 3D meshes using

Table 4.1: Quantitative comparisons by using our method and state-of-the-arts. The units of CD, HD, and P2F are all 10^{-3}.

Methods	4X				16X			
	Size	CD	HD	P2F	Size	CD	HD	P2F
PU-Net [54]	10.1M	0.844	7.061	9.431	24.5M	0.699	8.594	11.619
MPU [56]	23.1M	0.632	6.998	6.199	92.5M	0.348	7.187	6.822
PU-GAN [101]	9.57M	0.483	5.323	5.053	9.57M	0.269	7.127	6.306
Our	13.2M	**0.315**	**4.201**	**4.149**	13.2M	**0.199**	**4.716**	**4.249**

ball-pivoting surface reconstruction algorithm [102]. Note that, since PU-Net is an early work with not very promising results, we thus omit its results in visual comparisons. As shown in the top row of Figure 4.4(a), to upsample the real-scanned sparse inputs is very challenging, since these points are not only noisy and non-uniform, but also exhibit many small holes and structural defects. Thus, reconstructing meshes directly from sparse inputs often results in incomplete surfaces with many holes; see the bottom results in (a). Comparing the upsampled points produced by various methods, the other methods tend to retain noise in their results, or fail to generate a uniform output, thus resulting in low-quality reconstructed meshes with small holes or rough surfaces. On the contrary, our method enables to produce uniform dense points with low deviations to the underlying object surface. Hence, the reconstructed meshes from our upsampled points can well describe the geometric structures with smooth and complete surfaces.

4.3.3 Results on Synthetic Dataset

Next, we compared our method with state-of-the-arts on synthetic test models provided by [101]. Figure 4.5 shows the visual comparisons on three sparse inputs, where we set r=16. Comparing the dense points produced by our method (e) and others (b-d) with the target (a), we can see that other methods tend to introduce excessive noise (*e.g.*, (b)), cluster points together with a non-uniform distribution (*e.g.*, (c)), or destroy some tiny structures (*e.g.*, (d)) in the results. In contrast, our method produces the most similar visual results to the target points, and our dense points can well preserve tiny local structures with a uniform point distribution; see particularly the blown-up view in Figure 4.5. Besides, we show also the associated error maps, where the colors reveal the nearest distance for each point in target point set to the predicted point set. We can see that the errors of our upsampled results are the lowest (*i.e.*, most points are blue), which is also verified by both CD and HD values.

Table 6.1 shows the quantitative comparisons on all the synthetic test models under different upsampling rates. We can see that our method achieves the lowest values on all the evaluation metrics in terms of both upsampling rates. Note that, different from PU-Net and MPU, the number of learnable parameters in our network will not increase as r increases. Hence, our method has good scalability to a large upsampling rate. More importantly, when r increases, the advantages of our method com-

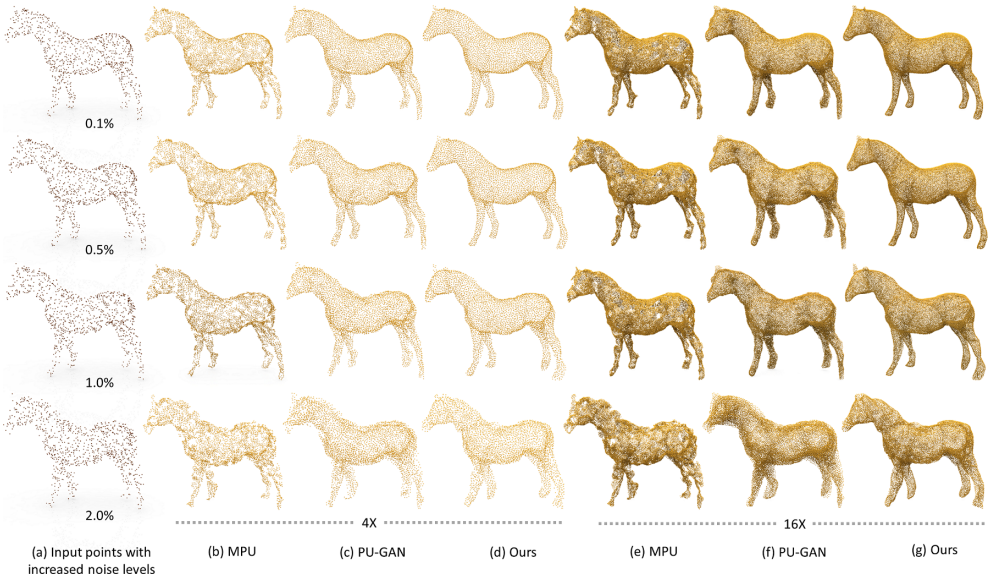


Figure 4.6: Comparing point set upsampling results produced using different methods under different upsampling rate, when given noisy sparse inputs with increasing noise level, *i.e.*, 0.1%, 0.5%, 1.0%, and 2.0%.

pared with others become more obvious. The reason behind is that, the prediction difficulty will significantly increase given a large r for existing approaches. However, thanks to the disentangled refinement scheme, our method has better adaptability.

4.3.4 Noise Robustness Test

We explored the noise robustness of our method by adding Gaussian noise of different levels to the synthetic test inputs. Figure 4.6 shows the visual comparisons. Clearly, our method achieves more uniform upsampling results (d & g) without excessive noise, under both upsampling rates. The quantitative comparisons are summarized in Table 4.2, where $r = 16$ and we show the CD values. Obviously, our method produces the low-

Table 4.2: Quantitative comparisons by using our method and state-of-the-arts to upsample noisy inputs with increasing noise level ($r = 16$). Here we show CD values with the unit of 10^{-3}.

Methods	Perturbation with different noise levels				
	0%	0.1%	0.5%	1.0%	2.0%
PU-Net [54]	0.699	0.717	0.794	0.860	0.945
MPU [56]	0.348	0.364	0.426	0.524	0.831
PU-GAN [101]	0.269	0.309	0.381	0.562	0.899
Our	**0.199**	**0.213**	**0.229**	**0.310**	**0.592**

Table 4.3: Comparing the upsampling performance of our full pipeline with various cases in the ablation study (r=16). Here we show CD values with the unit of 10^{-3}.

Model	Spatial Refiner	Local	Global	Offset	CD
A					0.684
B	✓		✓	✓	0.378
C	✓	✓		✓	0.343
D	✓	✓	✓		0.228
Full					**0.199**

est values across all the noise levels with a significant margin, compared to others.

4.3.5 Ablation Study

To evaluate the effectiveness of the major components in our framework, we conducted an ablation study by simplifying our full pipeline in the following four cases: (A) removing the spa-

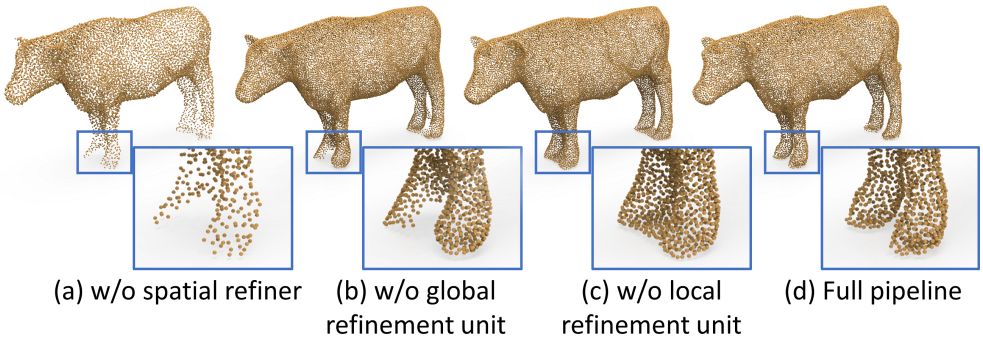


Figure 4.7: Visualization results for the ablation study.

tial refiner and only keeping the dense generator (see Figure 4.2); (B) removing the local refinement unit; (C) removing the global refinement unit; and (D) removing offsets and directly regressing the point coordinates. In each case, we re-trained the network and tested the performance using synthetic data. Table 4.3 summarizes the results of each case in terms of CD value, compared to our full pipeline (bottom row). Clearly, our full pipeline performs the best with the lowest CD value, and removing any component reduces the overall performance, meaning that each component in our framework contributes. We also present a visual result associated with the ablation study in Figure 4.7. Noted that since the dense generator adopts an identical or simplified design from the previous works [56, 101], our spatial refiner is generally applicable to other networks.

4.4 Summary

In this paper, we present a disentangled refinement framework for point cloud upsampling. Different from existing approaches

that try to meet the various upsampling goals all in a single network, we propose to disentangle the upsampling task into two sub-goals, where we first generate coarse but dense points, and then refine these points by adjusting the location of each point. To this end, we formulate an end-to-end disentangled refinement framework with two cascaded sub-networks: a dense generator and a spatial refiner. In the spatial refiner, we introduce a pair of local and global refinement units to evolve the coarse feature map by considering both local and global geometric structures. Also, we design our spatial refiner to regress offset vectors to adjust the coarse outputs in fine scale. Experimental results demonstrate the superiority of our method over others.

Actually, this work aims to provide a generic framework to disentangle the point cloud upsampling task. In the future, we may continue to explore a more comprehensive architecture for dense generator and spatial refiner. We may further explore the possibility of designing a region adaptive refiner, meaning that we only fine-tune regions that are non-uniform and noisy, thus improving the overall efficiency. Lastly, designing the refiner to be aware of edges may be helpful for downstream tasks like mesh reconstruction.

□ **End of chapter.**

Chapter 5

PointAugment: an Auto-Augmentation Framework for Point Cloud Classification

5.1 Introduction

In recent years, there has been a growing interest in developing deep neural networks [33, 5, 6, 105, 10] for 3D point cloud processing. To robustly train a network often relies on the availability and diversity of the data. However, unlike 2D image benchmarks such as ImageNet [14] and MS COCO dataset [15], which have over millions of training samples, 3D datasets are typically much smaller in quantity, with relatively small amount of labels and limited diversity. For instance, ModelNet40 [18], one of the most commonly-used benchmark for 3D point cloud classification, only has 12,311 models of 40 categories. The limited data quantity and diversity may cause overfitting problem

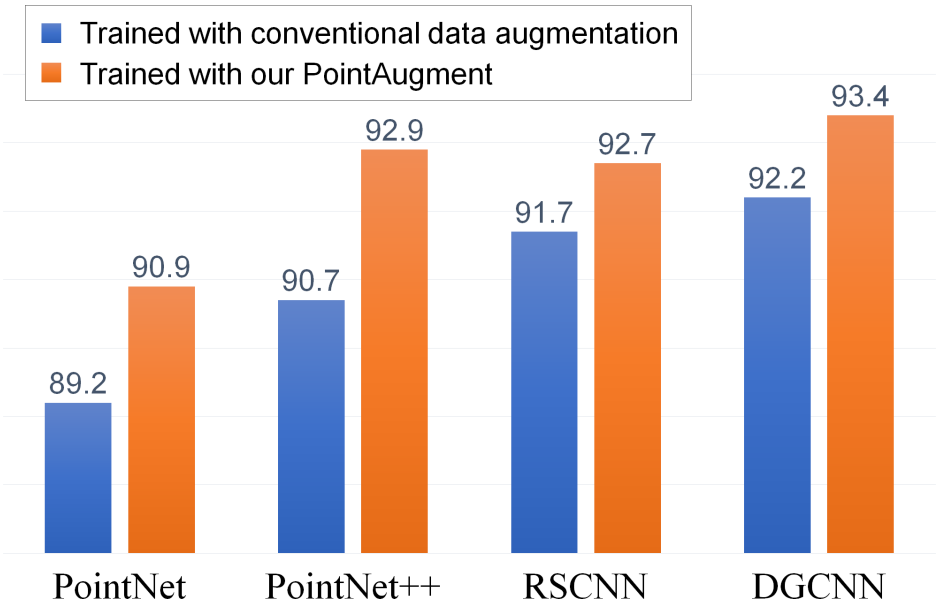


Figure 5.1: Classification accuracy (%) on ModelNet40 with or without training the networks with our PointAugment. We can see clear improvements on four representative networks.

and further affect the generalization ability of the network.

Nowadays, data augmentation (DA) is a very common strategy to avoid overfitting and improve the network generalization ability by artificially enlarging the quantity and diversity of the training samples. For 3D point clouds, due to the limited amount of training samples and an immense augmentation space in 3D, conventional DA strategies [33, 5] often simply perturb the input point cloud randomly in a small and fixed pre-defined augmentation range to maintain the class label. Despite its effectiveness for the existing classification networks, this conventional DA approach may lead to insufficient training.

First, existing methods for deep 3D point cloud processing regard the network training and DA as two independent phases

without *jointly optimizing* them, *e.g.*, feedback the training results to enhance the DA. Hence, the trained network could be suboptimal. Second, existing methods apply the *same fixed augmentation* process with rotation, scaling, and/or jittering, to all input point cloud samples. The shape complexity of the samples is ignored in the augmentation, *e.g.*, a sphere remains the same no matter how we rotate it, but a complex shape may need larger rotations. Hence, conventional DA may be redundant or insufficient for augmenting the training samples [106].

To improve the augmentation of point cloud samples, we formulate *PointAugment*, a new auto-augmentation framework for 3D point clouds, and demonstrate its effectiveness to enhance shape classification; see Figure 5.1. Different from the previous works for 2D images, PointAugment *learns to produce augmentation functions specific to individual samples*. Further, the learnable augmentation function considers both *shape-wise transformation* and *point-wise displacement*, which relate to the characteristics of 3D point cloud samples. Also, PointAugment *jointly optimizes the augmentation process with the network training*, via an adversarial learning strategy to train the augmentation network (augmentor) together with the classification network (classifier) in an end-to-end manner. By taking the classifier losses as *feedbacks*, the augmentor can learn to enrich the input samples by enlarging the intra-class data variations, while the classifier can learn to combat this by extracting insensitive features. Benefited by such adversarial learning, the augmentor

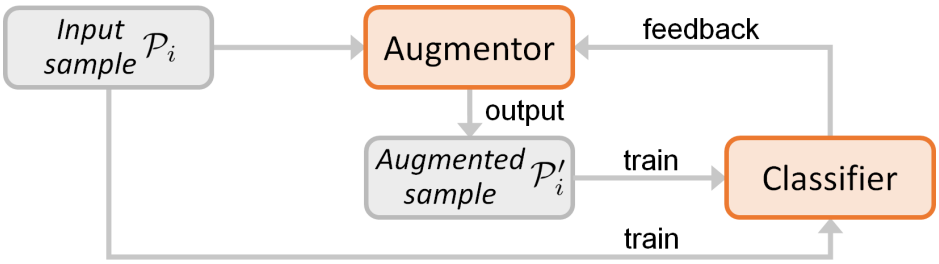


Figure 5.2: An overview of our PointAugment framework. We jointly optimize the augmentor and classifier in an end-to-end manner with an adversarial learning strategy.

can then learn to generate augmented samples that best fit the classifier in different stages of the training, thus maximizing the capability of the classifier.

As the first attempt to explore auto-augmentation for 3D point clouds, we show by replacing conventional DA with PointAugment, clear improvements in shape classification on ModelNet40 [18] (see Figure 5.1) and SHREC16 [107] (see Section 5.4) datasets can be achieved on four representative networks, including PointNet [33], PointNet++ [5], RSCNN [10], and DGCNN [6]. Also, we demonstrate the effectiveness of PointAugment on shape retrieval and evaluate its robustness, loss configuration, and modularization design. More results are presented in Section 5.4.

5.2 Overview

The main contribution of this work is the PointAugment framework that automatically optimizes the augmentation of the input point cloud samples for more effectively training the classification network. Figure 5.2 illustrates the design of our frame-

work, which has two deep neural network components: (i) an augmentor $\mathcal{A}$ and (ii) a classifier $\mathcal{C}$. Given an input training dataset $\{\mathcal{P}_i\}_{i=1}^{M}$ of M samples, where each sample has N points, before we train classifier $\mathcal{C}$ with sample $\mathcal{P}_i$, we feed $\mathcal{P}_i$ first to our augmentor $\mathcal{A}$ to generate an augmented sample $\mathcal{P}'_i$. Then, we feed $\mathcal{P}_i$ and $\mathcal{P}'_i$ to classifier $\mathcal{C}$ for training, and further take $\mathcal{C}$'s results as feedback to guide the training of augmentor $\mathcal{A}$.

Before elaborating the PointAugment framework, we first discuss our key ideas behind the framework. These are new ideas (not present in previous works [68, 69, 70]) that enable us to efficiently augment the training samples, which are now 3D point clouds instead of 2D images.

- *Sample-aware.* Rather than finding a universal set of augmentation policy or procedure for processing every input data sample, we aim to regress a specific augmentation function for each input sample by considering the underlying geometric structure of the sample. We call this a sample-aware auto-augmentation.
- *2D vs. 3D augmentation.* Unlike 2D augmentations for images, 3D augmentation involves a more immense and different spatial domain. Accounting for the nature of 3D point clouds, we consider two kinds of transformations on point cloud samples: shape-wise transformation (including rotation, scaling, and their combinations), and point-wise displacement (jittering of point locations), where our aug-

mentor should learn to produce them to enhance the network training.

