ChartPointFlow for Topology-Aware 3D Point Cloud Generation

Takumi Kimura
Graduate School of System Informatics, Kobe University
Kobe, Japan
kimura@ai.cs.kobe-u.ac.jp

Takashi Matsubara
Graduate School of Engineering Sciences, Osaka University
Toyonaka, Japan
matsubara@sys.es.osaka-u.ac.jp

Kuniaki Uehara
Faculty of Business Administration, Osaka Gakuin University
Suita, Japan
kuniaki.uehara@ogu.ac.jp

ABSTRACT
A point cloud serves as a representation of the surface of a three-dimensional (3D) shape. Deep generative models have been adapted to model their variations typically using a map from a ball-like set of latent variables. However, previous approaches did not pay much attention to the topological structure of a point cloud, despite that a continuous map cannot express the varying numbers of holes and intersections. Moreover, a point cloud is often composed of multiple subparts, and it is also difficult to express. In this study, we propose ChartPointFlow, a flow-based generative model with multiple latent labels for 3D point clouds. Each label is assigned to points in an unsupervised manner. Then, a map conditioned on a label is assigned to a continuous subset of a point cloud, similar to a chart of a manifold. This enables our proposed model to preserve the topological structure with clear boundaries, whereas previous approaches tend to generate blurry point clouds and fail to generate holes. The experimental results demonstrate that ChartPointFlow achieves state-of-the-art performance in terms of generation and reconstruction compared with other point cloud generators. Moreover, ChartPointFlow divides an object into semantic subparts using charts, and it demonstrates superior performance in case of unsupervised segmentation.

CCS CONCEPTS
• Computing methodologies → Point-based models.

KEYWORDS
point clouds, generative model, manifold

1 INTRODUCTION
A three-dimensional (3D) point cloud, which is a set of 3D locations in a Euclidean space, has gained popularity as a representation of a geometric shape [25, 34, 35, 40, 41, 44–46, 49, 51] (see the survey [12] for more details). Specifically, the point cloud of an object’s surface is easily acquired using sensors such as LiDARs and Kinects. Point clouds can capture a much higher resolution than voxels, and can be processed using simpler manipulations than meshes. By leveraging the flexibility of deep learning, a deep generative model of point clouds enables a variety of synthesis tasks such as generation, reconstruction, and super-resolution [1, 2, 13, 18, 26, 36, 39, 43, 47]. Because it is difficult to measure the quality of a generated point cloud numerically, most studies employ flow-based generative models [6, 10, 20] or generative adversarial networks (GANs) [9]. These methods learn a map that transforms a latent distribution into an object in the data space, and then they evaluate the object without a heuristic distance.

As a representation of an object’s surface, a point cloud often has a thin, circular, or hollow structure [30]. Flow-based generative models encounter a difficulty in expressing such manifold-like structures because a bijective map that is necessary for these models does not exist between a Euclidean space and a manifold with holes, as shown in the top panel of Fig. 1. To express a point cloud X...
lying on the one-dimensional (1D) circle $S^1$, a map $\varphi$ modeled using a neural network squashes a two-dimensional (2D) ball in a latent space and stretches it to trace an arc, resulting in a discontinuity and outliers. Several existing methods address a similar issue using a flow on a manifold or a dynamic chart method [29, 38]. However, such methods are applicable only when the geometric property of the target manifold is known and fixed. This assumption does not always hold for point cloud datasets of a variety of shapes. Moreover, a point cloud is often composed of multiple subparts, some of which can be disconnected; additionally, it is difficult to express. This is true for methods based on GANs and autoencoders (AEs) as well, as long as their neural networks are continuous.

Considering these drawbacks, we propose ChartPointFlow, a generative model for 3D point clouds with latent labels. Each label is assigned to points in an unsupervised manner. Then, a map conditioned on a label is assigned to a continuous subset of a given point cloud, similar to a chart of a manifold, and a set of charts forms an atlas that covers the entire point cloud. Taking Fig. 1 as an example, ChartPointFlow with two charts, namely, $\varphi_1$ and $\varphi_2$, generates two arcs separately and concatenates them in the data space, thereby generating a continuous and hollow circle. For a more complex object, each chart is assigned to a semantic subpart, e.g., the airframe, right wing, nose, and left wing of an airplane, as shown in Fig. 2. From the perspective of the generative model, ChartPointFlow with $n$ labels provides a mixture of $n$ distributions.

Furthermore, we evaluate ChartPointFlow through its performance on synthetic datasets and ShapeNet dataset [4] of point clouds. The experiments demonstrate that ChartPointFlow preserves the topological structure in detail, whereas previous approaches tend to generate blurry point clouds and fail to generate holes. Numerical results demonstrate that ChartPointFlow outperforms other state-of-the-art point cloud generators, such as r-GAN [1], I-GAN [1], FC-GAN [26], ShapeGF [3], PointFlow [47], SoftFlow [18], AtlasNet [11], AtlasNet V2 [5], tree-GAN [39], and GCN-GAN [43]. In terms of reconstruction and unsupervised semantic segmentation, ChartPointFlow outperforms AtlasNet [11] and AtlasNet V2 [5], which are based on AEs and share the concept of charts and atlases.

2 RELATED WORK

Deep Learning on Point Clouds: A point cloud is composed of points in no particular order. PointNet takes each point separately and performs a permutation-invariant operation (max-pooling), thereby obtaining the global feature [34]. Following PointNet, many studies focused on classification and segmentation tasks [25, 34, 35, 40, 41, 44–46, 49, 51].

Likelihood-based Point Cloud Generation: One of the earliest models for point cloud generation is MR-VAE [8], which is based on a variational AE (VAE). A VAE is a probabilistic model that is implemented using two neural networks, namely a decoder that generates a sample and an encoder that performs the variational inference of the latent variable [37]. MR-VAE was trained to minimize a heuristic distance between real and generated point clouds. Zamorski et al. [50] employed an adversarial AE to regularize the latent variables. Liu et al. [27] employed a recurrent neural network to generate a point cloud step-by-step. Instead of an AE, Cai et al. [3] proposed ShapeGF, which used an implicit function defined using a neural network.

Yang et al. [47] proposed PointFlow, which is a combination of a permutation-invariant encoder and a point-wise flow-based generative model. A flow-based generative model is a neural network that forms a bijective map and obtains a likelihood using the change of variables without a heuristic distance [6, 10, 20]. Moreover, this model can accept and generate an arbitrary number of points.