- *Joint optimization.* During the network training, the classifier will gradually learn and become more powerful, so we need more challenging augmented samples to better train the classifier, as the classifier becomes stronger. Hence, we design and train the PointAugment framework in an end-to-end manner, such that we can jointly optimize both the augmentor and classifier. To achieve so, we have to carefully design the loss functions and dynamically adjust the difficulty of the augmented samples, while considering both the input sample and the capacity of the classifier.

5.3 Method

In this section, we first present the network architecture details of the augmentor and classifier (Section 5.3.1). Then, we present our loss functions formulated for the augmentor (Section 5.3.2) and classifier (Section 5.3.3), and introduce our end-to-end training strategy (Section 5.3.4). Lastly, we present the implementation details (Section 5.3.5).

5.3.1 Network Architecture

Augmentor. Different from existing works [68, 69, 70], our augmentor is sample-aware, and it learns to generate a specific

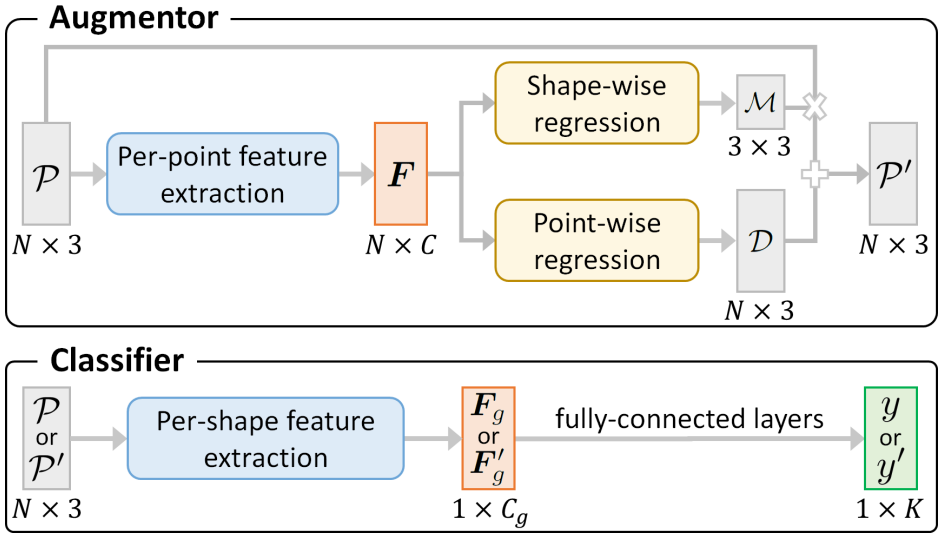


Figure 5.3: Illustrations of the augmentor and classifier. The augmentor generates augmented sample $\mathcal{P}'$ from $\mathcal{P}$, and the classifier predicts the class label given $\mathcal{P}'$ or $\mathcal{P}$ as inputs.

function for augmenting each input sample. From now on, we drop subscript i for ease of reading, and denote $\mathcal{P}$ as the training sample input to augmentor $\mathcal{A}$ and $\mathcal{P}'$ as the corresponding augmented sample output from $\mathcal{A}$.

The overall architecture of our augmentor is illustrated in Figure 5.3 (top). First, we use a per-point feature extraction unit to embed point features $\boldsymbol{F} \in \mathbb{R}^{N \times C}$ for all N points in $\mathcal{P}$, where C is the number of feature channels. From $\boldsymbol{F}$, we then regress the augmentation function specific to input sample $\mathcal{P}$ using two separate components in the architecture: (i) shape-wise regression to produce transformation $\mathcal{M} \in \mathbb{R}^{3 \times 3}$ and (ii) point-wise regression to produce displacement $\mathcal{D} \in \mathbb{R}^{N \times 3}$. Note that, the learned $\mathcal{M}$ is a linear matrix in 3D space, combining

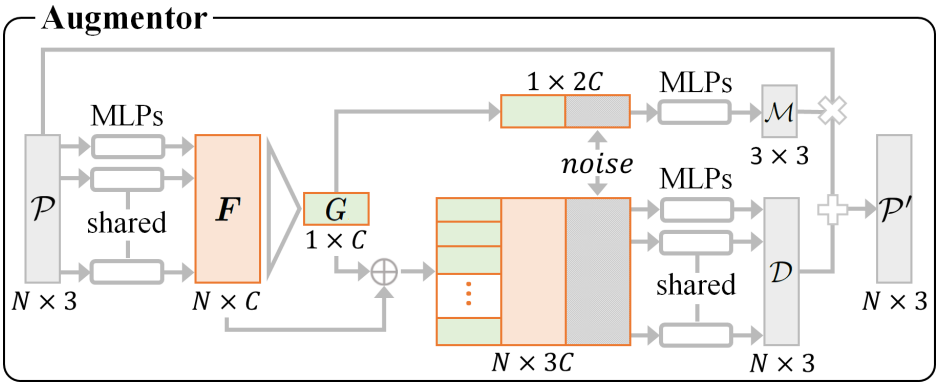


Figure 5.4: Our implementation of the augmentor.

mainly rotation and scaling, whereas the learned $\mathcal{D}$ gives point-wise translation and jittering. Using $\mathcal{M}$ and $\mathcal{D}$, we can then generate the augmented sample $\mathcal{P}'$ as $\mathcal{P} \cdot \mathcal{M} + \mathcal{D}$.

The design of our proposed framework for the augmentor is generic, meaning that we may use different models to build its components. Figure 5.4 shows our current implementation, for reference. Specifically, similar to PointNet [33], we first employ a series of shared multi-layer perceptron (MLPs) to extract per-point features $\boldsymbol{F} \in \mathbb{R}^{N \times C}$. We then employ max pooling to obtain the per-shape feature vector $\boldsymbol{G} \in \mathbb{R}^{1 \times C}$. To regress $\mathcal{M}$, we generate a C-dimension noise vector based on a Gaussian distribution and concatenate it with $\boldsymbol{G}$, and then employ MLPs to obtain $\mathcal{M}$. Note that the noise vector enables the augmentor to explore more diverse choices in regressing the transformation matrix through the randomness introduced into the regression process. To regress $\mathcal{D}$, we concatenate N copies of $\boldsymbol{G}$ with $\boldsymbol{F}$, together with an $N \times C$ noise matrix, whose values are randomly and independently generated based on a Gaussian distribution.

Lastly, we employ MLPs to obtain $\mathcal{D}$.

Classifier. Figure 5.3 (bottom) shows the general architecture of classifier $\mathcal{C}$. It takes $\mathcal{P}$ and $\mathcal{P}'$ as inputs in two separate rounds and predicts corresponding class labels y and y'. Both y and $y' \in \mathbb{R}^{1\times K}$, where K is the total number of classes in the classification problem. In general, $\mathcal{C}$ first extracts per-shape global features $\boldsymbol{F}_g$ or $\boldsymbol{F}'_g \in \mathbb{R}^{1\times C_g}$ (from $\mathcal{P}$ or $\mathcal{P}'$), and then employ fully-connected layers to regress a class label. Also, the choice of implementing $\mathcal{C}$ is flexible. We may employ different classification networks as $\mathcal{C}$. In Section 5.4, we shall show that the performance of several conventional classification networks can be further boosted when equipped with our augmentor in the training.

5.3.2 Augmentor loss

To maximize the network learning, augmented sample $\mathcal{P}'$ generated by the augmentor should satisfy two requirements: (i) $\mathcal{P}'$ should be more challenging than $\mathcal{P}$, *i.e.*, we aim for $L(\mathcal{P}') \geq L(\mathcal{P})$; and (ii) $\mathcal{P}'$ should not lose its shape distinctiveness, meaning that it should describe a shape that is not too far (or different) from $\mathcal{P}$.

To achieve requirement (i), a simple way to formulate the loss function for the augmentor (denoted as $\mathcal{L}_{\mathcal{A}}$) is to maximize the difference between the cross entropy losses on $\mathcal{P}$ and $\mathcal{P}'$, or

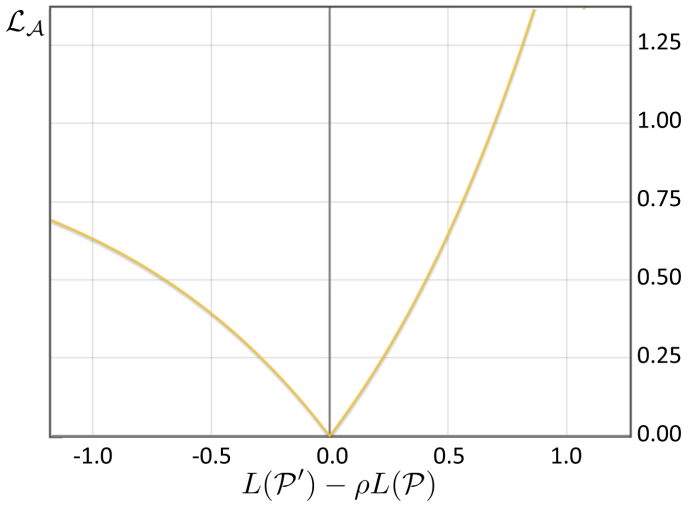


Figure 5.5: Graph plot of Eq. (5.2).

equivalently, to minimize

$$\mathcal{L}_{\mathcal{A}} = \exp[-(L(\mathcal{P}') - L(\mathcal{P}))], \tag{5.1}$$

where $L(\mathcal{P})=-\sum_{c=1}^{K} \hat{y}_c \log(y_c))$ is $\mathcal{P}$'s cross entropy loss; $\hat{y}_c \in \{0,1\}$ denotes the one-hot ground-truth label when $\mathcal{P}$ belongs to the c-th class; and $y_c \in [0,1]$ is the probability of predicting $\mathcal{P}$ as c-th class. Note also that, for $\mathcal{P}'$ to be more challenging than $\mathcal{P}$, we assume that $L(\mathcal{P}')\geq L(\mathcal{P})$ and a larger $L(\mathcal{P}')$ indicates a larger magnitude of augmentation, which can be defined as $\xi = L(\mathcal{P}')-L(\mathcal{P})$.

However, if we naively minimize Eq. (5.1) for $\mathcal{L}_{\mathcal{A}}\rightarrow 0$, we encourage $L(\mathcal{P}')-L(\mathcal{P})\rightarrow\infty$. So, a simple solution for $\mathcal{P}'$ is an arbitrary sample regardless of $\mathcal{P}$. Such $\mathcal{P}'$ clearly violates requirement (ii). Hence, we further restrict the augmentation magnitude ξ. Inspired by LS-GAN [108], we first introduce a

dynamic parameter ρ and re-formulate $\mathcal{L}_{\mathcal{A}}$ as

$$\mathcal{L}_{\mathcal{A}} = |1.0 - \exp[L(\mathcal{P}') - \rho L(\mathcal{P})]|. \tag{5.2}$$

See Figure 5.5 for the graph plot of Eq. (5.2). In this formulation, we want $L(\mathcal{P}')$ to be large (for requirement (i)) but it should not be too large (for requirement (ii)), so we upper-bound $L(\mathcal{P}')$ by $\rho L(\mathcal{P})$. Hence, we can obtain

$$\xi \;=\; L(\mathcal{P}') - L(\mathcal{P}) \;\leq\; (\rho - 1)L(\mathcal{P}), \tag{5.3}$$

where we denote $\xi_o = (\rho - 1)L(\mathcal{P})$ as ξ's upper bound.

Note that, when we train the augmentor, the classifier is fixed (to be presented in Section 5.3.4), so $L(\mathcal{P})$ is fixed. Hence, ξ_o depends only on ρ. Since it should be non-negative, we thus ensure $\rho \geq 1$. Moreover, considering that the classifier is very fragile at the beginning of the training, we pay more attention to training the classifier rather than generating a challenging $\mathcal{P}'$. Hence, ξ_o should not be too large, meaning that $\mathcal{P}'$ should not be too challenging. Later, when the classifier becomes more powerful, we can gradually enlarge ξ_o to allow the augmentor to generate a more challenging $\mathcal{P}'$. Therefore, we design a dynamic ρ to control ξ_o with the following formulation:

$$\rho = \max\Big(1, \exp(\sum_{c=1}^{K} \hat{y}_c \cdot y_c)\Big), \tag{5.4}$$

where $\max(1, *)$ ensures $\rho \geq 1$. At the beginning of the network training, the classifier predictions may not be accurate. Hence, the prediction probability y_c is generally small, resulting in a

small ρ, and ξ_o will also be small according to Eq. (5.3). When the classifier becomes more powerful, y_c will increase, and we will have larger ρ and ξ_o accordingly.

Lastly, to further ensure the augmented sample $\mathcal{P}'$ to be shape distinctive (for requirement (ii)), we add $L(\mathcal{P}')$, as a fidelity term, to Eq. (5.2) to construct the final loss $\mathcal{L}_{\mathcal{A}}$:

$$\mathcal{L}_{\mathcal{A}} = L(\mathcal{P}') + \lambda|1.0 - \exp(L(\mathcal{P}') - \rho L(\mathcal{P}))|, \tag{5.5}$$

where λ is a fixed hyper-parameter to control the relative importance of each term. A small λ encourages the augmentor to focus more on the classification with less augmentation on $\mathcal{P}$, and vice versa. In our implementation (all experiments), we set $\lambda = 1$ to treat the two terms equally.

5.3.3 Classifier loss

The goal of the classifier $\mathcal{C}$ is to correctly predict both $\mathcal{P}$ and $\mathcal{P}'$. Additionally, $\mathcal{C}$ should also have the ability to learn stable per-shape global features, no matter given $\mathcal{P}$ or $\mathcal{P}'$ as input. We thus formulate the classifier loss $\mathcal{L}_{\mathcal{C}}$ as

$$\mathcal{L}_{\mathcal{C}} = L(\mathcal{P}') + L(\mathcal{P}) + \gamma\|\boldsymbol{F}_g - \boldsymbol{F}_{g'}\|_2, \tag{5.6}$$

where γ is to balance the importance of the terms (we empirically set γ as 10.0), and $\|\boldsymbol{F}_g - \boldsymbol{F}_{g'}\|_2$ helps explicitly penalize the feature difference between the augmented sample and the original one, and stabilize the network training.

5.3.4 End-to-end training strategy

Algorithm 1 summarizes our end-to-end training strategy. Overall, the procedure alternatively optimizes and updates the learnable parameters in augmentor $\mathcal{A}$ and classifier $\mathcal{C}$, while fixing the other one, during the training. Given input sample $\mathcal{P}_i$, we first employ $\mathcal{A}$ to generate its augmented sample $\mathcal{P}'_i$. We then update the learnable parameters in $\mathcal{A}$ by calculating the augmentor loss using Eq. (5.5). In this step, we keep $\mathcal{C}$ unchanged. After updating $\mathcal{A}$, we keep $\mathcal{A}$ unchanged, and generate the updated $\mathcal{P}'_i$. We then feed $\mathcal{P}_i$ and $\mathcal{P}'_i$ to $\mathcal{C}$ one by one to obtain $L(\mathcal{P})$ and $L(\mathcal{P}')$, respectively, and update the learnable parameters in $\mathcal{C}$ by calculating the classifier loss using Eq. (5.6). In this way, we can optimize and train $\mathcal{A}$ and $\mathcal{C}$ in an end-to-end manner.

5.3.5 Implementation details

We implement PointAugment using PyTorch [109]. In detail, we set the number of training epochs $S = 250$ with a batch size $B = 24$. To train the augmentor, we adopt the Adam optimizer with a learning rate of 0.001. To train the classifier, we follow the respective original configuration from the released code and paper. Specifically, for PointNet [33], PointNet++ [5], and RSCNN [10], we use the Adam optimizer with an initial learning rate of 0.001, which is gradually reduced with a decay rate of 0.5 every 20 epochs. For DGCNN [6], we use the SGD solver with a momentum of 0.9 and a base learning rate of 0.1,

which decays using a cosine annealing strategy [110].

Note also that, to reduce model oscillation [111], we follow [62] to train PointAugment by using mixed training samples, which contain the original training samples as one half and our previously-augmented samples as the other half, rather than using only the original training samples. Please refer to [62] for more details. Moreover, to avoid overfitting, we set a dropout probability of 0.5 to randomly drop or keep the regressed shape-wise transformation and point-wise displacement. In the testing phase, we follow previous networks [33, 5] to feed the input test samples to the trained classifier to obtain the predicted labels, without any additional computational cost.

5.4 Experiments

We conducted extensive experiments on PointAugment. First, we introduce the benchmark datasets and classifiers employed in our experiments (Section 5.4.1). We then evaluate PointAugment on shape classification and shape retrieval (Section 5.4.2). Next, we perform detailed analysis on PointAugment's robustness, loss configuration, and modularization design (Section 6.5.3). Lastly, we present further discussion and potential future extensions (Section 5.4.4).

Algorithm 1: Training Strategy in PointAugment

Input: training point sets $\{\mathcal{P}_i\}_{i=1}^{M}$, corresponding ground-truth class labels $\{\hat{y}_i\}_{i=1}^{M}$, and the number of training epochs S.