Because no bijective map exists between manifolds of different topologies, a flow-based generative model tends to be destabilized when modeling zero-width structures, such as a surface. This is often the case with point clouds. Kim et al. [18] proposed SoftFlow to address this issue by adding perturbations to points at the training phase. SoftFlow emphasizes the importance of the topology, but remains inapplicable to general topological structures, such as holes, intersections, and disconnections. ChartPointFlow addresses this problem by using charts.

Likelihood-free Point Cloud Generation: Another group of models for point cloud generation involves those based on GANs. A GAN comprises a pair of neural networks, namely, a generator that outputs artificial samples and a discriminator that evaluates their similarity to real samples without a heuristic distance or an explicit likelihood [9]. r-GAN generates all the points of a point cloud simultaneously [1]. l-GAN applies a GAN to the feature vector extracted by a pretrained AE [1]. PC-GAN employs a permutation-invariant generator [26]. Spectral-GAN handles point clouds in the spectral domain [36].

Other GAN-based approaches can be regarded as recursive super-resolutions. Each model first generates a sparse point cloud, and then it adds more points to interpolate the existing ones repeatedly [2, 13, 39, 43]. Valsesia et al. [43] found that the points close to each other have similar feature vectors. Shu et al. [39] also found that each point generated at the first step may be associated with a semantic subpart of the point cloud. These results demonstrate the importance of semantic subparts. However, the above-mentioned studies do not deal with subparts explicitly.

Generative Model with Labels: For modeling samples of multiple categories, deep learning-based generative models have been extended to mixture distributions, such as conditional VAEs [21], conditional GANs [31], and conditional flow-based generative models [6, 20, 24]. The condition represents the class label that an image or object belongs to. In contrast, ChartPointFlow divides each point in a single object into a class. As shown in Fig. 1, existing generative models encounter a difficulty in expressing a single cluster if the cluster has a different topology.

AtlasNet [11] and AtlasNet V2 [5] share the concept of charts and atlases with ChartPointFlow. However, they assume to express all objects in the same category using a fixed number of fixed-size charts. This assumption is unnatural when the objects’ shapes vary widely. For example, the topology of a chair with armrests is different from that of a chair without armrests. In contrast, ChartPointFlow resizes charts and discards unnecessary charts by inferring the occurrence probability of each chart from a given object shape. Although AtlasNets aim to reconstruct point clouds, they cannot generate point clouds without modification. AtlasNets are based on ordinary neural networks, which approximate arbitrary functions.
Appendix A for more details on flow-based models. The same is true for methods based on GANs and AEs because a neural network failed in expressing holes, intersections, and disconnections. See contributions to express thin structures, stretched them to trace arcs, and Euclidean space. PointFlow and SoftFlow squashed 2D latent distribution to a connected manifold with no hole, as long as the latent space is a Euclidean space. Therefore, a generated point cloud X always lies on a connected manifold with no hole, as long as the latent space is a Euclidean space. PointFlow and SoftFlow squashed 2D latent distributions to express thin structures, stretched them to trace arcs, and failed in expressing holes, intersections, and disconnections. ChartPointFlow employs a flow-based generative model, which approximates only bijective functions [42]. Compared with AtlasNets, ChartPointFlow has an architecture that is more consistent with the definition of charts. Luo and Hu [30] also introduced a similar concept for denoising.

3 BACKGROUND

To clarify the issues with existing methods, this section provides preliminary results. We prepared synthetic datasets, namely, the circle [10], 2sines [18], four-circle [32], and double-moon [6], each of which has only one object X comprising many points \( \{ x_j \} \), as shown in Fig. 3. The leftmost column shows the datasets. The second and third columns show the generation results of PointFlow [47] and SoftFlow [18], for which we employed Glow [20] as the backbone. The generated circles, 2sines, and four-circle show the discontinuities and blurred intersections. The generated double-moons show the string-shaped artifacts. Using FFJORD as the backbone, PointFlow and SoftFlow suppressed the undesired discontinuities for circle and 2sines but not for four-circle (see the fourth and fifth columns). Moreover, they still show the string-shaped artifacts that ruin the desired disconnections. FFJORD is a flow-based generative model inspired by a differential equation, and it learns a bijective map as a vector flow [10]. Because of numerical integration, FFJORD involves significantly high computational costs. SoftFlow did not employ FFJORD for 3D point cloud generation.

A flow-based generative model always learns a continuous deformation. Therefore, a generated point cloud X always lies on a connected manifold with no hole, as long as the latent space is a Euclidean space. PointFlow and SoftFlow squashed 2D latent distributions to express thin structures, stretched them to trace arcs, and failed in expressing holes, intersections, and disconnections. See Appendix A for more details on flow-based models. The same is true for methods based on GANs and AEs because a neural network is continuous in general. This limitation is more problematic in practical tasks, as demonstrated by the results in Section 6.4. These results motivate this study.

4 FLOW-BASED MODEL WITH CHARTS

Prior to ChartPointFlow, we propose a flow-based model with charts, which is a generator of a single point cloud \( X \).

Network Structure: A point generator \( F \) is a flow-based generative model of a point \( x \in X \) conditioned on a label \( y \). The point generator \( F \) conditioned on a label \( y \) is regarded as a chart, and a set of \( n \) charts forms an atlas that covers the entire point cloud \( X \). The conditional log-likelihood of the point \( x \) is obtained using the change of variables, as follows:

\[
\log p_F(x|y) = \log p(z) + \log \left| \det \mathbf{a}^{-1}(x; y) \right|,
\]

where \( z \) denotes the latent variable \( z = F^{-1}(x; y) \), and its prior \( p(z) \) denotes the standard Gaussian distribution. One can obtain the marginal log-likelihood \( \log p_F(x) \) as the sum of all possible labels \( \log p_F(x) = \log \sum_y p_F(x|y)p(y) \). Instead, we employed a variational inference model \( q_C(y|x) \), which was implemented as a neural network called a chart predictor \( C \). The evidence lower bound (ELBO) \( \mathcal{L}_{ELBO}(F, C; x) \) is then calculated as,

\[
\mathcal{L}_{ELBO}(F, C; x) = \mathbb{E}_{q_C(y|x)} \left[ \log \frac{p_F(x|y)q_C(y|x)}{q_C(y|x)} \right] - \mathbb{H}[q_C(y|x)p(y)] + \mathbb{H}[q_C(y|x)]
\]

where \( H[q_C(y|x)p(y)] \) and \( H[q_C(y|x)] \) denote the cross-entropy and entropy, respectively. We assume that the label prior \( p(y) \) is the uniform distribution, which implies that the cross-entropy \( H[q_C(y|x)p(y)] \) has a constant value. We emphasize that the label \( y \) is inferred to maximize the ELBO in an unsupervised manner.