Output: $\mathcal{C}$ and $\mathcal{A}$.

for $s = 1, \cdots, S$ **do**

  for $i = 1, \cdots, M$ **do**

    // Update augmentor $\mathcal{A}$

    Generate augmented sample $\mathcal{P}_i'$ from $\mathcal{P}_i$

    Calculate the augmentor loss using Eq. (5.5)

    Update the learnable parameters in $\mathcal{A}$

    // Update classifier $\mathcal{C}$

    Calculate the classifier loss using Eq. (5.6) by feeding $\mathcal{P}_i$ and $\mathcal{P}_i'$ alternatively to $\mathcal{C}$

    Update the learnable parameters in $\mathcal{C}$

  end

end

5.4.1 Datasets and Classifiers

Datasets. We employed three 3D benchmark datasets in our evaluations, *i.e.*, ModelNet10 [18], ModelNet40 [18], and SHREC16 [107], for which we denote as MN10, MN40, and SR16, respectively. Table 5.1 presents statistics about the datasets, showing that, MN10 is a very small dataset with only 10 classes. Though most networks [33, 112] can achieve a high classification accuracy on MN10, they may easily overfit. SR16 is the largest data with over 36,000 training samples. However, the high standard deviation (std.) value, *i.e.*, 1111, shows the uneven distribution of training samples among the classes. For example, in

Table 5.1: Statistics of the ModelNet10 (MN10) [18], ModelNet40 (MN40) [18], and SHREC16 (SR16) [107] datasets, including the number of categories (classes), number of training and testing samples, average number of samples per class, and the corresponding standard deviation value.

Dataset	#Class	#Training	#Testing	Average	Std.
MN10	10	3991	908	399.10	233.36
MN40	40	9843	2468	246.07	188.64
SR16	55	36148	5165	657.22	1111.49

SR16, the *Table* class has 5,905 training samples, while the *Cap* class has only 39 training samples. For MN40, we directly adopt the data kindly provided by PointNet [33] and follow the same train-test split. For MN10 and SR16, we uniformly sample 1,024 points on each mesh surface and normalize the point sets to fit a unit ball centered at the origin.

Classifiers. As explained in Section 5.3.1, our overall framework is generic, and we can employ different classification networks as classifier $\mathcal{C}$. To show that the performance of conventional classification networks can be further boosted when equipped with our augmentor, in the following experiments, we employ several representative classification networks as classifier $\mathcal{C}$, including (i) PointNet [33], a pioneer network that processes points individually; (ii) PointNet++ [5], a hierarchical feature extraction network; (iii) RSCNN[1] [10], a recently-released enhanced version of PointNet++ with a relation weight inside each

[1]Only the single-scale RSCNN [10] is released so far.

local region; and (iv) DGCNN [6], a graph-based feature extraction network. Note that, most existing networks [113, 13, 112] are built and extended from the above networks with various means of adaptation.

5.4.2 PointAugment Evaluation

Shape classification. First, we evaluate our PointAugment on the shape classification task using the classifiers listed in Section 5.4.1. For comparison, when we train the classifiers without PointAugment, we follow [5] to augment the training samples by random rotation, scaling, and jittering, which are considered as conventional DA.

Table 5.2 summarizes the quantitative evaluation results for comparison. We report the overall classification accuracy (%) of each classifier on all the three benchmark datasets, with conventional DA and with our PointAugment. From the results we can clearly see that, by employing PointAugment, the shape classification accuracies of all classifier networks can improve for all the three benchmark datasets. Particularly, on MN40, the classification accuracy achieved by DGCNN+PointAugment is 93.4%, which is a very high accuracy value comparable with the very recent works [113, 13, 112]. Moreover, our PointAugment is shown to be more effective on the imbalanced SR16 dataset; see the right-most column in Table 5.2, showing that PointAugment can alleviate the class size imbalance problem

Table 5.2: Comparing the overall shape classification accuracy (%) on MN40, MN10, and SR16, for various classifiers equipped with conventional DA (first four rows) and with our PA (last four rows); PA denotes PointAugment. We can observe improvements for all datasets and all classifiers.

Method	MN40	MN10	SR16
PointNet [33]	89.2	91.9	84.4
PointNet++ [5]	90.7	93.3	85.1
RSCNN [10]	91.7	94.2	86.6
DGCNN [6]	92.2	94.8	87.0
PointNet (+PA)	90.9 (1.7↑)	94.1 (2.2↑)	88.4 (4.0↑)
PointNet++ (+PA)	92.9 (2.2↑)	95.8 (2.5↑)	89.5 (4.4↑)
RSCNN (+PA)	92.7 (1.0↑)	96.0 (1.8↑)	90.1 (3.5↑)
DGCNN (+PA)	93.4 (1.2↑)	96.7 (1.9↑)	90.6 (3.6↑)

through our sample-aware auto-augmentation strategy to introduce more intra-class variation to the augmented samples.

Shape retrieval. To validate whether PointAugment facilitates the classifiers to learn a better shape signature, we compare the shape retrieval performance on MN40. Specifically, we regard each sample in the testing split as a query, and aim to retrieve the best similar shapes from the testing split by comparing the cosine similarity between their global features $\boldsymbol{F}_g$. In this experiment, we employ the mean Average Precision (mAP) as the evaluation metric.

Table 5.3 presents the evaluation results, which clearly show that PointAugment improves the shape retrieval performance

Table 5.3: Comparing the shape retrieval results (mAP, %) on MN40, for various methods equipped with conventional DA or with our PointAugment. Again, clear improvements are observed in retrieval accuracy for all methods.

Method	Conventional DA	PointAugment	Change
PointNet [33]	70.5	75.8	5.2↑
PointNet++ [5]	81.3	86.7	5.4↑
RSCNN [10]	83.2	86.6	3.4↑
DGCNN [6]	85.3	89.0	3.7↑

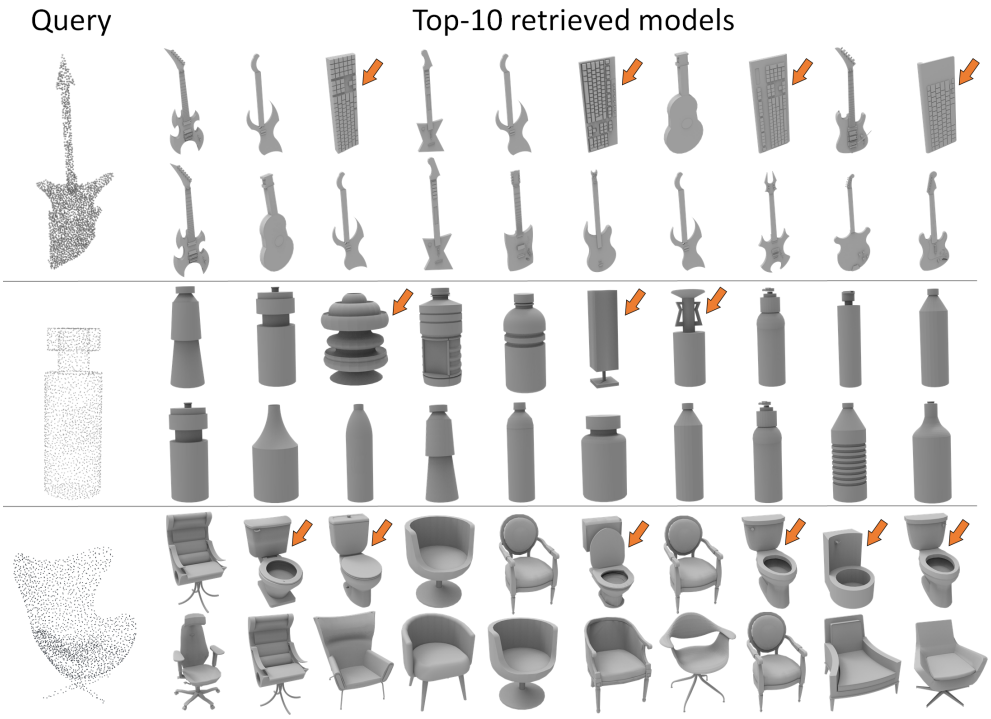


Figure 5.6: Shape retrieval results on MN40. For each query shape on the left, we present two rows of Top-10 retrieval results: the top row uses PointNet [33] and the bottom row uses PointNet+PointAugment. Note that the obviously-wrong retrieval results are marked with red arrows.

Table 5.4: Robustness test to compare our PointAugment with conventional DA. Here, we corrupt each input test sample by random jittering (Jitt.) with Gaussian noise in [-1.0, 1.0], by scaling with a ratio of 0.9 or 1.1, or by a rotation of 90° or 180° along the gravity axis. Also, we show the original accuracy (Ori.) without using corrupted samples.

Method	Ori.	Jitt.	0.9	1.1	90°	180°
Without DA	89.1	88.2	88.2	88.2	48.2	40.1
Conventional DA	90.7	90.3	90.3	90.3	89.9	89.7
PointAugment	**92.9**	**92.8**	**92.8**	**92.8**	**92.7**	**92.6**

for all the four classifier networks. Especially, for PointNet [33] and PointNet++ [5], the percentage of improvement is over 5%. Besides, we show visual results on shape retrieval for three different query models in Figure 5.6. Compared with the original PointNet [33], which is equipped with conventional DA, the augmented version with PointAugment produces more accurate retrievals.

5.4.3 PointAugment Analysis

Further, we conducted more experiments to evaluate various aspects of PointAugment, including a robustness test (Section 5.4.3), an ablation study (Section 5.4.3), and a detailed analysis on its Augmentor network (Section 5.4.3). Note that, in these experiments, we employ PointNet++ [5] as the classifier and perform experiments on MN40.

Robustness Test

We conducted the robustness test by corrupting test samples using the following five settings: (i) adding random jittering with Gaussian noise ranged $[-1.0, 1.0]$; (ii,iii) adding uniform scaling with a ratio of 0.9 or 1.1; and (iv,v) adding rotation with 90° or 180° along the gravity axis. For each setting, we use three different DA strategies: without DA, conventional DA, and our PointAugment.

Table 5.4 reports the results, where we show also the original test accuracy (Ori.) without using corrupted test samples as a reference. The results in the first two rows show that DA is an efficient way to improve the classification performance. Further, comparing the last two rows in the table, we can see that for all settings, our PointAugment consistently outperforms the conventional DA, which is random-based and may not yield good augmented samples all the time. Particularly, by comparing the results with the original test accuracy, PointAugment is less sensitive to corruption, where the achieved accuracy reduces only slightly. Such a result shows that PointAugment improves the robustness of a network with better shape recognition.

Ablation Study

Table 5.5 summarizes the results of the ablation study. Model A denotes PointNet++ [5] without our augmentor, which gives a baseline classification accuracy of 90.7%. On top of Model A,

Table 5.5: Ablation study of PointAugment. $\mathcal{D}$: point-wise displacement, $\mathcal{M}$: shape-wise transformation, DP: dropout, and Mix: mixed training samples (see Section 5.3.4).

Model	$\mathcal{D}$	$\mathcal{M}$	DP	Mix	Acc.	Inc.↑
A					90.7	-
B	✓				91.7	1.0
C		✓			91.9	1.2
D	✓	✓			92.5	1.8
E	✓	✓	✓		92.8	2.1
F	✓	✓	✓	✓	**92.9**	**2.2**

we employ our augmentor with point-wise displacement $\mathcal{D}$ alone (Model B), with shape-wise transformation $\mathcal{M}$ alone (Model C), or with both (Model D). From the results shown in the first four rows in Table 5.5, we can see that, each of the augmentation functions contributes more effective augmented samples.

Besides, we also ablate the dropout strategy (DP) for training, and the use of mixed training samples (Mix), as presented in Section 5.3.5, where we create Models E & F for comparison; see Table 5.5. By comparing the classification accuracies achieved by Models D, E, and F, we can see that both DP and Mix help to slightly improve the overall results. Note, these strategies are typically for stabilizing the model training and exploring more transformations.

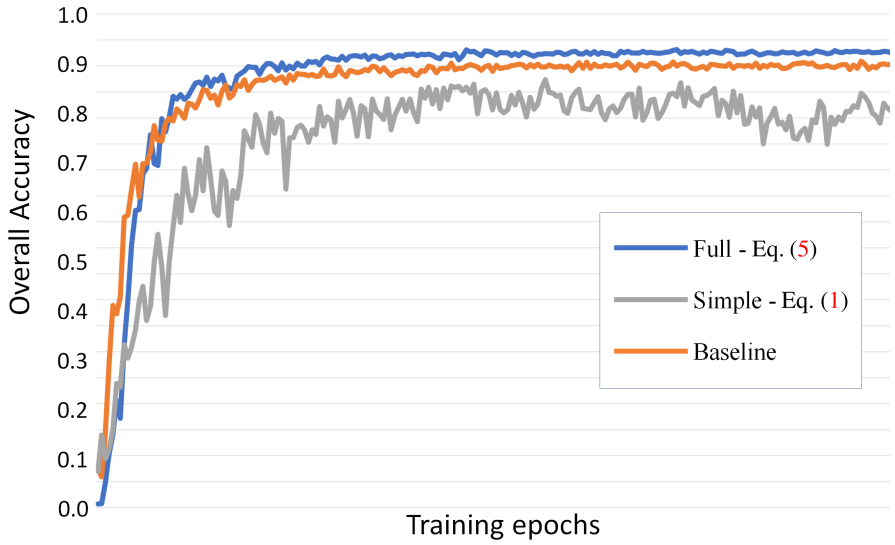


Figure 5.7: Evaluation curves: shape classification accuracy using different versions of $\mathcal{L}_{\mathcal{A}}$ over training epochs.

Augmentor Analysis

Analysis on $\mathcal{L}_{\mathcal{A}}$. As described in Section 5.3.2, we employ $\mathcal{L}_{\mathcal{A}}$ (see Eq. (5.5)) to guide the training of our augmentor. To demonstrate its superiority, we compare it with (i) a simple version (see Eq. (5.1)) and (ii) a baseline, *i.e.*, the conventional DA employed in PointNet++ [5]. Figure 5.7 plots the evaluation accuracy curves in terms of the training epochs.

Clearly, the training state achieved by using the simple version is very unstable; see the gray plot.This indicates that simply enlarging the difference between $\mathcal{P}$ and $\mathcal{P}'$ without restriction will create turbulence in the training process, resulting in a worse classification performance, when compared even with the baseline; see the orange plot. Comparing the blue and orange plots in Figure 5.7, we can see that, at the beginning of the train-

Table 5.6: Classification accuracy (%) for diff. λ in Eq. (5.5).

$\lambda = 0.5$	$\lambda = 1.0$	$\lambda = 2.0$
92.1	**92.9**	92.3

Table 5.7: Accuracy (%) with diff. feature extraction units.

DenseConv	EdgeConv	our
92.5	92.7	**92.9**

ing, since the augmentor is initialized randomly, the accuracy of employing PointAugment is slightly lower than the baseline. However, when the training continues, PointAugment rapidly surpasses the baseline and shows a clear improvement over the baseline, showing the effectiveness of designed augmentor loss.

Hyper-parameter λ in Eq. (5.5). Table 5.6 shows the classification accuracy for different choices of λ. As we mentioned in Section 5.3.2, a small λ encourages the augmentor to focus more on the classification. However, if it is too small, *e.g.*, 0.5, the augmentor tends to take no actions in the augmentation function, thus leading to a worse classification performance; see the comparisons in the left two columns in Table 5.6. On the other hand, if we set a larger λ, *e.g.*, 2.0, the augmented samples may be too difficult for the classifier. Such a result also hinders the network training; see the right-most column. Hence, in PointAugment, we adopt $\lambda = 1.0$ to treat them equally.

Analysis on the augmentor architecture design. As we mentioned in Section 5.3.1, we employ a per-point feature ex-

traction unit to embed per-point features $\boldsymbol{F}$ given the training sample $\mathcal{P}$. In our implementation, we use shared MLPs to extract $\boldsymbol{F}$. In this part, we further explore two other choices of feature extraction units for replacing the MLPs, including DenseConv [112] and EdgeConv [6]. Please refer to their original papers for the detailed methods. Table 5.7 shows the accuracy comparisons for the three implementations. From the results, we can see that, although using MLPs is a relative simple implementation compared with DenseConv and EdgeConv, it can lead to the best classification performance. We think of the following reasons. The aim of our augmentor is to regress a shape-wise transformation $\mathcal{M} \in \mathbb{R}^{3\times3}$ and a point-wise displacement $\mathcal{D} \in \mathbb{R}^{N\times3}$ from the per-point features $\boldsymbol{F}$, which is not a very tough task. If we apply a complex unit to extract $\boldsymbol{F}$, it may easily have overfitting problem. The results shown in Table 5.7 demonstrate that the MLPs is already enough for our augmentor to regress the augmentation functions.

5.4.4 Discussion and Future work

Overall, the augmentor network learns the sample-level augmentation function in a self-supervised manner, by taking the feedback from the classifier to update its parameters. Hence, the advantage on the classifier is that by exploring those well-tuned augmented samples, the classifier can enhance its capability and better learn to uncover intrinsic variations among the different

classes and discover the intra-class insensitive features.

In the future, we plan to adapt PointAugment for more tasks, such as part segmentation [33, 10], semantic segmentation [114, 11, 115], object detection [116, 38], upsampling [54, 101], denoising [49, 48], etc. However, it is worth to note particularly that different tasks require different considerations. For example, for parts segmentation, it is better for the augmentor to be part-aware and produce distortions on parts without changing the parts semantic; for object detection, the augmentor should be able to generate a richer variety of 3D transformations for various kinds of object instances in 3D scenes, etc. Therefore, one future direction is to explore part-aware or instance-aware auto-augmentation to extend PointAugment for other tasks.

5.5 Summary

We presented PointAugment, the first auto-augmentation framework that we are aware of for 3D point clouds, considering both the capability of the classification network and the complexity of the training samples. First, PointAugment is an end-to-end framework that jointly optimizes the augmentor and classifier networks, such that the augmentor can learn to improve based on feedback from the classifier and the classifier can learn to process wider variety of training samples. Second, PointAugment is sample-aware with its augmentor learns to produce augmentation functions specific to the input samples, with a shape-wise

transformation and a point-wise displacement for handling point cloud samples. Third, we formulate a novel loss function to enable the augmentor to dynamically adjust the augmentation magnitude based on the learning state of the classifier, so that it can generate augmented samples that best fit the classifier in different training stages. In the end, we conducted extensive experiments and demonstrated how PointAugment contributes to improve the performance of four representative networks on the MN40 and SR16 datasets.

□ **End of chapter.**

Chapter 6

SP-GAN: Sphere-Guided 3D Shape Generation and Manipulation

6.1 Introduction

A challenging problem in 3D shape creation is how to build generative models to synthesize new, diverse, and realistic-looking 3D shapes, while having structure-aware generation and manipulation. One solution is to decompose shapes into parts [117, 118] and learn to compose new shapes according to part-level relations. With parts correspondence, these approaches can further enable various forms of structure-aware manipulation; however, the granularity of the shape generation is part-based and it depends on the availability and quality of the parts annotations.

Another feasible solution is to design deep generative models to characterize the shape distribution and learn to directly gen-

Figure 6.1: SP-GAN not only enables the generation of diverse and realistic shapes as point clouds with fine details (see the two chairs on the left and right) but also embeds a dense correspondence across the generated shapes, thus facilitating part-wise interpolation between user-selected local parts in the generated shapes. Note how the left chair's back (blue part in top arc) and the right chair's legs (red part in bottom arc) morph in the two sequences.

erate shapes in an unsupervised manner. Prior researches generate shapes represented as 3D point clouds, typically by learning to map random latent code to point clouds, via auto-regressive models [74], flow-based models [41, 75, 77], and generative adversarial nets (GAN) [79, 81, 83]. Though substantial progress has been made, the problem is still very challenging, due to the diverse shape variations and the high complexity in 3D space. Hence, existing generative models often struggle with the fine details and tend to produce noisy point samples in the generated shapes. Also, existing models lack structure controllability for part-aware generation and manipulation; it is because the learned mapping from a single latent code merely characterizes the overall shape variation, so it is hard to obtain a plausible

correspondence between the parts in generated shapes and the dimensions in latent code, as well as across different shapes generated by the learned mapping.

In this chapter, we present a new GAN model called SP-GAN, in which we use a Sphere as Prior to guide the direct generation of 3D shapes (in the form of point clouds). Our new approach goes beyond generating diverse and realistic shapes, in which we are able to *generate point clouds with finer details and less noise*, as compared with the state-of-the-arts. Importantly, our design facilitates *controllability in the generative process*, since our model implicitly embeds a dense correspondence between the generated shapes, and training our model is *unsupervised*, without requiring any parts annotations. By this means, we can perform *part-aware generation and manipulation of shapes*, as demonstrated by the results shown in Figure 6.1.

Figure 6.2 illustrates the key design in SP-GAN. Instead of having a single latent code as input like the conventional generative models, we design our generator with two *decoupled* inputs: (i) *a global prior* $\mathcal{S}$, which is a fixed 3D point cloud in the form of a unit sphere, to provide an isotropic (unbiased) spatial guidance to the generative process; and (ii) *a local prior* $\mathbf{z}$, which is a random latent code to provide local details. We pack these two inputs into a *prior latent matrix* by attaching a latent code $\mathbf{z}$ to every point in $\mathcal{S}$, as illustrated in Figure 6.2. By this means, the generative process starts from a shared global initialization (*e.g.*, a common ground), yet accommodating the spatial varia-

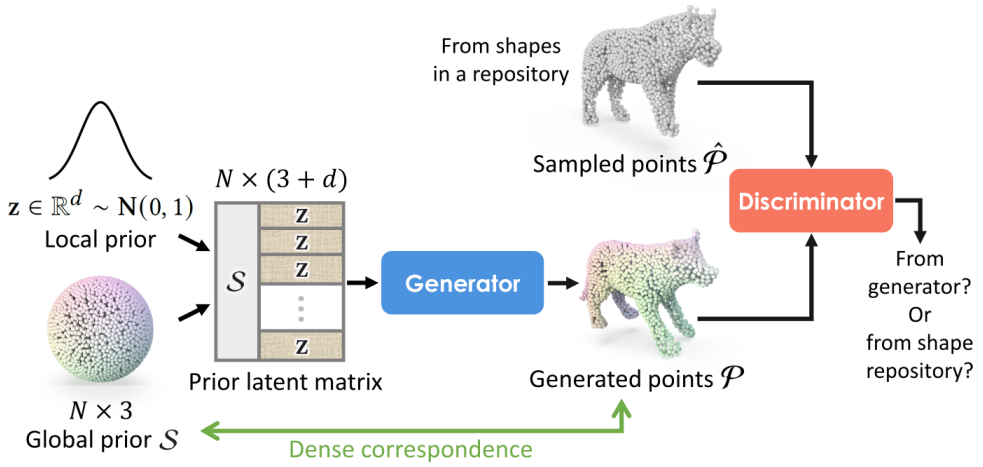


Figure 6.2: An overview of SP-GAN. Its input is a *prior latent matrix* with (i) a random latent code $\mathbf{z}$, and (ii) a unit 3D sphere $\mathcal{S}$, which is represented as N points evenly-distributed on the sphere. In SP-GAN, $\mathcal{S}$ provides an *unbiased spatial guidance* to the generator for it to learn to synthesize point cloud $\mathcal{P}$ from $\mathbf{z}$ with finer details and less noise. Also, SP-GAN implicitly embeds a *dense correspondence* between $\mathcal{S}$ and $\mathcal{P}$ (see their colors above), thus promoting structure-aware shape generation and manipulation.

tions, such that every point moves towards its desired location for forming the shape. A key insight behind our design is that we *formulate the generation task as a transformation* and *disentangle the complex 3D shape generation task* into (i) a global shape modeling (*e.g.*, chair) and (ii) a local structure adjustment (*e.g.*, chair's legs). Such a decoupling scheme eases the learning process and enhances the quality of the generated shapes.

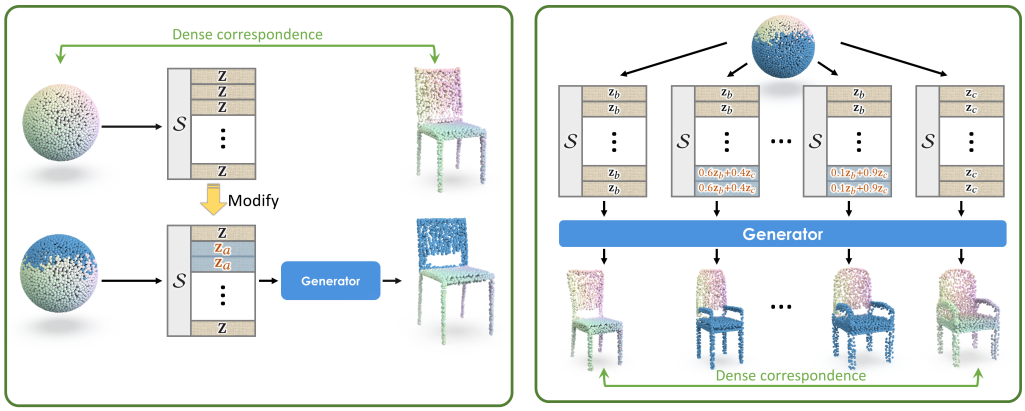
Another important consequence is that our new model facilitates structure controllability in the generative process, *through an implicit dense correspondence between the input sphere and the generated shapes*. Metaphorically, the sphere provides a common working space (like the canvas in painting) for shape

generation and manipulation; painting on a certain area on the sphere naturally manipulates the corresponding part in different generated shapes. So, if we modify the latent vectors associated with specific points on $\mathcal{S}$ while keeping others unchanged, we can manipulate local structures for the associated parts in the shape. Also, since parts are connected geometrically and semantically with one another, changing one part may cause slight changes in some other parts for structural compatibility; see Figure 6.1 (bottom). Furthermore, the dense correspondence extends *across all the generated shapes* produced from the generator, with $\mathcal{S}$ serving as a proxy. Hence, we can interpolate the latent code of different generated shapes to morph between shapes in a shape-wise or part-wise fashion.

In the following, We first elaborate an overview of our design (Section 6.2) and the details of our unsupervised generative framework (Section 6.3). Next, we show various structure-aware shape manipulations(Section 6.4), Finally, we evaluate the effectiveness of SP-GAN via extensive experiments (Section 6.5).

6.2 Overview

The basic model of SP-GAN is illustrated in Figure 6.2, with shapes represented as point clouds like the previous works we discussed. In SP-GAN, the generator consumes two *decoupled inputs*: a global prior $\mathcal{S}$, which contains the 3D coordinates of N points uniformly on a unit sphere, and a local prior $\mathbf{z}$,



(a) Part Editing (b) Part-wise Shape Interpolation

Figure 6.3: SP-GAN establishes a dense correspondence implicitly between sphere $\mathcal{S}$ and all the generated shapes, with sphere $\mathcal{S}$ as a proxy. By this model, we can modify or interpolate specific latent code in the prior latent matrix, e.g., for part editing (a) and part-wise shape interpolation (b).

which is a d-dimensional random latent vector, with each element randomly sampled from a standard normal distribution. Very importantly, we pack one latent vector $\mathbf{z}$ with each point in $\mathcal{S}$ to form the *prior latent matrix* as the generator input; see again Figure 6.2. During the training, we use the generator to synthesize point cloud $\mathcal{P} \in \mathbb{R}^{N\times 3}$ from $\mathcal{S}$ and $\mathbf{z}$; besides, we sample another point cloud $\hat{\mathcal{P}} \in \mathbb{R}^{N\times 3}$ from shape in a given 3D repository. So, when we train SP-GAN, the discriminator should learn to differentiate $\mathcal{P}$ and $\hat{\mathcal{P}}$, while the generator should learn to produce $\mathcal{P}$ that looks like $\{\hat{\mathcal{P}}\}$ from the 3D repository.

During the testing, we randomize a latent code and pack it with sphere $\mathcal{S}$ into a prior latent matrix, and feed it to the trained generator to produce a new shape; see $\mathcal{P}_a$ in Figure 6.3(a). Thanks to the sphere proxy, training SP-GAN creates an *implicit association* between points in $\mathcal{S}$ and points in the gener-

ated shape. This is a *dense point-wise correspondence*, as illustrated by the smoothly-varying point colors on the sphere and on the generated shape shown in Figure 6.3(a). Also, this dense correspondence extends across all the shapes produced from the generator; see the colors of the points in generated shapes $\mathcal{P}_b$ and $\mathcal{P}_c$ in Figure 6.3(b). With this important property, SP-GAN facilitates various forms of structure- and part-aware shape generation and manipulation. Below, we first describe two basic cases and more can be found later in Section 6.4:

- *Part editing.* The latent code associated with each point in sphere $\mathcal{S}$ is a local prior. If we change the latent code for some of the points, we can make local changes on the associated part in the generated shape. Figure 6.3(a) shows an example, in which we modify the latent code of the points associated with the chair's back (in blue) from $\mathbf{z}_a$ to $\mathbf{z}_{a'}$. By then, the generator produces a new chair $\mathcal{P}_{a'}$, whose back follows $\mathbf{z}_{a'}$ and other parts are slightly adjusted for compatibility with the newly-modified back.

- *Part-wise interpolation.* Further, the dense correspondence enables us to interpolate between the latent code of different generated shapes to morph shapes in part-wise fashion. This manipulation cannot be achieved by any existing unconditional generative models for 3D point clouds. Figure 6.3(b) shows an example, in which we interpolate the latent code associated with the chair's lower part (marked

as blue) between $\mathbf{z}_b$ and $\mathbf{z}_c$ by setting different interpolation weights. With this change, the generator produces a set of new chairs, in which their lower parts morph from $\mathcal{P}_b$ to $\mathcal{P}_c$.

6.3 Method

In this section, we first present the architecture design of the generator and discriminator networks in SP-GAN, and then present the training and implementation details.

6.3.1 Generator

Figure 6.4 shows the architecture of the generator in SP-GAN, which produces point cloud $\mathcal{P} \in \mathbb{R}^{N\times 3}$ from sphere $\mathcal{S}$ and a prior latent matrix. Inside the generator, $\mathbf{z}$ introduces diverse local styles and fine details into point features, whereas $\mathcal{S}$ serves as a prior shape for feature extraction and guides the generative process. Particularly, $\mathcal{S}$ also allows us to employ graph-related convolutions with spatial correlation for feature extraction. This is very different from existing works on 3D point cloud generation, which take only a single latent code as input; hence, existing models can only use fully-connected layers at early stages and require relatively large amount of learnable parameters, yet having limited expressiveness.

To elaborate on how our generator works, let's start with the bottom branch shown in Figure 6.4. First, the generator em-

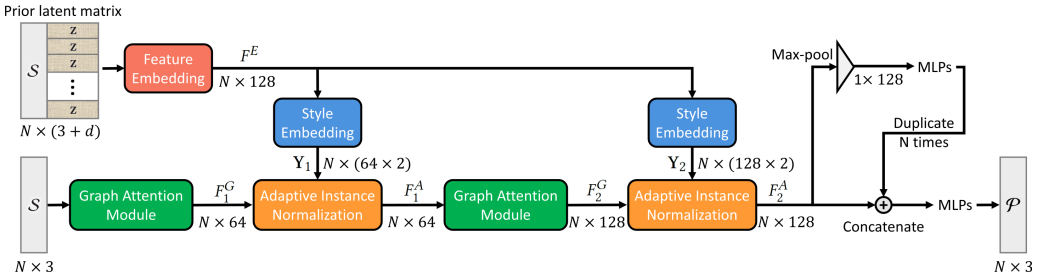


Figure 6.4: Architecture of the generator in SP-GAN.

ploys a graph attention module (to be detailed soon) to extract a point-wise feature map $F_1^G \in \mathbb{R}^{N\times 64}$ from $\mathcal{S}$. Now, to generate 3D shapes, an important question is how to insert the structure variations and local styles brought from latent code $\mathbf{z}$ into feature map F_1^G. Inspired by style transfer [119, 120], we propose to use an adaptive instance normalization layer (the left orange box in Figure 6.4) by modulating the mean and variance of the feature vectors in F_1^G. Specifically, as shown in the top branch, we employ a non-linear feature embedding, which is implemented using multi-layer perceptrons (MLPs), to transform the prior latent matrix into a high-level feature map $F^E \in \mathbb{R}^{N\times 128}$. Then, we use a style embedding (the left blue box in Figure 6.4), which is also implemented using MLPs, to specialize F^E into styles $\mathbf{Y}_1 = (\mathbf{Y}_1^s, \mathbf{Y}_1^b)$, where $\mathbf{Y}_1^s \in \mathbb{R}^{N\times 64}$ and $\mathbf{Y}_1^b \in \mathbb{R}^{N\times 64}$ control the scale and bias, respectively, when normalizing each point feature vector in F_1^G. Thus, the adaptive instance normalization operation becomes

$$F_1^A(i) = \mathbf{Y}_1^s(i) \cdot \frac{F_1^G(i) - \mu(F_1^G(i))}{\sigma(F_1^G(i))} + \mathbf{Y}_1^b(i), i \in [1, ..., N], \quad (6.1)$$

where $F_1^G(i)$ is the i-th point feature vector in F_1^G, $F_1^A(i)$ is

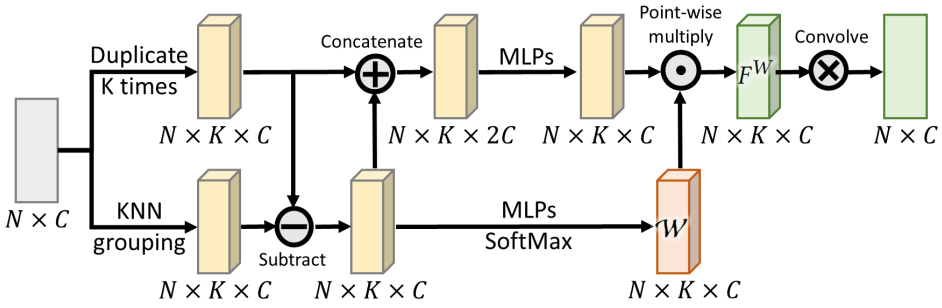


Figure 6.5: Architecture of the graph attention module.

the corresponding feature vector after the normalization, and $\mu(\cdot)$ and $\sigma(\cdot)$ compute the mean and standard deviation across the spatial axes of its argument, which is a feature vector. In general, Eq. (6.1) allows us to encode the learned per-point style, *i.e.*, $(\mathbf{Y}_1^s(i), \mathbf{Y}_1^b(i))$, into the associated embedded feature vector $F_1^G(i)$, so that the final adjusted F_1^A contains richer local details.