Training: The label \( y \) is represented by a one-hot vector, and the ELBO \( \mathcal{L}_{ELBO} \) is given by the weighted average over all possible labels. This approach requires the computational cost to be proportional to the number of labels. To avoid this issue, we employed the Gumbel-Softmax approach [17]. Specifically,

\[
\tilde{y} = \text{softmax}(\log q_C(x) + g)/\tau, \quad g \sim \text{Gumbel}(0, 1),
\]

where \( g \) denotes a vector, each of whose elements follows the Gumbel distribution \( \text{Gumbel}(0, 1) \), \( \tau \in (0, \infty) \) denotes the temperature of the softmax function, and \( q_C(x) \) denotes the vector of the label posterior \( q_C(y|x) \), i.e., \( (q_C(x))_y = q_C(y|x) = 1|x \). This approach allows us to apply the Monte Carlo sampling to the label \( y \) in a differentiable manner. One uses a sufficiently small temperature \( \tau \), draws an almost one-hot vector \( \tilde{y} \), substitutes it into the ELBO \( \mathcal{L}_{ELBO} \), and trains neural networks using gradient descent algorithms. The ELBO \( \mathcal{L}_{ELBO} \) is approximated as,

\[
\mathcal{L}_{ELBO}(F, C; x) \approx \mathcal{L}_{ELBO}(F, C; x) 
\]

where \( \mathcal{L}_{ELBO}(F, C; x) \) is the ELBO for the chart generator.

Figure 3: Point clouds generated using the proposed ChartPointFlow, PointFlow [47], and SoftFlow [18]. Color represents the chart that the point belongs to.
where the vector $\hat{y}$ is given by Eq. (3). Owing to the Gumbel-Softmax approach, we emphasize that the computational cost is constant regardless of the number of charts.

When maximizing the approximated ELBO $\tilde{L}_{ELBO}$, the entropy $H[q_C(y|x)]$ is maximized, resulting in each point belonging to all labels with the same probabilities and the charts overlapping with each other. To assign each chart to a specific connected region of a manifold, i.e., a point cloud, we introduce a regularization term $L_{MI}(x,y)$, which is based on the mutual information $I(x; y)$, as follows.

$$I(y; x) = H[q_C(y)] - H[q_C(y|x)]$$

$$= H \left[ \frac{1}{M} \sum_{x \in X} q_C(y|x) \right] - H[q_C(y|x)]$$

$$= L_{MI}(C; x).$$

The maximization of the regularization term $L_{MI}$ cancels out the maximization of the entropy $H[q_C(y|x)]$ in the ELBO $\tilde{L}(F, C; X)$, and it additionally maximizes the entropy $H[q_C(y)]$. Thus, each sample belongs to only one chart, and all charts are used with uniform probabilities.

For the i.i.d. assumption, the objective function to be maximized for the entire point cloud $X$ is defined using the sum over the points $x$, as follows.

$$\mathcal{L}(F, C; X; \lambda) = \sum_{x \in X} \left[ \tilde{L}_{ELBO} + \lambda L_{MI} \right],$$

where $\lambda$ adjusts the regularization term $L_{MI}(x,y)$.

**Experiments on Synthetic Data:** As shown in Fig. 3, we conducted preliminary experiments on 2D synthetic datasets to prove the concept of the proposed method. We employed Glow [20] as the backbone of the point generator $F$. We used $n = 4$ charts for the circle and 2sines datasets, $n = 8$ charts for the four-circle dataset, and $n = 2$ charts for the double-moon dataset. We set $\lambda$ to 1.1 and $r$ to 0.1. For other experimental settings, we followed SoftFlow [18], such as Adam optimizer [19] with a batch size of 100 for 35K iterations. Following FFJORD [10], the learning rate was set to $10^{-4}$ for Glow and to $10^{-3}$ for FFJORD. After training, each point $x$ was drawn using the point generator $F$, as follows.

$$x = F(z; y) \quad \text{for} \quad y \sim p(y) \quad \text{and} \quad z \sim p(z).$$

PointFlow [47] and SoftFlow [18] were trained under the same experimental settings. Note that the proposed method with only a single chart is the same as PointFlow.

The generated point clouds are summarized in Fig. 3. PointFlow and SoftFlow generated point clouds suffering from discontinuities, blurs, and artifacts, as mentioned in Section 3. In contrast, the proposed method generated a circle without any discontinuity, intersections free from a severe blur, and two arcs clearly separated without any artifacts, even though the backbone was Glow. Color represents the chart that the point belongs to. In the circle, 2sines, and four-circle datasets, subparts are connected smoothly and form the manifold with holes. The intersection is expressed as the intersection of the subparts. In the double-moon, each chart is assigned to one of the arcs exclusively, and thereby expresses the disconnected manifold without artifacts. These results imply that the proposed concept of charts works well for various topological structures, even with the same latent variable distribution $p(z)$.

The left panel of Fig. 4 shows the results without the regularization term $L_{MI}(x,y)$. Each label is then assigned to the entire point cloud overlapping with each other, and the model generates the discontinuity. This is because the maximization of the entropy $H[q_C(y|x)]$ results in the uniform posterior $q_C(y|x)$, and each label works similarly.

## 5 ChartPointFlow

In this section, we extend the model proposed in Section 4 and apply it to 3D point cloud datasets. We name it ChartPointFlow. Figure 5 shows a conceptual diagram of ChartPointFlow. We assume that a point cloud dataset $X$ is composed of $N$ objects $\{X_1, X_2, \ldots, X_N\}$, and each object $X_i$ is represented by a cloud of $M_i$ points $\{x_1, x_2, \ldots, x_{M_i}\}$.

**Network Structure:** The feature encoder $E$ is the same as those used in PointFlow [47] and SoftFlow [18]. The feature encoder $E$ is a permutation-invariant neural network that accepts a point cloud $X$ consisting of $M$ points and encodes it to a posterior $q_E(s_X|X)$ of a feature vector $s_X$ using the reparameterization trick [23]. The feature vector $s_X$ is considered a representation of the entire shape of the point cloud $X$. With a Gaussian prior, the reparameterization trick is known to suffer from posterior collapse, where the output $s_X$ ignores the input $X$ [22, 47]. To make the prior more expressive, the feature encoder $E$ is combined with a flow-based generative model called a prior flow $G$, which maps the feature vector $s_X$ to the latent variable $w$. The trainable prior $p_G(s_X)$ of the feature vector $s_X$ is then given by,

$$\log p_G(s_X) = \log p(w) + \int_{s_X} \log \left| \det \frac{\partial G^{-1}(s_X)}{\partial s_X} \right|,$$

where $w = G^{-1}(s_X)$, and the prior $p(w)$ is set to the standard Gaussian distribution. Thereby, the prior flow $G$ learns the distribution of point clouds.

In addition to the architectures in the previous studies, ChartPointFlow has a chart predictor $q_C(y|x, s_X)$, which is introduced in Section 4. The chart predictor $q_C(y|x, s_X)$ is conditioned on the feature vector $s_X$. It accepts a point $x \in X$ and infers the label $y$ that corresponds to the chart that the point $x$ belongs to. The condition on $s_X$ implies that different point clouds have different atlases, even in the same dataset. For example, points in the same location can be part of the engine or the airframe depending on the airplane’s width. Moreover, the posterior of the label $y$ is $q_C(y|x, s_{X}) = \mathbb{E}_{s_X} \sum_{y} q_C(y|x, s_{X})$, indicating that the size of each chart depends on point cloud $X$. A zero posterior implies that the corresponding chart is discarded. In this way, ChartPointFlow differs significantly from AtlasNets, whose charts (patches) have the same size [5, 11].
In addition, the chart generator $K$ is trained separately to estimate the label posterior $q_C(y|X)$ by maximizing the objective function

$$L_{CP}(K; X) = - \sum_{X \in \mathcal{X}} D_{KL}(p_K(y|X)||q_C(y|X)).$$.

**Usage and Tasks:** For the generation tasks, one can follow the right panel of Fig. 5. First, draw a latent variable $w$ from the prior $p(w)$ and feed it to the prior flow $G$, obtaining a feature vector $s_X = G(w)$. By feeding the feature vector $s_X$ to the chart generator $K$, the label posterior $p_K(y|s_X)$ is obtained as a categorical distribution. Repeat the following step $M$ times for $M$ points: draw a label $y_j$ from the posterior $p_K(y|s_X)$ and a latent variable $z_j$ from the prior $p(z_j)$, feed the pair to the point generator $F$, and obtain a point $x_j = F(z_j, y_j, s_X)$. The set of the obtained points is the generated point cloud $X$. Formally, $p(X) = \int_{s_X} p_K(s_X) \prod_j p_F(x_j | y_j, s_X) p_K(y_j | s_X)$.

For the reconstruction or super-resolution task, feed a given point cloud $X$ to the feature encoder $E$ and obtain a feature vector $s_X$ instead of drawing the feature vector $s_X$ from the prior flow $G$. Drawing the same number of points is called reconstruction, and adding drawn points to a given point cloud is called super-resolution.

The computational cost of ChartPointFlow is almost the same as that of the comparison methods, PointFlow [47] and SoftFlow [18], when the same backbones are used. Recall that the computational cost of the proposed method is constant regardless of the number of charts owing to the Gumbel-Softmax approach. The chart predictor $C$ is used only during the training phase. The computational cost of the chart generator $K$ is negligible because it is proportional to the number of point clouds (i.e., objects), whereas other components $E$, $F$, and $C$ require a computational cost that is proportional to the number of points in all point clouds. In previous studies, PointFlow employed FFJORD as the backbone, but SoftFlow employed Glow. We employed Glow as the backbone of ChartPointFlow. Therefore, its computational cost is at the same level as that of SoftFlow and significantly smaller than that of PointFlow.
While Chamfer distance (CD) has also been used, recent studies have pointed out that it yields misleading results [1]. CD focuses on the number of points, \(|X_1|\) and \(|X_2|\) are composed of the same number of points, \(\phi\) denotes a bijective map from the point cloud \(X_1\) to the other \(X_2\), and \(\|x - \phi(x)\|_2\) denotes the nearest neighbor of \(x\) in \(X_2\). We decayed the learning rate by quarter after every 5K epochs. We obtained \(M = 2,048\) points randomly from each object \(X\).

We set \(\tau = 0.1\) for the Gumbel-Softmax approach, \(\mu = 0.05\) and \(\lambda = 1.0\) for the regularization term \(L_M\), and searched the number \(n\) of charts from a range of \([4, 8, 12, 16, 20, 24, 28, 32]\). The architectures of the neural networks followed those of SoftFlow [18]. The detailed architectures are summarized in Appendix B.

### 6.2 Evaluation Metrics

To measure the distance between a pair of point clouds \(X_1\) and \(X_2\), we employed the earth mover’s distance (EMD) [1, 18, 47]. The EMD is the minimum of the total travel distance of points to deform a point cloud to the other. Specifically, the EMD is defined as

\[
EMD(X_1, X_2) = \min_{\phi: X_1 \rightarrow X_2} \sum_{x \in X_1} \|x - \phi(x)\|_2.
\]

where both point clouds \(X_1\) and \(X_2\) are composed of the same number of points, \(\phi\) denotes a bijective map from the point cloud \(X_1\) to the other \(X_2\), and \(\|\cdot\|_2\) denotes the Euclidean distance on \(\mathbb{R}^3\).

While Chamfer distance (CD) has also been used, recent studies have pointed out that it yields misleading results [1]. CD focuses on populated regions (e.g., a chair’s seat cushion) and ignores sparsely placed points (e.g., a chair’s mesh backrest).

To evaluate the similarity between a pair of sets \(X_1\) and \(X_2\) of point clouds, we employed the 1-nearest neighbor accuracy (1-NNA) [18, 28, 47], which aims to evaluate whether two distributions are identical in two-sample tests. 1-NNA is obtained as

\[
1-\text{NNA}(X_1, X_2) = \frac{\sum_{x_1 \in X_1} \mathbb{1}[N_{X_2}(x_1) = \min_{x_2 \in X_2} \mathbb{1}[N_{X_1}(x_2)]]}{|X_1| + |X_2|}
\]

where both sets \(X_1\) and \(X_2\) are composed of the same number of point clouds, \(N_X(x)\) denotes the nearest neighbor of \(x\) in \(X\), the distributions \(X_1\) and \(X_2\) are composed of the same number of points, \(\mathbb{1}[\cdot]\) denotes the indicator function. Roughly speaking, a 1-nearest neighbor classifier classifies a given point cloud \(X\) into \(X_1\) or \(X_2\) according to the nearest sample \(N_X\) in terms of the EMD. The closer to 50\% the accuracy of the 1-NNA is, the more similar the distributions \(X_1\) and \(X_2\) are. Previous studies also used Jensen-Shannon divergence (JSD), minimum matching distance (MMD), and coverage (COV). However, recent studies have revealed that they may give good scores to poor models [18, 47]. For example, JSD gives a good score to a model that generates an average shape without considering individual shapes [47].