To further enrich the feature embedding, we pass F_1^A to another set of graph attention module and adaptive instance normalization to obtain F_2^G and F_2^A, respectively. As shown on the right side of Figure 6.4, we further regress the output 3D point cloud $\mathcal{P}$ by following PointNet [4] to reconstruct the 3D coordinates, *i.e.*, by concatenating F_2^A with the duplicated global feature vectors. Please refer to [4] for the details.

Graph attention module. Next, we elaborate on the design of the graph attention module; see Figure 6.5 for an illustration of its architecture. Here, we adopt the basic module (marked in light yellow) from DGCNN [73] and make our adjustments to further account for the relationships among the K neighbors

in the feature space. Instead of taking the neighboring features equally, we regress weights $\mathcal{W} \in \mathbb{R}^{N\times K\times C}$ (orange box) and employ point-wise multiplication to obtain the weighted neighbouring feature map F^W. Lastly, the output $N \times C$ feature map is obtained by applying a convolution of a kernel size of $1 \times K$ to F^W.

6.3.2 Discriminator

Figure 6.6 shows the architecture of the discriminator in SP-GAN. Its input is either a point cloud $\mathcal{P}$ produced by the generator or a point cloud $\hat{\mathcal{P}}$ sampled from shape in a given 3D repository. From the N points in $\mathcal{P}$ or $\hat{\mathcal{P}}$, a conventional discriminator would learn a $1\times C$ global feature vector for the whole shape and predict a single score that indicates the source of the point cloud. Considering that the "realism" of an input can be explored by looking into its local details, as inspired by [121], we extend the conventional discriminator to additionally perform classifications on a per-point basis, *i.e.*, by predicting a score for each point in the input.

As shown in Figure 6.6, after we extract point features from the input, we use the top branch to predict a per-shape score and the bottom branch to predict per-point scores. In this way, the discriminator can effectively regularize both global and local variations in the input point cloud. Correspondingly, we can then encourage the generator to focus on both global structures

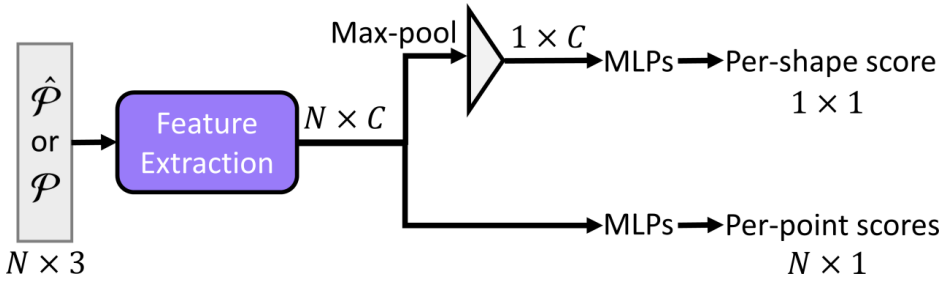


Figure 6.6: Architecture of the discriminator in SP-GAN.

and local details, such that it can better synthesize point clouds to fool this more powerful discriminator. Note that in our implementation, we adopt PointNet [4] as the backbone for the feature extraction, which is the violet box shown in Figure 6.6.

6.3.3 Training and Implementation details

Loss functions. SP-GAN can be trained end-to-end using one loss for generator G and another loss for discriminator D. In this work, we design the two losses based on the least squares loss [93], which is originally formulated by minimizing the following competing objectives in an alternating manner:

$$\mathcal{L}_G = \frac{1}{2}[D(\mathcal{P}) - 1]^2 \tag{6.2}$$

$$\text{and} \quad \mathcal{L}_D = \frac{1}{2}[(D(\mathcal{P}) - 0)^2 + (D(\hat{\mathcal{P}}) - 1)^2], \tag{6.3}$$

where $D(\mathcal{P})$ and $D(\hat{\mathcal{P}})$ are the confidence values predicted by the discriminator on $\mathcal{P}$ and $\hat{\mathcal{P}}$, respectively, and $\mathcal{L}_G$ and $\mathcal{L}_D$ are the loss for training the generator and discriminator, respectively.

Since our discriminator outputs two types of predictions, *i.e.*, per-shape score and per-point scores. Hence, we model the dis-

criminator loss $\mathcal{L}_D$ as a summation of shape loss $\mathcal{L}_D^{\text{shape}}$ and point loss $\mathcal{L}_D^{\text{point}}$:

$$\mathcal{L}_D = \mathcal{L}_D^{\text{shape}} + \lambda \mathcal{L}_D^{\text{point}}, \tag{6.4}$$

$$\mathcal{L}_D^{\text{shape}} = \frac{1}{2}[(D(\mathcal{P}) - 0)^2 + (D(\hat{\mathcal{P}}) - 1)^2], \tag{6.5}$$

$$\mathcal{L}_D^{\text{point}} = \frac{1}{2N} \sum_{i=1}^{N} [(D(p_i) - 0)^2 + (D(\hat{p}_i) - 1)^2]. \tag{6.6}$$

where λ is a balance parameter; p_i and $\hat{p}_i$ are the i-th point in $\mathcal{P}$ and $\hat{\mathcal{P}}$, respectively; and $\mathcal{L}_D^{\text{point}}$ is computed by averaging the predictions of all the points.

Correspondingly, the objective for training the generator becomes

$$\mathcal{L}_G = \frac{1}{2}[D(\mathcal{P}) - 1]^2 + \beta \frac{1}{2N} \sum_{i=1}^{N} [D(p_i) - 1]^2, \tag{6.7}$$

where β is a weight to balance the terms.

Network training. Overall, we train SP-GAN by alternatively optimizing the discriminator using Eq. (6.4) and the generator using Eq. (6.7). Note also that, we keep sphere $\mathcal{S}$ unchanged during the training and randomly sample a latent code $\mathbf{z}$ from a standard normal distribution in each of the training iterations.

Network inference. During the generation phase, we only need to use the trained generator to produce a point cloud, while keeping the same $\mathcal{S}$ as in the training. With different randomly-sampled $\mathbf{z}$ as inputs, the generator can produce diverse point cloud outputs.

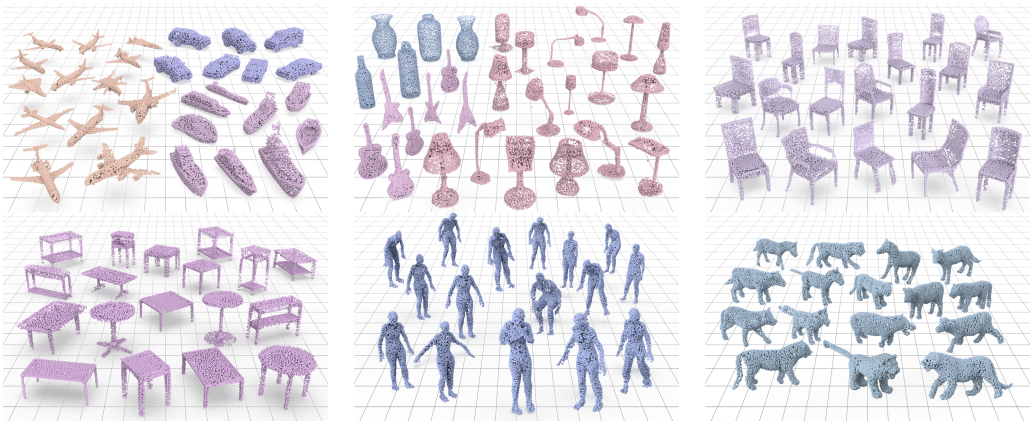


Figure 6.7: Galleries showcasing shapes of various categories generated by SP-GAN, and the rich global structures and fine details in the shapes.

Implementation details. We implement our framework using PyTorch and train it on a single NVidia Titan Xp GPU using the Adam optimizer. We use a learning rate of 10^{-4} to train both the generator and discriminator networks, and follow the alternative training strategy in [122] to train each of them for 300 epochs. In both networks, we use LeakyReLU as a nonlinearity activation function. In the last layer of the generator, we use tanh as the activation function. Also, we set K=20 to extract point neighborhood.

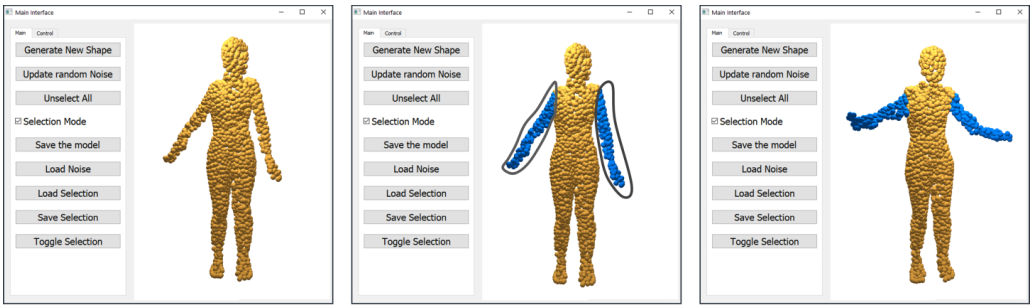
6.4 Shape Generation and Manipulation

This section presents various forms of shape generation, manipulation, and analysis that are enabled by SP-GAN.

Figure 6.8: Visualizing the dense correspondence between the sphere proxy and the generated shapes. Note that same color is assigned to associated points on the sphere proxy and on the generated shapes.

6.4.1 Shape Generation

Galleries of generated shapes. Figure 6.7 showcases varieties of shapes generated by SP-GAN in the form of point clouds. Besides the commonly-used ShapeNet dataset [123], we adopted SMPL [124] (human body shapes) and SMAL [125] (animal shapes) as our training data. In detail, on the 3D mesh of each shape, we uniformly sampled N points ($N = 2,048$ by default) and normalized the points to fit a unit ball. Like the existing generative models [79, 81, 83], we trained our network on each category separately. Note that we regard different kinds of animals as individual category. During the generation phase, we randomly sample a latent code from a standard Gaussian distribution and take it to SP-GAN to generate the associated shape (point cloud). As shown in Figure 6.7, our generated shapes exhibit fine details with less noise and also cover a rich variety of global and local structures.



(a) Generate a new shape (b) Lasso to select parts (c) Show manipulation result

Figure 6.9: Our interface for interactive shape generation and manipulation. Given a generated shape (a), one can use a lasso tool to select specific parts in the shape (b) and preview the manipulation result (c) in real-time.

Visualization of the dense correspondence. To visualize the learned implicit dense correspondence between the points on the sphere proxy $\mathcal{S}$ and the points in the generated shapes, we use same color to render associated points on the sphere and on the generated shapes; see Figure 6.8 for examples. Checking the point colors across sphere and generated shapes, we can see that different regions of the sphere correspond to different local parts in the shapes. For example, the lower left part of the sphere corresponds to the legs of the chairs. Also, in each category, the same semantic part of different objects, *e.g.*, the back of all chairs and the legs of all animals, receive similar colors. Particularly, as shown in the bottom row, these animals have different poses, yet our model can successfully establish a dense correspondence between the semantically-associated parts. This correspondence property is crucial to enable various structure-aware shape manipulations, as we shall show soon.

6.4.2 Single-shape Part Editing

With a dense correspondence established between the sphere proxy and each generated shape, we can easily locate "local" latent code associated with specific parts in a shape. As illustrated in Figure 6.3(a), we can perform part editing by re-sampling a new random latent code to replace the existing latent code associated with each specific part in the shape; by then, we can generate a new shape that is similar to the original one but with a different local part.

To facilitate part selection, we built the interactive interface shown in Figure 6.9, in which one can use a lasso tool to interactively select specific parts in a generated shape. The blue points in (b) indicate the lasso selection. Then, one can click a button to regenerate the shape with a real-time preview (c). Figure 6.10 shows more part-zediting examples. On the left of each row is an initial generated shape without any manipulation. By selecting specific parts (blue) in the shape, we can randomly modify the associated latent codes and produce the new shapes shown on the right. We can observe that, after modifying a portion of all the latent codes, the generator will synthesize new shapes with different local styles on the user-marked blue parts, while other object parts (yellow) only exhibit small changes for compatibility with the newly-modified parts.

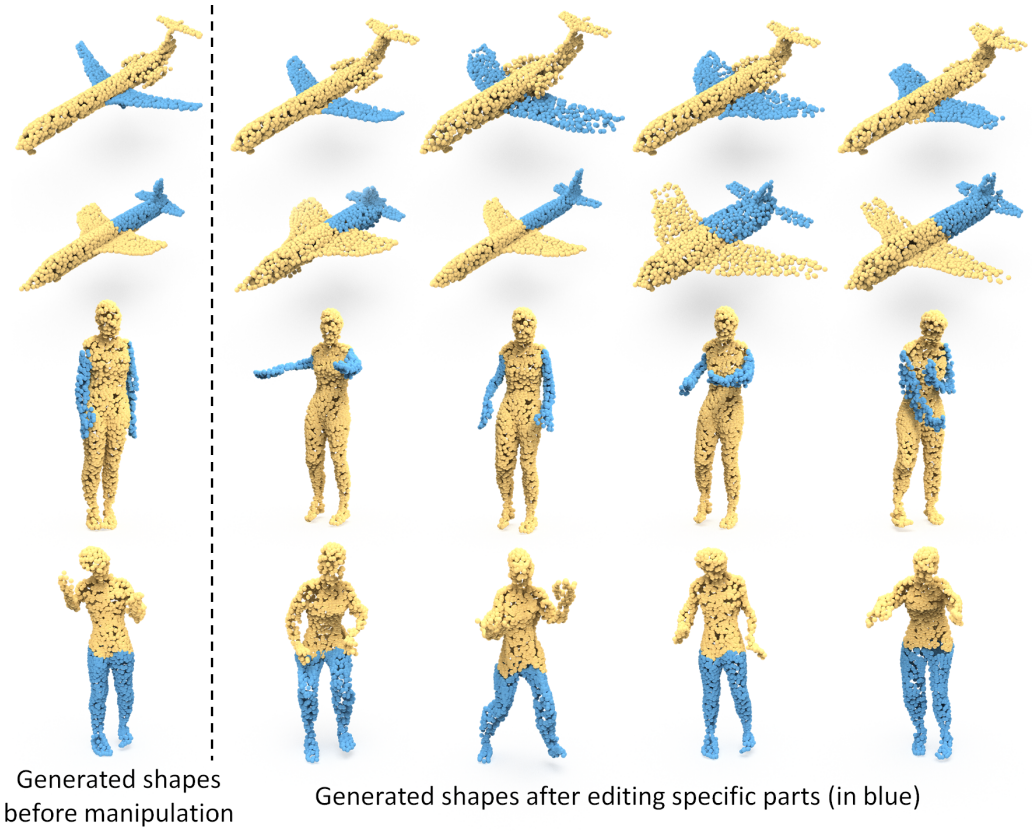


Figure 6.10: Part editing examples. Given a generated shape (left), we can select specific parts (blue) and modify their associated latent codes to produce new shapes with different styles (right).

6.4.3 Shape Interpolation

Shape-wise interpolation. Like existing models [79, 41, 81], we can also linearly interpolate between shapes of different appearance using SP-GAN. Specifically, given two different generated shapes $\mathcal{P}_a$ (source) and $\mathcal{P}_b$ (target) with their associated latent codes $\mathbf{z}_a$ and $\mathbf{z}_b$, respectively, the interpolated shape $\mathcal{P}_c$ can be easily generated by feeding the interpolated latent code $\mathbf{z}_c = (1-\alpha) \cdot \mathbf{z}_a + \alpha \cdot \mathbf{z}_b$ into our generator, where α is the

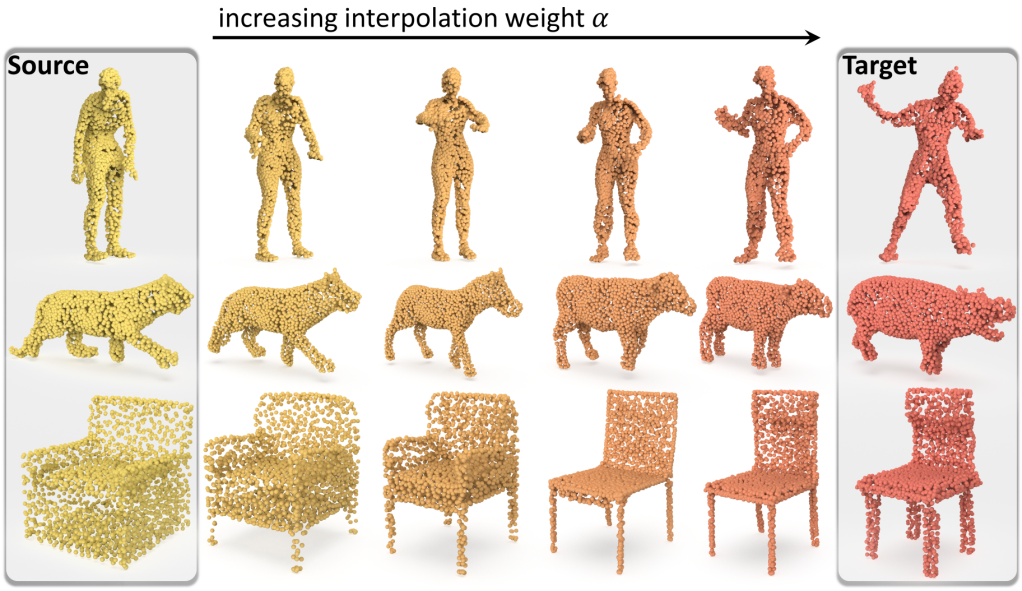


Figure 6.11: Shape-wise interpolation between the source (left) and target (right-most) in each row. Note the smooth transition from left to right, including the two human bodies with different poses (top row). Also, the generated intermediate shapes are high-quality with fine details.

interpolation weight ranged from zero to one.