### 6.3 Generation Task

For the generation task, we compared ChartPointFlow with point clouds generators, namely r-GAN [1], l-GAN [1], PC-GAN [26], ShapeGF [3], PointFlow [47], and SoftFlow [18].

For ChartPointFlow, we took the average results of 16 runs to suppress the variance due to the randomness in the generation, and summarized the results in Table 1. The results of ShapeGF were obtained using the official release code\(^1\) under the same experimental settings, and those of the other methods for comparison were obtained from [47] and [18]. The top four methods are based on GANs, ShapeGF is based on the implicit function theorem, and the others are flow-based models. One can see that ChartPointFlow outperforms the other methods in all categories. We provided

\(^1\)https://github.com/RuojinCai/ShapeGF

---

**Table 1: Generation performances. Closer to 50% is better.**

| Model        | Airplane | Chair | Car  |
|--------------|----------|-------|------|
| r-GAN [1]    | 99.51    | 99.47 | 99.86|
| l-GAN (CD) [1] | 97.28    | 85.27 | 88.07|
| l-GAN (EMD) [1] | 85.68    | 65.56 | 68.32|
| PC-GAN [26]  | 92.32    | 78.37 | 90.87|
| ShapeGF [3]  | 81.44    | 59.60 | 60.31|
| PointFlow [47] | 75.06    | 59.89 | 62.36|
| SoftFlow [18] | 69.44    | 63.51 | 64.71|

**Figure 6: Generation examples by ChartPointFlow.**
Figure 7: Generation examples nearest to the references taken from the datasets.

The results of ChartPointFlow with 28, 20, and 24 charts for the airplane, chair, and car categories, respectively. The results with different numbers of charts are summarized in Appendix C.1. ChartPointFlow achieved state-of-the-art results with 16–28 charts for all categories. The results with different numbers of charts are summarized in Appendix C.1 just for reference.

Figure 6 shows the samples generated by ChartPointFlow, each of which is composed of 10,000 points. Each protruding subpart of an object, such as the airplane’s horizontal tails, the chair’s legs, and the car’s wheels, is expressed using a different chart. The same subparts of different objects are expressed by the same charts. The chairs in Results A, C, and D do not have armrests and do not use the charts assigned to the armrests of the chairs in Results B and E (see the red arrows). ChartPointFlow assigned several charts to the chair’s seat and armrests only when needed, thereby expressing the varying topologies. Other results are summarized in Appendix C.3.

For comparison, we took reference samples from the evaluation subsets and chose the nearest samples in terms of EMD from the samples generated by each model, as shown in Fig. 7. We used pretrained models of PointFlow and SoftFlow distributed by the original authors. ChartPointFlow generated samples more similar than others, suggesting that it generated a variety of shapes.

The other GAN-based methods [36, 39, 43] used different experimental settings. Under the same experimental settings, we confirmed that ChartPointFlow outperformed these methods (see Appendix C.4).

6.4 Reconstruction Task

For the reconstruction (or super-resolution) task, we measured the EMD between a reference point cloud and a reconstructed one, and summarized the results averaged over five trials (see Table 2). In this section, we used the pretrained model of PointFlow, which was trained only on the reconstruction task, whereas ChartPointFlow and SoftFlow were trained only on the generation task. We also evaluated AtlasNet [11] and AtlasNet V2 (patch deformation (PD) and point translation (PT)) [5] with 25 patches (P25), which are specialized for the reconstruction task. Using the original codes4,5, we trained AtlasNets ourselves under the same experimental settings. We also evaluated ShapeGF [3].

ChartPointFlow outperformed all the comparison methods in all categories. The improvement from the performances of PointFlow and SoftFlow is the most significant for the chair category. This may be because the chair category shows the varying shapes of armrests and legs and the varying number of holes in the backrest, i.e., the varying topologies. Figure 8 shows that ChartPointFlow reconstructed such shapes clearly.6 Because of the same reason, AtlasNet V2 outperformed PointFlow and SoftFlow in the chair category, but not in other categories. Moreover, ChartPointFlow reconstructed even the airplane’s front wheel and the car’s mirrors. PointFlow and SoftFlow generate shapes with different topologies only for simple target domains (e.g., 2D synthetic datasets, as shown in Fig. 3), and they suffer from blurs and artifacts in practice. AtlasNet V2 reconstructed objects that are sharper than input objects; in other words, they have difficulty in expressing small subparts with accurate densities. This is because AtlasNet V2 deformed the fixed number of fixed-size 2D patches.

See Appendices C.1 and C.3 for more results.

6.5 Unsupervised Segmentation

ChartPointFlow and AtlasNets assign each point to one of the charts (or patches [5, 11]). This process can be regarded as clustering or unsupervised segmentation. We evaluated the performances of ChartPointFlow and AtlasNets on the unsupervised part segmentation task. The PartDataset of ShapeNet dataset contains labels corresponding to semantic parts for part segmentation, such as wings of an airplane [48]. In particular, each of the three used categories is divided into four parts.

| Model        | Airplane | Chair | Car |
|--------------|----------|-------|-----|
| ShapeGF [3]  | 2.55     | 5.22  | 4.63|
| AtlasNet [11]| 2.95     | 6.68  | 4.75|
| AtlasNet V2 (PD) [5] | 3.28 | 5.67  | 4.51|
| AtlasNet V2 (PT) [5] | 3.57 | 5.97  | 5.13|
| PointFlow [47]| 2.77     | 6.42  | 5.16|
| SoftFlow [18]| 2.60     | 6.60  | 5.08|
| ChartPointFlow| 2.23     | 4.62  | 3.96|

Table 2: Reconstruction errors. Smaller is better.

https://github.com/stevenygd/PointFlow
https://github.com/ANLGBOY/SoftFlow
https://github.com/ThibaultGROUEIX/AtlasNet
https://github.com/TheoDEPRELLE/AtlasNetV2

6AtlasNet V2 (PT) deals with a fixed number of points; thus, it is unavailable for reconstruction of a point cloud more dense than that used at the training phase.
After training, we fed all the unseen objects to a model to assign points to charts, and we obtained the purity (PUR) and normalized mutual information (NMI). They are defined as,

\[
\begin{align*}
\text{PUR}(Y, \hat{Y}) &= \frac{1}{|Y|} \sum_{l} \max_k \#\{y_j | y_j = k \text{ for } j \text{ such that } \hat{y}_j = l\}, \\
\text{NMI}(Y, \hat{Y}) &= \frac{2 I(Y, \hat{Y})}{H(Y) + H(\hat{Y})},
\end{align*}
\]

where \( y_j \in Y \) and \( \hat{y}_j \in \hat{Y} \) denote the ground truth label and the estimated chart of a point \( x_j \in X \), respectively. \( l \) denotes the \( l \)-th cluster estimated by a model, and \( k \) denotes the \( k \)-th ground truth label.

We evaluated ChartPointFlow and AtlasNets with 25 clusters (called charts or patches), which is the default number for AtlasNets. AtlasNets do not have a chart predictor. Instead, we performed a reconstruction task and a 1-nearest neighbor classification. Specifically, we assigned a given point to the chart that the nearest reconstructed point belongs to. ChartPointFlow outperformed AtlasNets for both criteria in all categories, except for the purity for airplane, as summarized in Table 3. See Appendix C.1 for the results with different numbers of charts. We obtained the results of the part segmentation by assigning a label to each cluster so as to maximize the purity, as shown in Fig. 9. ChartPointFlow segmented the tail wing of an airplane, the legs of a chair, and the wheels of a car more clearly than AtlasNets, which contaminated the leg part of a chair with the seat part and the backrest part. Because AtlasNets employed fixed-size patches, a patch used for a leg of a chair was used for different parts of other chairs when their legs were much smaller.

| Model                  | Airplane | Chair | Car   |
|------------------------|----------|-------|-------|
| AtlasNet [11]          | 0.22 / 0.76 | 0.23 / 0.74 | 0.11 / 0.71 |
| AtlasNet V2 (PD) [5]   | 0.25 / 0.79 | 0.24 / 0.75 | 0.13 / 0.72 |
| AtlasNet V2 (PT) [5]   | 0.27 / 0.80 | 0.24 / 0.74 | 0.17 / 0.73 |
| ChartPointFlow (proposed) | 0.30 / 0.80 | 0.35 / 0.86 | 0.19 / 0.79 |

Table 3: Segmentation performances (NMI/purity) with 25 clusters. Larger is better.

7 CONCLUSION

In this study, we proposed ChartPointFlow, which is a flow-based generative model of point clouds that employs multiple charts. Each chart is assigned to a semantic subpart of a point cloud, thereby expressing a variety of shapes with different topologies. Owing to Monte Carlo sampling, the computational cost is of the same order as that of the case without charts. The performance was evaluated using four 2D synthetic datasets and three 3D practical datasets, and the results demonstrated that ChartPointFlow generates various point clouds of various shapes with better accuracies than the comparison methods.

ACKNOWLEDGMENTS

This work was partially supported by the MIC/SCOPE #172107101, JST-CREST (JPMJCR1914), and JSPS KAKENHI (19H04172, 19K20344).
Appendix

A FLOW-BASED GENERATIVE MODEL

A flow-based generative model (or a normalizing flow) $f$ is a neural network composed of a sequence of $L$ invertible transformations $g_0 \ldots g_{L-1}$, i.e., $f = g_{L-1} \circ \cdots \circ g_0$ [6, 20]. The model $f$ maps a latent variable $z$ to a sample $x$ in the data space, i.e., $x = f(z)$. Specifically,

$$z = h_0 \circ g_1 \circ h_1 \circ g_2 \cdots \circ g_{L-1},$$

where

$$h_0 = \frac{z}{g_1^{-1} \circ \cdots \circ g_{L-1}^{-1}},$$

and $h_i$, $i = 1, 2, \ldots, L$ are deterministic functions that invert $g_i$. The prior $p(z)$ is often set to a simple distribution, such as the standard Gaussian distribution.

The flow $f$ satisfies the forward and reverse mapping properties:

$$f(z) = x$$

and

$$z = f^{-1}(x).$$

The log-likelihood of a sample $x$ is obtained using the change of variables,

$$\log p(x) = \log p(z) - \sum_{i=0}^{L-1} \log \left| \frac{\partial h_i}{\partial z} \right| = \log p(z) + \sum_{i=0}^{L-1} \log \left| \frac{\partial g_i}{\partial z} \right|,$$

where $p(z)$ denotes a prior, and $\log |\det \frac{\partial h_i}{\partial z}|$ denotes the log-absolute-determinant of the Jacobian matrix $\frac{\partial h_i}{\partial z}$.

The log-likelihood is given by

$$\log p(x) = \log p(z) - \int_{t_0}^{t_1} \log \left| \frac{\partial h_i}{\partial z} \right| \, dt.$$

where $p(z)$ denotes a prior, $\log |\det \frac{\partial h_i}{\partial z}|$ denotes the log-absolute-determinant of the Jacobian matrix $\frac{\partial h_i}{\partial z}$.

B MODEL ARCHITECTURE

ChartPointFlow is adaptable to any network architecture. In the experiments in Section 6, we employed architectures similar to those of PointFlow [26] and SoftFlow [18], as summarized below.

For the feature encoder $E$, we employed the same architecture as that used in PointFlow [47]. In particular, the former part was implemented as four 1D convolutional layers with 128-128-256-512 channels and a kernel size of 1. This architecture is equivalent to fully-connected layers applied to each point independently. The latter part was composed of a max-pooling over the points, followed by three fully-connected layers with 256-128-128 units. Applied to a set of points, the feature encoder $E$ obtains a permutation-invariant joint representation $s_X$. The hidden layer was followed by a batch normalization [15] and the ReLU function. Using the reparameterization trick, the output was regarded as the posterior $q_{\theta}(s_X|X)$ of the feature vector $s_X$.

Appendix

C ADDITIONAL RESULTS

C.1 Number of Charts

We also provide the results of the generation and reconstruction tasks with the varying number of charts in Tables A1, A2, and A3. Recall that the computational cost of the proposed method is constant regardless of the number $n$ of charts owing to the Gumbel-Softmax approach [17].
C.2 Additional Metrics

Chamfer distance (CD) has been used as a distance between two point clouds \(X_1\) and \(X_2\). Jensen-Shannon divergence (JSD), minimum matching distance (MMD), and coverage (COV) have been used to measure the similarity between two point cloud sets \(X_1\) and \(X_2\). However, previous studies pointed out that these measures may give good scores to poor models [1, 18, 47].