From the interpolation results shown in Figure 6.11, we can observe that as α increases, SP-GAN enables a smooth transition from the source to the target. See particularly the top row, the source and target reprensent the same human body of two different poses; the interpolation is able to produce rather smooth transitions between the two poses. Distinctively, just like the source and target, the intermediate (generated) shapes also contain fine details with little noise; this is very challenging to achieve, as many existing works easily introduce noise to the interpolated point clouds.

Part-wise interpolation. As shown in Figure 6.3(b), with the

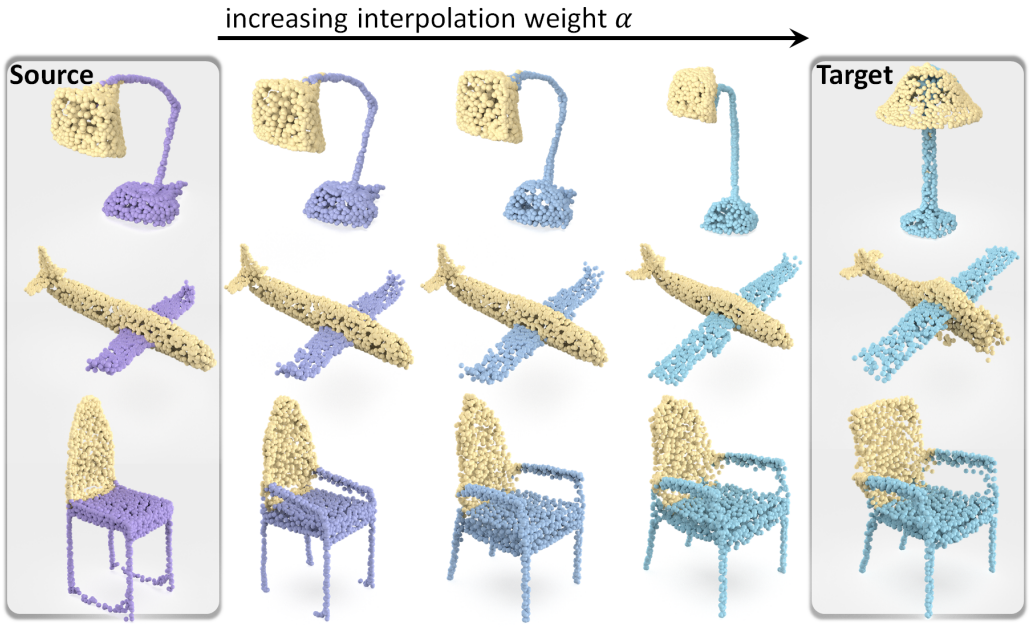


Figure 6.12: Interpolating only the latent code associated with the user-selected parts (non-yellow) between the source (left) and target (right) in each row. Note the smooth and part-based transition, as well as the quality of the generated point samples in the interpolated shapes.

dense correspondence established across the generated shapes, we can interpolate the latent code associated with specific part in a shape, instead of simply interpolating the whole shape.

Figure 6.12 shows three sets of part-wise interpolation results, where we interpolate only the latent codes associated with the user-selected points (non-yellow) between the source and target shapes. Here, we render the user-selected points using a color gradient from purple to blue to reveal the interpolation progress. From the results shown in the figure, we can observe that the interpolation happens mainly on the user-selected points, yet the remaining points (yellow) may exhibit small changes for the integrity of the generated shape. For example, in the first row,

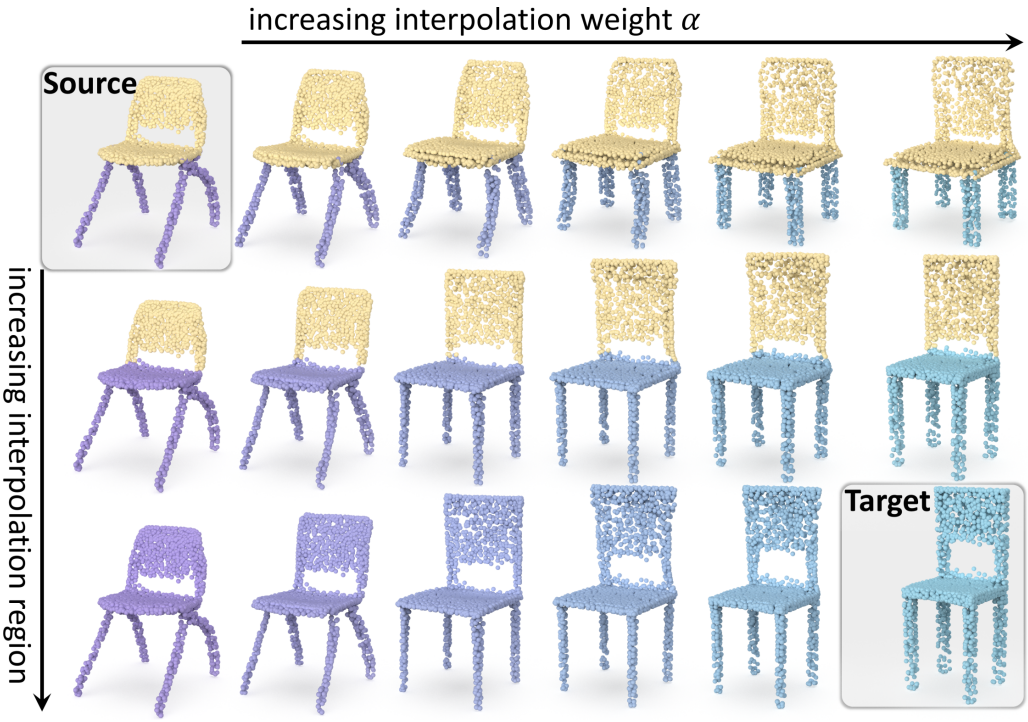


Figure 6.13: A two-dimensional part-wise interpolation example. Along each row (left to right), we interpolate the user-selected part, whereas along each column (top to bottom), we gradually enlarge the user-selected region. From the results shown above, we can see the smooth transition and the quality of the interpolated shapes produced by SP-GAN.

we select the bracket and base of the source and target lamps for interpolation, so the lamp's shape does not change significantly, while the style of the bracket and base gradually morphs from that of the source to the target.

Figure 6.13 further shows a two-dimensional part-wise interpolation example. Along each row, we fix the part being interpolated (non-yellow) and gradually increase the interpolation weight α. Taking the first row as an example, we only selected the legs for interpolation. As α increases, the legs of the source

(top-left) gradually becomes more and more similar to the legs of the target (bottom-right); see also the color gradient from purple to blue. On the other hand, along each column (top to bottom), we fix α but gradually enlarge the selected region. Taking the right-most column as an example, in which $\alpha = 1$, we can observe that as the selected region (blue) increases, the source gradually deforms further to become the target. Again, we would like to highlight that shapes generated by SP-GAN have fine details and little noise, including also the interpolated ones; see again Figures 6.12 and 6.13.

6.4.4 Multi-shape Part Composition

The implicit dense correspondence across multiple generated shapes also facilitates the composition of parts from multiple shapes. Specifically, given different semantic parts (*e.g.*, chair's back, legs, *etc.*) and their associated latent codes from different shapes, we can pack a new prior latent matrix with $\mathcal{S}$ and feed the matrix to our generator to produce new shapes. In this way, the synthesized shape would integrate different local styles from the parent shapes.

Figure 6.14 presents two examples, where the non-yellow parts in the shapes (top row) are selected for composition. We can observe that, for the two re-generated shapes (bottom row), their local parts generally inherit the styles of the parent shapes.

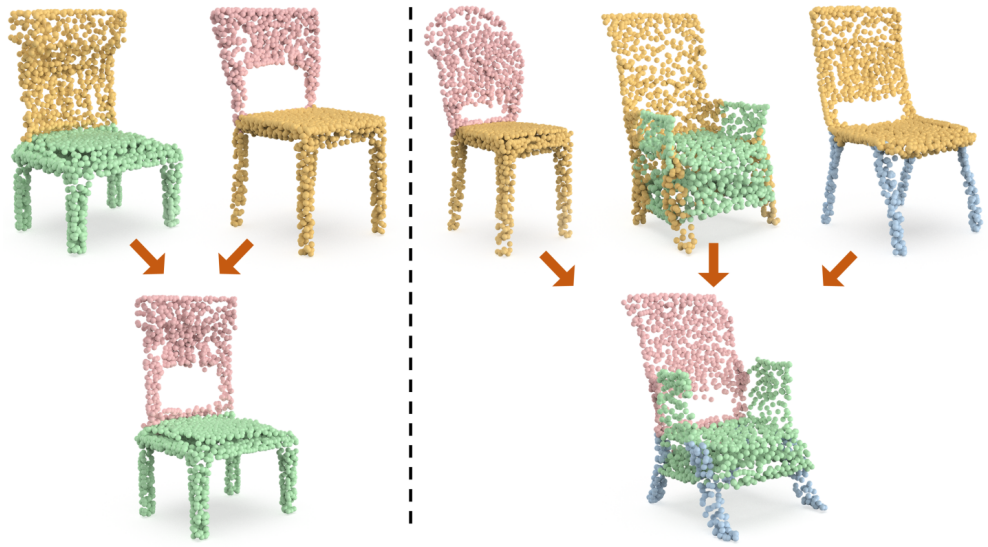


Figure 6.14: Two multi-shape part composition examples (left and right), in which the non-yellow parts in the top row are selected in the composition.

6.4.5 Part Co-Segmentation

Another benefit brought by the implicit dense correspondence is that SP-GAN enables part co-segmentation on a set of 3D shapes, *i.e.*, simultaneous segmentation of shapes into semantically-consistent parts. To do so, we only need to manually segment one of the shapes in the set, then the point-wise labels of the remaining shapes could be simultaneously obtained through the dense correspondence.

Figure 6.15 showcases an interesting co-segmentation example, in which only the top-left shape was manually segmented and all other shapes in the figure are *automatically* co-segmented. We can observe that, the learned implicit correspondence produced by SP-GAN can successfully facilitate part co-segmentation across shapes, even for shapes with different poses and local part

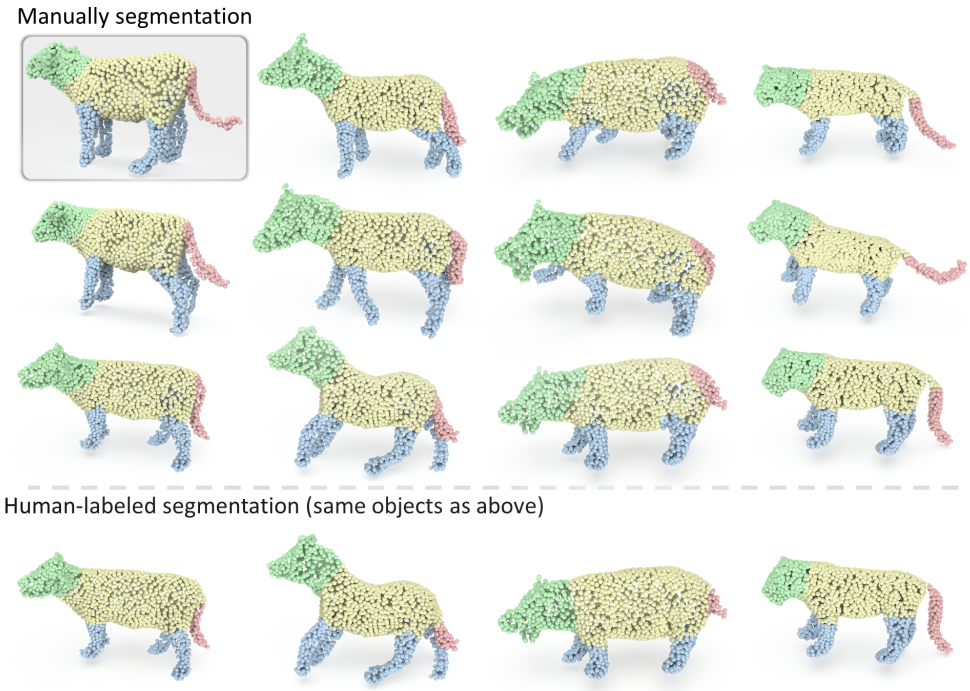


Figure 6.15: Top: co-segmentation results relying on the dense correspondence across different generated shapes, where only the top-left object (marked over a gray background) is manually segmented and the others are automatically co-segmented. Note that each column represents a different kind of animal. Bottom row: the associated human-labeled results.

orientations, *e.g.*, see the tails of the animals in the figure. Particularly, different columns actually feature different types of animals, yet our method can still achieve high robustness in co-segmenting these shapes.

Since generative tasks have no ground truths, we cannot directly evaluate the dense correspondence between the generated results. So, we resort to comparing our shape co-segmentation results with human-labeled results. Specifically, we manually label parts in 100 generated shapes of SMAL and calculate the mIoU between the co-segmentation results and human labels.

The results are very positive (87.7%). Also, we conducted a user study with 10 participants (who are graduate students aged 23 to 28), to compare the 1-shot co-segmented sample and human-labeled sample; the preference statistics of our result is 48.5%, which is close to a random selection. Figure 6.15 (bottom) shows some of the human-labeled results.

6.5 Evaluation and Discussion

6.5.1 Comparing with Other Methods

We evaluate the shape generation capability of our method against five state-of-the-art generative models, including r-GAN [79], tree-GAN [81], PointFlow [41], PDGN [83], and ShapeGF [78]. Here, we follow the same experimental setup as the above works to conduct this experiment. That is, we trained our SP-GAN model also on the Chair (and Airplane) shapes in ShapeNet [123], randomly generated 1,000 shapes for each category, and then evaluated the generated shapes using the same set of metrics as in the previous works (details to be provided below). Also, for the five state-of-the-art models being compared, we directly employ their publicly-released models to generate shapes.

Evaluation metrics. Different from supervised tasks, generation tasks have no explicit ground truths for evaluations. Hence, we directly follow the evaluation metrics in the existing works [79, 81, 83]: (i) Minimum Matching Distance (MMD)

measures how close the set of generated shapes is relative to the shapes in the given 3D repository $\{\hat{\mathcal{P}}\}$, so MMD indicates the fidelity of the generated shapes relative to $\{\hat{\mathcal{P}}\}$. (ii) Coverage (COV) measures the fraction of shapes in $\{\hat{\mathcal{P}}\}$ that can be matched (as a nearest shape) with at least one generated shape, so COV indicates how well the generated shapes cover the shapes in $\{\hat{\mathcal{P}}\}$; and (iii) Fréchet Point Cloud Distance (FPD) measures the 2-Wasserstein distance in the feature space between the generated shapes and the shapes in $\{\hat{\mathcal{P}}\}$, where we employ a pre-trained classification model, *i.e.*, DGCNN [73], for the feature extraction, so FPD indicates the distribution similarity between the generated shapes and $\{\hat{\mathcal{P}}\}$. For details of these metrics, readers may refer to [79, 81, 83]. Although these metrics are not absolutely precise, they still reflect the quality of the generated shapes in various aspects. Overall, a good method should have a low MMD, high COV, and low FPD.

Quantitative evaluation. Table 6.1 reports the quantitative comparison results between different methods. For a fair comparison of training time, all methods (including ours) were run on the same desktop computer with the same GPU, and for the five comparison methods, we used code from their project websites. From the results shown in the table, we can see that SP-GAN outperforms all the other methods consistently on the three evaluation metrics for both datasets by a large margin. The low MMD and FPD suggest that our generated shapes have

Table 6.1: Quantitative comparison of the generated shapes produced by SP-GAN and five state-of-the-art methods. The unit of MMD is 10^{-3}.

Categories	Metrics	Methods					
		r-GAN	tree-GAN	PointFlow	PDGN	ShapeGF	SP-GAN
Airplane	MMD(↓)	3.81	4.32	3.68	3.27	3.72	**1.95**
	COV(%,↑)	42.17	39.37	44.98	45.68	46.79	**50.50**
	FPD(↓)	3.54	2.98	2.01	1.84	1.61	**0.96**
Chair	MMD(↓)	18.18	16.14	15.02	15.56	14.81	**8.24**
	COV(%,↑)	38.52	39.37	44.98	45.89	47.79	**52.10**
	FPD(↓)	5.28	4.44	3.83	3.77	3.52	**2.13**
Parameter size (M)		7.22	40.69	1.61	12.01	5.23	**1.01**
Training time (Days)		0.9	4.1	2.5	2.3	1.2	**0.5**

high fidelity compared with the shapes in the 3D repository in both the spatial and feature space, and the high COV suggests that our generated shapes have a good coverage of the shapes in the 3D repository. Also, our network has the fewest learnable parameters and requires much less time to train.

Qualitative evaluation. Figure 6.16 shows some visual comparison results. Here, we pick a random shape generated by our method and use the Chamfer distance to retrieve the nearest shapes generated by each of the other comparison methods. From these results, we can see that the point clouds generated by our method (e) clearly exhibit more fine details and less noise, while other methods (a)-(d) tend to generate noisy samples.

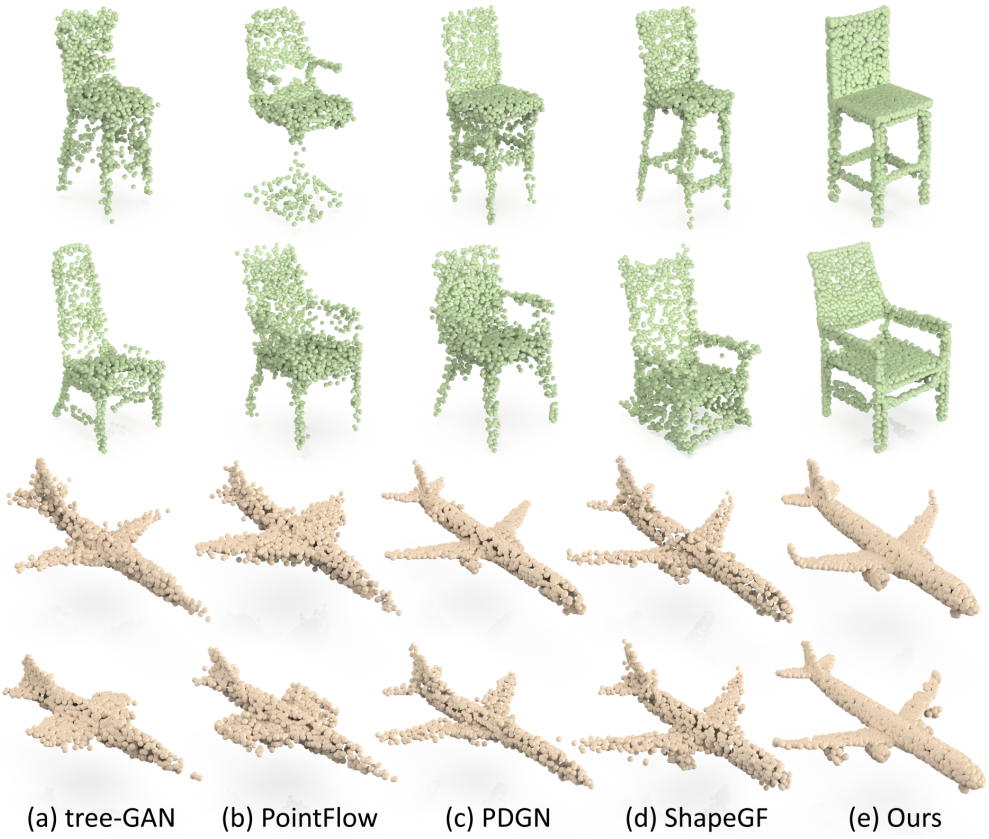


Figure 6.16: Visual comparisons with state-of-the-art methods. Clearly, the point clouds generated by our method (e) exhibit more fine details and less noise, while other methods (a)-(d) tend to generate noisy point samples.

Table 6.2: Comparing the generation performance of our full pipeline with various simplified cases in the ablation study. The unit of MMD is 10^{-3}.

Model	GAM	AdaIN	PPS	Sphere	MMD(↓)	COV(%,↑)	FPD(↓)
(i)		✓	✓	✓	10.38	48.24	2.88
(ii)	✓		✓	✓	9.54	49.72	2.66
(iii)	✓	✓		✓	9.81	50.63	2.52
(iv)	✓	✓	✓		8.69	51.22	2.29
Full	✓	✓	✓	✓	**8.24**	**52.10**	**2.13**

6.5.2 Ablation Study

To evaluate the effectiveness of the major components in our framework, we conducted an ablation study by simplifying our full pipeline for four cases: (i) replace the graph attention module (GAM) with the original EdgeConv [73]; (ii) remove the adaptive instance normalization (AdaIN) and directly take the prior latent matrix as the only input (see Figure 6.4); (iii) remove per-point score (PPS) (see Figure 6.6); and (iv) replace the sphere with a unit cube. In each case, we re-trained the network and tested the performance on the Chair category. Table 6.2 summarizes the results of (i) to (iv), demonstrating that our full pipeline (bottom row) performs the best and removing any component reduces the overall performance, thus revealing that each component contributes.

6.5.3 Shape Diversity Analysis

Further, to show that our generator is able to synthesize diverse shapes with fine details and it does not simply memorize the latent code of the shapes in the training set $\{\hat{\mathcal{P}}\}$, we conducted a shape retrieval experiment. Specifically, we take random shapes produced by our generator to query the shapes in the training set, *i.e.*, we find the shapes in training set $\{\hat{\mathcal{P}}\}$ that have the smallest Chamfer distance with each generated shape. Figure 6.17 shows some of the shape retrieval results. Compared with the top-five similar shapes retrieved from $\{\hat{\mathcal{P}}\}$, we can see

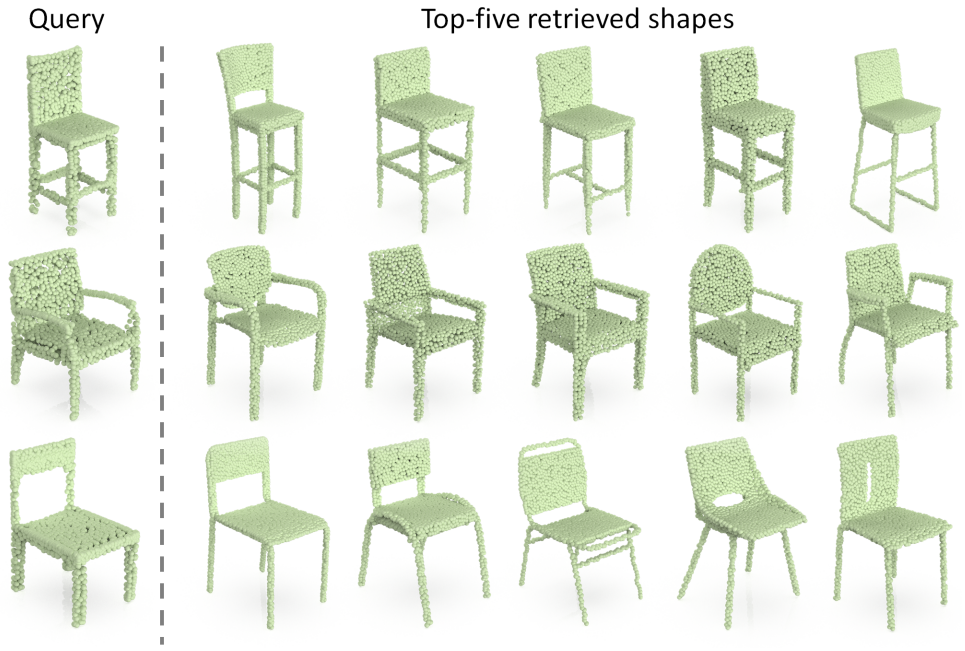


Figure 6.17: Shape diversity analysis. We take shapes (left) randomly produced by our generator to query the shapes in the training set.

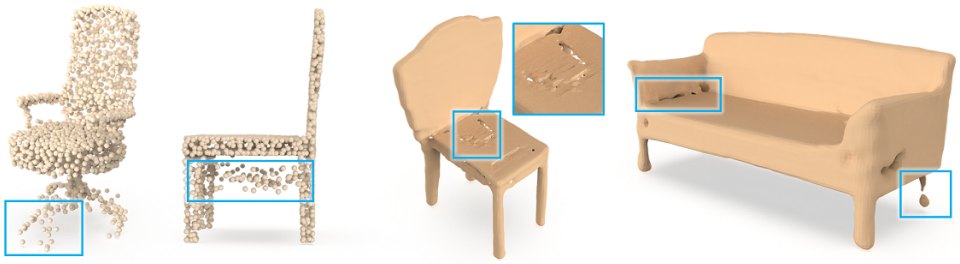


Figure 6.18: Failure cases. Left: complex and thin structures may be blurry. Right: distorted edges(holes) could be produced on the reconstructed surface.

that our generated shapes have similar overall structures yet different local fine details. Particularly, the points in our generated shapes have less noise (compared with those produced by other unsupervised generative models), just like the sampled ones from the training set.

6.5.4 Discussion on Limitations

Our approach still has several limitations. (i) SP-GAN is able to learn in an unsupervised manner but it still requires a large amount of shapes for training. Hence, for shapes with limited training samples or with complex or thin structures, the generated shapes may still be blurry (or noisy); see Figure 6.18 (left). (ii) Though the point cloud representation is flexible for shape generation and manipulation, we can not directly produce surface or topological information and require a post processing to reconstruct the surface. So, distorted edges and holes could be produced on the reconstructed surface; see Figure 6.18 (right). In the future, we plan to explore point normals in the generation process to enhance the mesh reconstruction. (iii) Lastly, parts relations are not explicitly or clearly represented in our model, even it embeds an implicit correspondence; looking at the human-labeled samples in Figure 6.14, we may incorporate certain parts priors into the network to enrich the generation.

6.6 Summary

In this chapter, we presents SP-GAN, a new generative model for direct generation of 3D shapes represented as point clouds. By formulating the generator input using the prior latent matrix, we decouple the input into a global prior (sphere points) and a local prior (random latent code), and formulate the gener-

ator network with style embedding and adaptive instance normalization to bring local styles from the random latent code into the point features of the sphere points. This disentanglement scheme articulates the 3D shape generation task as a global shape modeling and a local structure regulation, and enables the generative process to start from a shared global initialization, yet accommodating the spatial variations.

Very importantly, distinctive from existing works on direct point cloud generation, our new design introduces structure controllability into the generative process through the implicit dense correspondence. So, we can modify or interpolate latent codes in a shape-wise or part-wise manner, and enable various forms of structure-aware shape manipulations that cannot be achieved by previous works on direct point cloud generation. Both quantitative and qualitative experimental results demonstrate that SP-GAN is able to generate diverse, new, and realistic shapes that exhibit finer details and less noise.

□ **End of chapter.**

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In the thesis, we introduce four novel deep learning methods for enhancing the 3D point cloud representation and artificially enlarging the 3D datasets. All of them can be leveraged for improving the performance and generalization of 3D point cloud analysis and applications.

First, in Chapter 3, to address the distribution sparsity problem of the scanned point set, we presented an efficient upsampling network, i.e., PU-GAN, by leveraging the generative network for generating a denser and uniform set of points from the input sparse set. After revisiting this task, in Chapter 4 we introduce Dis-PU to disentangle the task based on its multi-objective nature and formulate two cascaded sub-networks. Extensive experiments demonstrated the effectiveness and superiority of both PU-GAN and Dis-PU over the state-of-the-arts.

Then, in Chapter 5, we presented PointAugment, an auto-

augmentation framework to enlarge the quantity and diversity of the training samples and improve the network generalization. It automatically optimizes and augments point cloud samples to enrich the data diversity when we train a classification network. By replacing conventional data augmentation with our method, clear improvements are shown in 3D classification and retrieval.

Lastly, in Chapter 6, we presented an unsupervised method SP-GAN for shape generation and manipulation. Compared with existing models, our method is able to synthesize diverse and high-quality shapes with fine details and promote controllability for part-aware shape generation and manipulation, yet trainable without any parts annotations.

7.2 Future Work

Though the significant progress of point cloud analysis is achieved in recent years, 3D deep learning is still a young field, there are many more interesting directions for future exploration.

The current models for point feature extraction are still not as powerful as CNNs in the 2D image domain. More effective point feature extractors or architectures are required for sufficiently exploring both the local and global 3D structures and also for analyzing large-scale city scenes with large amounts of points.

Deep networks are often vulnerable to adversarial attack which are carefully crafted instances to cause the models to make wrong predictions. Despite the extensive studies in 2D images,

less attention has been paid to the 3D domain like point clouds. With the increasing popularity of 3D sensors, the research on adversarial attacks and defenses will become essential, particularly in safety-critical systems such as autonomous driving.

Another direction is to develop architectures that further respect rotation invariance or even achieve rotation equivariance. Although existing deep learning networks have certain robustness on rotated point clouds via a spatial transformer or data augmentation, there is no guarantee of any invariance or equivariance in rotations. Some works tried to address the problem with spherical CNNs or rotation-invariant features, but they are restricted to single objects rather than 3D scenes, or have an inferior performance compared to use Cartesian coordinates.

Few-shot learning and unsupervised learning are essential in the field of 3D analysis. The workload of 3D data collection and high-quality labeling is far more than that in 2D domain. It includes a series of professional activities, such as scene scanning, complex post-processing, and tedious annotations, thus resulting in a smaller size of training dataset so far. It is desirable to leverage the intrinsic geometric properties of 3D objects to provide auxiliary information for network learning, such as the self-similarity and symmetry of different local structures, and the geometric relationships of different objects, thus greatly alleviating the demand for the training data.

□ **End of chapter.**

Chapter 8

List of Publications

Below lists my publications during my Ph.D. study:

[1] *SP-GAN: Sphere-Guided 3D Shape Generation and Manipulation*
Ruihui Li, Xianzhi Li, Ka-Hei Hui, and Chi-Wing Fu
ACM Transactions on Graphics (SIGGRAPH), 2021

[2] *Point Cloud Upsampling via Disentangled Refinement*
Ruihui Li, Xianzhi Li, Pheng-Ann Heng, and Chi-Wing Fu
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

[3] *PointAugment: an Auto-Augmentation Framework for Point Cloud Classification*
Ruihui Li, Xianzhi Li, Pheng-Ann Heng, and Chi-Wing Fu

IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2020. (**Oral**)

[4] *PU-GAN: a Point Cloud Upsampling Adversarial Network*
Ruihui Li, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, Pheng-Ann Heng
IEEE International Conference on Computer Vision (ICCV), 2019.

[5] *Point Set Self-Embeddeding (under review)*
Ruihui Li, Xianzhi Li, Tien-Tsin Wong, and Chi-Wing Fu
Submitted to IEEE International Conference on Computer Vision (ICCV), 2021.

[6] *DNF-Net: a Deep Normal Filtering Network for Mesh Denoising*
Xianzhi Li, **Ruihui Li**, Lei Zhu, Chi-Wing Fu, Pheng-Ann Heng
IEEE Transactions on Visualization and Computer Graphics (TVCG), 2020.

[7] *Enhancing Augmented VR Interaction via Egocentric Scene Analysis*
Yang Tian, Chi-Wing Fu, Shengdong Zhao, **Ruihui Li**, Xiao Tang, Xiaowei Hu, Pheng-Ann Heng
ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (Ubicomp), 2019.

under review:

[8] *A Rotation-Invariant Framework for Deep Point Cloud Analysis (under second review)*
Xianzhi Li, **Ruihui Li**, Chi-Wing Fu, Guangyong Chen, Daniel Cohen-Or, Pheng-Ann Heng
Submitted to IEEE Transactions on Visualization and Computer Graphics (TVCG), 2020.

[9] *Non-Local Part-Aware Point Cloud Denoising (under review)*
Chao Huang*, **Ruihui Li***, Xianzhi Li, Pheng-Ann Heng, Chi-Wing Fu (*co-first author)

Bibliography

[1] D. M. Cole and P. M. Newman, "Using Laser range data for 3D SLAM in outdoor environments," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*.

[2] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1907–1915.

[3] S. Orts-Escolano, C. Rhemann, S. Fanello, W. Chang, A. Kowdle, Y. Degtyarev, D. Kim, P. L. Davidson, S. Khamis, M. Dou *et al.*, "Holoportation: Virtual 3D teleportation in real-time," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, 2016, pp. 741–754.

[4] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

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

[6] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *arXiv preprint arXiv:1801.07829*, 2018.

[7] Y. Shen, C. Feng, Y. Yang, and D. Tian, "Mining point cloud local structures by kernel correlation and graph pooling," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4548–4557.

[8] J. Li, B. M. Chen, and G. Hee Lee, "SO-Net: Self-organizing network for point cloud analysis," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 9397–9406.