CD is defined as the sum of the squared distance of each point to the nearest point among the points obtained from the other point cloud. Specifically,

\[
CD(X_1, X_2) = \sum_{x \in X_1} \min_{\xi \in X_2} \|x - \xi\|_2^2 + \sum_{x \in X_2} \min_{\xi \in X_1} \|x - \xi\|_2^2. \tag{22}
\]

JSD measures the distance between two empirical distributions \(P_1\) and \(P_2\). For JSD, a canonical voxel grid was introduced, the number of points lying in each voxel was counted, and then an empirical probability distribution was obtained for each of the reference and generated sets. MMD is the distance between a point cloud in the reference set and its nearest neighbor in the generated set. COV measures the fraction of point clouds in the reference set that can be matched with at least one point cloud in the generated set.

We summarized the results evaluated using these measures in Tables A1–A6 just for reference. ChartPointFlow achieved the best scores in most criteria for the generation task, as shown in Table A4.

C.3 Additional Images

We also provide additional results for the qualitative assessment.

Figures A1–A3 summarize samples generated by ChartPointFlow. One can see that a wide variety of objects are generated, and the same chart is assigned to the same subpart across objects, such as the airplane wings, chair legs, and car wheels. For example, in Fig. A1, the charts denoted by yellow, purple, and pink colors cover the front half, rear half, and wing tip of the left wing of an airplane, respectively. The assignment is independent of the absolute position or the shape of the left wing. This is true even for a stealth aircraft, whose left wing is not separated from the main body. Therefore, we conclude that ChartPointFlow learned the fine-grained semantic information.

Figure A4 shows the point clouds obtained through the linear interpolation of the feature vector \(\phi_X\) between two point clouds. To improve the visibility, we set the number \(n\) of charts to 8. At the leftmost column in the chair category, each of the four legs is covered by a different chart. With the changing feature vector \(\phi_X\), the two legs on each side come close to each other and collide, forming a different structure. In this way, ChartPointFlow expresses a variety of shapes through a continuous deformation.

Figures A5–A7 summarize the reconstruction results of objects used for training (i.e., seen objects). Figures A8–A10 summarize the reconstruction results of objects unused for training (i.e., unseen objects). Due to the randomness of the point generator \(F\), the reconstruction results are not completely the same as the original point clouds.

Recall that, in Fig. 3, PointFlow and SoftFlow generated blurred holes and intersections in the four-circle, whereas the result of ChartPointFlow is unblurred. This tendency is true for chairs’ holes in backrests, under armrests, and formed by legs in Figs. 8, A6, and A10. Also in the 1st column of Fig. A8, ChartPointFlow generated rear engines of the airplane as hollow objects accurately, whereas PointFlow and SoftFlow generated rear engines as dense point clouds. These results show that ChartPointFlow generated varying topological structures successfully.

In Fig. 3, PointFlow and SoftFlow generated the 2sines and double-moon suffering from string-shaped artifacts. They generated similar artifacts near airplanes’ wings in the 1st and 2nd columns of Fig. A5, near chairs’ legs in the 4th column of Fig. A6, and in cars’ side mirrors in the 4th and 6th columns of Fig. A10. Conversely, ChartPointFlow did not. These results show that ChartPointFlow generated protruding small subparts successfully.

C.4 Additional Methods and Dataset

Yang et al. [47] evaluated PointFlow as well as the previous works: r-GAN, I-GAN [1], and PC-GAN [26]. Kim et al. [18] ported PointFlow’s codes to SoftFlow, and we did the same to ChartPointFlow and ShapeGF [3]. Hence, the results in Tables 1 and A4 are surely obtained under the same experimental settings.

GCN-GAN [43], tree-GAN [39], and Spectral-GAN [36] share experimental settings, which are different from those of the above-mentioned studies. These studies employed the PartDataset [48] of ShapeNet for training and evaluation, did not use 1-NNA as a metric, and did not use the car category. GCN-GAN and tree-GAN are GAN-based methods regarded as recursive super-resolutions. Each method first generates a sparse point cloud, and then it adds more points recursively. GCN-GAN assumed a graph structure among points and employed a graph convolution [39]. Tree-GAN assumed a tree structure among points [36]. Spectral-GAN is a GAN-based method that handles point clouds in the spectral domain [36]. We also trained ChartPointFlow under the same experimental settings, and summarized the results in Table A5 when available. ChartPointFlow outperformed these methods in terms of JSD, and MMD-EMD, and COV-EMD. Recall that EMD is more reliable than CD; thus, ChartPointFlow is considered superior to these methods.