[9] P. Hermosilla, T. Ritschel, P.-P. Vázquez, À. Vinacua, and T. Ropinski, "Monte Carlo convolution for learning on non-uniformly sampled point clouds," *ACM Trans. on Graphics (SIGGRAPH Asia)*, vol. 37, no. 6, pp. 235:1–12, 2018.

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

[11] H. Zhao, L. Jiang, C.-W. Fu, and J. Jia, "PointWeb: Enhancing local neighborhood features for point cloud processing," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 5565–5573.

[12] W. Wu, Z. Qi, and L. Fuxin, "PointConv: Deep convolutional networks on 3D point clouds," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 9621–9630.

[13] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "KPConv: Flexible and deformable convolution for point clouds," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 6411–6420.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, 2012, pp. 1097–1105.

[15] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *European Conf. on Computer Vision (ECCV)*, 2014, pp. 740–755.

[16] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3D point clouds: A survey," *arXiv preprint arXiv:1912.12033*, 2019.

[17] D. Maturana and S. Scherer, "VoxNet: a 3D convolutional neural network for real-time object recognition," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015, pp. 922–928.

[18] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D shapeNets: a deep representation for volumetric shapes," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1912–1920.

[19] G. Riegler, A. O. Ulusoys, and A. Geiger, "OctNet: Learning deep 3D representations at high resolutions," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[20] A. Dai, C. R. Qi, and M. Niessner, "Shape completion using 3D-Encoder-Predictor CNNs and shape synthesis," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[21] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[22] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on riemannian manifolds," in *Proc. of the IEEE ICCV workshops*, 2015, pp. 37–45.

[23] B.-S. Hua, M.-K. Tran, and S.-K. Yeung, "Point-wise convolutional neural network," 2017.

[24] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, "SpiderCNN: Deep learning on point sets with parameterized convolutional filters," in *European Conf. on Computer Vision (ECCV)*, 2018, pp. 87–102.

[25] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz, "SPLATNet: Sparse lattice networks for point cloud processing," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2530–2539.

[26] Y. Li, R. Bu, M. Sun, and B. Chen, "PointCNN," *arXiv preprint arXiv:1801.07791*, 2018.

[27] Y. Yang, C. Feng, Y. Shen, and D. Tian, "FoldingNet: Point cloud auto-encoder via deep grid deformation," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 206–215.

[28] Y. Zhao, T. Birdal, H. Deng, and F. Tombari, "3D point capsule networks," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 1009–1018.

[29] S. Chen, C. Duan, Y. Yang, D. Li, C. Feng, and D. Tian, "Deep unsupervised learning of 3D point clouds

via graph topology inference and filtering," *arXiv preprint arXiv:1905.04571*, 2019.

[30] Z. Han, X. Wang, Y.-S. Liu, and M. Zwicker, "Multi-angle point cloud-VAE: Unsupervised feature learning for 3D point clouds from multiple angles by joint self-reconstruction and half-to-half prediction," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 10 442–10 451.

[31] J. Sauder and B. Sievers, "Self-supervised deep learning on point clouds by reconstructing space," in *Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, 2019, to appear.

[32] K. Hassani and M. Haley, "Unsupervised multi-task feature learning on point clouds," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 8160–8171.

[33] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: deep learning on point sets for 3D classification and segmentation," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 77–85.

[34] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su, "PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 909–918.

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

[36] X. Wang, S. Liu, X. Shen, C. Shen, and J. Jia, "Associatively segmenting instances and semantics in point clouds," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4096–4105.

[37] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast encoders for object detection from point clouds," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 12 697–12 705.

[38] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D object proposal generation and detection from point cloud," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 770–779.

[39] C. R. Qi, O. Litany, K. He, and L. J. Guibas, "Deep hough voting for 3D object detection in point clouds," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 9277–9286.

[40] C.-L. Li, M. Zaheer, Y. Zhang, B. Poczos, and R. Salakhutdinov, "Point cloud GAN," *arXiv preprint arXiv:1810.05795*, 2018.

[41] G. Yang, X. Huang, Z. Hao, M.-Y. Liu, S. Belongie, and B. Hariharan, "PointFlow: 3D point cloud generation with continuous normalizing flows," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 4541–4550.

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

[43] X. Chen, B. Chen, and N. J. Mitra, "Unpaired point cloud completion on real scans using adversarial training," *Int. Conf. on Learning Representations (ICLR)*, 2020.

[44] M. Sarmad, H. J. Lee, and Y. M. Kim, "RL-GAN-Net: A reinforcement learning agent controlled GAN network for real-time point cloud shape completion," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 5898–5907.

[45] Y. Aoki, H. Goforth, R. A. Srivatsan, and S. Lucey, "PointNetLK: Robust & efficient point cloud registration using PointNet," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 7163–7172.

[46] W. Lu, G. Wan, Y. Zhou, X. Fu, P. Yuan, and S. Song, "DeepVCP: An end-to-end deep neural network for 3D point cloud registration," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 12–21.

[47] Y. Wang and J. M. Solomon, "Deep closest point: Learning representations for point cloud registration," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 3523–3532.

[48] M.-J. Rakotosaona, V. La Barbera, P. Guerrero, N. J. Mitra, and M. Ovsjanikov, "PointCleanNet: Learning to denoise and remove outliers from dense point clouds," *Computer Graphics Forum (CGF)*, vol. 39, no. 1, pp. 185–203, 2020.

[49] P. Hermosilla, T. Ritschel, and T. Ropinski, "Total Denoising: Unsupervised learning of 3D point cloud cleaning," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 52–60.

[50] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, "Computing and rendering point set surfaces," *IEEE Trans. Vis. & Comp. Graphics (TVCG)*, vol. 9, no. 1, pp. 3–15, 2003.

[51] Y. Lipman, D. Cohen-Or, D. Levin, and H. Tal-Ezer, "Parameterization-free projection for geometry reconstruction," *ACM Trans. on Graphics (SIGGRAPH)*, vol. 26, no. 3, pp. 22:1–5, 2007.

[52] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, and H. R. Zhang, "Edge-aware point set resampling," *ACM*

Trans. on Graphics (TOG), vol. 32, no. 1, pp. 9:1–9:12, 2013.

[53] S. Wu, H. Huang, M. Gong, M. Zwicker, and D. Cohen-Or, "Deep points consolidation," *ACM Trans. on Graphics (SIGGRAPH Asia)*, vol. 34, no. 6, pp. 176:1–13, 2015.

[54] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "PU-Net: Point cloud upsampling network," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2790–2799.

[55] ——, "EC-Net: An edge-aware point set consolidation network," in *European Conf. on Computer Vision (ECCV)*, 2018, pp. 386–402.

[56] W. Yifan, S. Wu, H. Huang, D. Cohen-Or, and O. Sorkine-Hornung, "Patch-based progressive 3D point set upsampling," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 5958–5967.

[57] Y. Qian, J. Hou, S. Kwong, and Y. He, "PUGeo-Net: A geometry-centric network for 3D point cloud upsampling," in *European Conf. on Computer Vision (ECCV)*, 2020.

[58] G. Qian, A. Abualshour, G. Li, A. Thabet, and B. Ghanem, "PU-GCN: Point cloud upsampling using graph convolutional networks," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[59] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *Int. Conf. on Learning Representations (ICLR)*, 2018.

[60] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "CutMix: Regularization strategy to train strong classifiers with localizable features," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 6023–6032.

[61] J. Lemley, S. Bazrafkan, and P. Corcoran, "Smart augmentation learning an optimal data augmentation strategy," *IEEE Access*, vol. 5, pp. 5858–5869, 2017.

[62] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2107–2116.

[63] A. J. Ratner, H. Ehrenberg, Z. Hussain, J. Dunnmon, and C. Ré, "Learning to compose domain-specific transformations for data augmentation," in *Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, 2017, pp. 3236–3246.

[64] T. Tran, T. Pham, G. Carneiro, L. Palmer, and I. Reid, "A Bayesian data augmentation approach for learning deep models," in *Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, 2017, pp. 2797–2806.

[65] T. DeVries and G. W. Taylor, "Dataset augmentation in feature space," in *Int. Conf. on Learning Representations (ICLRW)*, 2017.

[66] B. Liu, X. Wang, M. Dixit, R. Kwitt, and N. Vasconcelos, "Feature space transfer for data augmentation," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 9090–9098.

[67] Z. Cheny, Y. Fuy, Y. Zhang, Y.-G. Jiang, X. Xue, and L. Sigal, "Multi-level semantic feature augmentation for one-shot learning," *IEEE Trans. Image Proc. (TIP)*, vol. 28, no. 9, pp. 4594–4605, 2019.

[68] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "AutoAugment: Learning augmentation strategies from data," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 113–123.

[69] S. Lim, I. Kim, T. Kim, C. Kim, and S. Kim, "Fast AutoAugment," in *Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, 2019.

[70] D. Ho, E. Liang, I. Stoica, P. Abbeel, and X. Chen, "Population Based Augmentation: Efficient learning of augmentation policy schedules," in *Int. Conf. on Machine Learning (ICML)*, 2019.

[71] Z. Tang, X. Peng, T. Li, Y. Zhu, and D. N. Metaxas, "AdaTransform: Adaptive data transformation," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 2998–3006.

[72] X. Zhang, Q. Wang, J. Zhang, and Z. Zhong, "Adversarial AutoAugment," in *Int. Conf. on Learning Representations (ICLR)*, 2020.

[73] 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 Trans. on Graphics (TOG)*, vol. 38, no. 5, pp. 146:1–12, 2019.

[74] Y. Sun, Y. Wang, Z. Liu, J. Siegel, and S. Sarma, "PointGrow: Autoregressively learned point cloud generation with self-attention," in *The IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2020, pp. 61–70.

[75] R. Klokov, E. Boyer, and J. Verbeek, "Discrete point flow networks for efficient point cloud generation," in *European Conf. on Computer Vision (ECCV)*, 2020.

[76] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," in *Int. Conf. on Learning Representations (ICLR)*, 2016.

[77] H. Kim, H. Lee, W. H. Kang, J. Y. Lee, and N. S. Kim, "SoftFlow: Probabilistic framework for normalizing flow on manifolds," in *Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, 2020.

[78] R. Cai, G. Yang, H. Averbuch-Elor, Z. Hao, S. Belongie, N. Snavely, and B. Hariharan, "Learning gradient fields for shape generation," in *European Conf. on Computer Vision (ECCV)*, 2020.

[79] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3D point clouds," in *Int. Conf. on Learning Representations (ICLR)*, 2018.

[80] S. Ramasinghe, S. Khan, N. Barnes, and S. Gould, "Spectral-GANs for high-resolution 3D point-cloud generation," in *IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 2019, pp. 8169–8176.

[81] D. W. Shu, S. W. Park, and J. Kwon, "3D point cloud generative adversarial network based on tree structured graph convolutions," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 3859–3868.

[82] R. Gal, A. Bermano, H. Zhang, and D. Cohen-Or, "MRGAN: Multi-rooted 3D shape generation with unsupervised part disentanglement," *arXiv preprint arXiv:2007.12944*, 2020.

[83] L. Hui, R. Xu, J. Xie, J. Qian, and J. Yang, "Progressive point cloud deconvolution generation network," in *European Conf. on Computer Vision (ECCV)*, 2020.

[84] M. S. Arshad and W. J. Beksi, "A progressive conditional generative adversarial network for generating dense and colored 3D point clouds," in *International Conference on 3D Vision (3DV)*, 2020.

[85] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[86] S. Song, S. P. Lichtenberg, and J. Xiao, "SUN RGB-D: A RGB-D scene understanding benchmark suite," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 567–576.

[87] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "ScanNet: Richly-annotated 3D reconstructions of indoor scenes," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[88] H. Huang, D. Li, H. R. Zhang, U. Ascher, and D. Cohen-Or, "Consolidation of unorganized point clouds for surface reconstruction," *ACM Trans. on Graphics (SIGGRAPH Asia)*, vol. 28, no. 5, pp. 176:1–176:8, 2009.

[89] I. J. Goodfellow, "On distinguishability criteria for estimating generative models," in *Int. Conf. on Learning Representations (ICLR) Workshops*, 2015.

[90] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *Int. Conf. on Learning Representations (ICLR)*, 2016.

[91] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *Int. Conf. on Machine Learning (ICML)*. PMLR, 2019, pp. 7354–7363.

[92] M. Corsini, P. Cignoni, and R. Scopigno, "Efficient and flexible sampling with blue noise properties of triangular meshes," *IEEE Trans. Vis. & Comp. Graphics (TVCG)*, vol. 18, no. 6, pp. 914–924, 2012.

[93] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, "Least squares generative adversarial networks," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2017, pp. 2794–2802.

[94] H. Fan, H. Su, and L. J. Guibas, "A point set generation network for 3D object reconstruction from a single image," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[95] "Visionair," http://www.infra-visionair.eu/, accessed: 2019-07-24.

[96] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[97] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local Nash equilibrium," in *Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, 2017, pp. 6626–6637.

[98] M. Berger, J. A. Levine, L. G. Nonato, and et al., "A benchmark for surface reconstruction," *ACM Trans. on Graphics (TOG)*, vol. 32, no. 2, p. 20, 2013.

[99] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Trans. on Graphics (TOG)*, vol. 32, no. 3, pp. 29:1–29:13, 2013.

[100] M. A. Uy, Q.-H. Pham, B.-S. Hua, T. Nguyen, and S.-K. Yeung, "Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 1588–1597.

[101] R. Li, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "PU-GAN: A point cloud upsampling adversarial net-

work," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 7203–7212.

[102] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Trans. Vis. & Comp. Graphics (TVCG)*, vol. 5, no. 4, pp. 349–359, 1999.

[103] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[104] H. Fan, H. Su, and L. J. Guibas, "A point set generation network for 3D object reconstruction from a single image," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 605–613.

[105] J. Mao, X. Wang, and H. Li, "Interpolated convolutional networks for 3D point cloud understanding," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 1578–1587.

[106] B. Graham, "Fractional max-pooling," *arXiv preprint arXiv:1412.6071*, 2014.

[107] M. Savva, F. Yu, H. Su, M. Aono, B. Chen, D. Cohen-Or, W. Deng, H. Su, S. Bai, X. Bai *et al.*, "SHREC16' track: largescale 3D shape retrieval from ShapeNet Core55," in

Proceedings of the eurographics workshop on 3D object retrieval, 2016, pp. 89–98.

[108] G.-J. Qi, "Loss-sensitive generative adversarial networks on Lipschitz densities," *arXiv preprint arXiv:1701.06264*, 2017.

[109] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NeurIPS Workshop*, 2017.

[110] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot ensembles: Train 1, get M for free," in *Int. Conf. on Learning Representations (ICLR)*, 2017.

[111] I. Goodfellow, "NeurIPS 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160*, 2016.

[112] Y. Liu, B. Fan, G. Meng, J. Lu, S. Xiang, and C. Pan, "DensePoint: Learning densely contextual representation for efficient point cloud processing," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 5239–5248.

[113] Z. Zhang, B.-S. Hua, and S.-K. Yeung, "ShellNet: Efficient point cloud convolutional neural networks using concentric shells statistics," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 1607–1616.

[114] H.-Y. Meng, L. Gao, Y.-K. Lai, and D. Manocha, "VV-Net: Voxel VAE net with group convolutions for point cloud segmentation," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 8500–8508.

[115] Z. Chen, W. Zeng, Z. Yang, L. Yu, C.-W. Fu, and H. Qu, "LassoNet: Deep Lasso-selection of 3D point clouds," *IEEE Trans. Vis. & Comp. Graphics (TVCG)*, vol. 26, no. 1, pp. 195–204, 2020.

[116] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia, "STD: Sparse-to-dense 3D object detector for point cloud," in *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 1951–1960.

[117] K. Mo, P. Guerrero, L. Yi, H. Su, P. Wonka, N. Mitra, and L. J. Guibas, "StructureNet: Hierarchical graph networks for 3D shape generation," *ACM Trans. on Graphics (SIGGRAPH Asia)*, vol. 38, no. 6, pp. 242:1–242:19, 2019.

[118] R. Wu, Y. Zhuang, K. Xu, H. Zhang, and B. Chen, "PQ-NET: A generative part Seq2Seq network for 3D shapes," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 829–838.

[119] V. Dumoulin, J. Shlens, and M. Kudlur, "A learned representation for artistic style," in *Int. Conf. on Learning Representations (ICLR)*, 2017.

[120] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4401–4410.

[121] E. Schonfeld, B. Schiele, and A. Khoreva, "A U-Net based discriminator for generative adversarial networks," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 8207–8216.

[122] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, 2014, pp. 2672–2680.

[123] A. X. Chang, T. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "ShapeNet: An information-rich 3D model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[124] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: A skinned multi-person linear model," *ACM Trans. on Graphics (SIGGRAPH Asia)*, vol. 34, no. 6, pp. 248:1–248:16, 2015.

[125] S. Zuffi, A. Kanazawa, D. Jacobs, and M. J. Black, "3D Menagerie: modeling the 3D shape and pose of animals,"

in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5524–5532.