The experimental settings of PCGAN [2] and PDGN [13] are unclear. Taking their descriptions at face value, these studies compare methods evaluated using Core and methods evaluated using PartDataset in one table. To avoid a confusing comparison, we omitted their results.
| Category | Number of Charts | JSD(↓) | MMD(↓) | COV(↑) | 1-NNA(%) |
|----------|-----------------|--------|--------|--------|----------|
|          |                 | CD     | EMD    | CD     | EMD      | CD     | EMD    |
| Airplane | 1               | 3.54   | 0.221  | 3.15   | 49.63    | 53.21  | 72.67  | 68.90 |
|          | 4               | 3.62   | 0.220  | 3.11   | 48.89    | 51.79  | 71.77  | 67.30 |
|          | 8               | 3.39   | 0.217  | 3.08   | 49.66    | 51.70  | 70.90  | 66.54 |
|          | 12              | 3.60   | 0.213  | 3.06   | 48.40    | 51.73  | 70.20  | 65.99 |
|          | 16              | 3.93   | 0.215  | 3.07   | 49.52    | 51.08  | 70.72  | 66.48 |
|          | 20              | 3.82   | 0.218  | 3.09   | 48.10    | 51.02  | 71.20  | 66.53 |
|          | 24              | 3.01   | 0.214  | 3.06   | 50.20    | 51.79  | 69.39  | 65.62 |
|          | 28              | 3.49   | 0.213  | 3.05   | 50.57    | 52.35  | 69.48  | 65.08 |
|          | 32              | 3.46   | 0.211  | 3.04   | 49.69    | 51.82  | 70.59  | 66.08 |
| Chair    | 1               | 1.96   | 2.50   | 8.06   | 43.04    | 45.38  | 59.75  | 63.16 |
|          | 4               | 1.96   | 2.48   | 7.90   | 43.45    | 44.30  | 59.64  | 61.79 |
|          | 8               | 1.82   | 2.50   | 7.86   | 43.85    | 44.76  | 58.76  | 60.44 |
|          | 12              | 1.89   | 2.54   | 7.87   | 44.22    | 45.50  | 58.37  | 59.96 |
|          | 16              | 1.57   | 2.48   | 7.78   | 45.37    | 46.03  | 58.04  | 59.51 |
|          | 20              | 1.83   | 2.52   | 7.84   | 45.61    | 45.85  | 57.89  | 58.31 |
|          | 24              | 1.97   | 2.53   | 7.87   | 45.05    | 45.69  | 58.20  | 59.29 |
|          | 28              | 1.97   | 2.45   | 7.75   | 43.76    | 45.78  | 58.40  | 58.94 |
|          | 32              | 1.63   | 2.44   | 7.79   | 44.23    | 45.42  | 59.52  | 60.76 |
| Car      | 1               | 0.96   | 0.95   | 5.25   | 44.98    | 47.78  | 61.86  | 61.56 |
|          | 4               | 0.93   | 0.92   | 5.17   | 46.20    | 46.86  | 60.94  | 60.48 |
|          | 8               | 0.91   | 0.90   | 5.15   | 45.42    | 46.45  | 60.04  | 60.84 |
|          | 12              | 0.93   | 0.92   | 5.14   | 44.76    | 46.31  | 59.50  | 59.76 |
|          | 16              | 0.86   | 0.91   | 5.13   | 46.41    | 48.81  | 58.13  | 58.80 |
|          | 20              | 0.83   | 0.92   | 5.14   | 45.38    | 46.89  | 59.10  | 59.65 |
|          | 24              | 0.87   | 0.94   | 5.14   | 44.83    | 47.66  | 59.42  | 58.68 |
|          | 28              | 0.90   | 0.94   | 5.12   | 44.50    | 46.06  | 60.49  | 59.67 |
|          | 32              | 0.83   | 0.89   | 5.07   | 45.81    | 48.08  | 58.96  | 58.75 |

Table A1: Generation performances with different numbers of charts. The scores are multiplied by $10^2$ for JSD and MMD-EMD, and by $10^3$ for MMD-CD. ↑ denotes that a higher score is better. ↓ denotes that a lower score is better.
Table A2: Reconstruction performance evaluated through CD (×10^4) and EMD (×10^2).

| Category  | Number of Charts | CD   | EMD   |
|-----------|------------------|------|-------|
|           | 1                | 1.18 | 2.64  |
|           | 4                | 1.13 | 2.40  |
|           | 8                | 1.13 | 2.32  |
|           | 12               | 1.14 | 2.30  |
| Airplane  | 16               | 1.09 | 2.26  |
|           | 20               | 1.08 | 2.25  |
|           | 24               | 1.07 | 2.23  |
|           | 28               | 1.12 | 2.27  |
|           | 32               | 1.14 | 2.25  |
| Chair     | 1                | 11.76| 6.92  |
|           | 4                | 10.89| 5.82  |
|           | 8                | 10.43| 5.47  |
|           | 12               | 9.40 | 4.90  |
|           | 16               | 9.04 | 4.71  |
|           | 20               | 8.76 | 4.64  |
|           | 24               | 8.78 | 4.62  |
|           | 28               | 9.47 | 4.62  |
|           | 32               | 10.31| 4.79  |
| Car       | 1                | 6.95 | 5.47  |
|           | 4                | 6.78 | 4.58  |
|           | 8                | 6.66 | 4.39  |
|           | 12               | 6.56 | 4.19  |
|           | 16               | 6.34 | 4.12  |
|           | 20               | 6.31 | 4.08  |
|           | 24               | 6.20 | 3.96  |
|           | 28               | 6.35 | 3.98  |
|           | 32               | 6.27 | 3.96  |

Table A3: Segmentation performance evaluated through NMI and purity. Larger is better.

| Category  | Number of Charts | NMI  | purity |
|-----------|------------------|------|--------|
|           | 4                | 0.29 | 0.63   |
|           | 8                | 0.33 | 0.76   |
|           | 12               | 0.33 | 0.79   |
| Airplane  | 16               | 0.31 | 0.79   |
|           | 20               | 0.31 | 0.80   |
|           | 24               | 0.30 | 0.80   |
|           | 28               | 0.30 | 0.81   |
|           | 32               | 0.29 | 0.81   |
| Chair     | 4                | 0.23 | 0.65   |
|           | 8                | 0.31 | 0.71   |
|           | 12               | 0.39 | 0.85   |
|           | 16               | 0.35 | 0.84   |
|           | 20               | 0.36 | 0.86   |
|           | 24               | 0.35 | 0.86   |
|           | 28               | 0.34 | 0.85   |
|           | 32               | 0.32 | 0.84   |
| Car       | 4                | 0.10 | 0.71   |
|           | 8                | 0.15 | 0.71   |
|           | 12               | 0.18 | 0.72   |
|           | 16               | 0.19 | 0.74   |
|           | 20               | 0.17 | 0.75   |
|           | 24               | 0.18 | 0.79   |
|           | 28               | 0.18 | 0.77   |
|           | 32               | 0.19 | 0.79   |
Table A4: Generation performances. The scores are multiplied by $10^2$ for JSD and MMD-EMD, and by $10^3$ for MMD-CD. ↑ denotes that a higher score is better. ↓ denotes that a lower score is better.

| Category   | Model          | JSD(↓) | MMD(↓) | COV(%, ↑) | 1-NN(%) |
|------------|----------------|--------|---------|-----------|---------|
| Airplane   |                |        |         |           |         |
|            | r-GAN [1]      | 7.44   | 0.261   | 5.47      | 42.72   | 18.02   | 93.58   | 99.51   |
|            | l-GAN (CD) [1] | 4.62   | 0.239   | 4.27      | 43.21   | 21.23   | 86.30   | 97.28   |
|            | l-GAN (EMD) [1]| 3.61   | 0.269   | 3.29      | 47.90   | 50.62   | 87.65   | 85.68   |
|            | PC-GAN [26]    | 4.63   | 0.287   | 3.57      | 36.46   | 40.94   | 94.35   | 92.32   |
|            | ShapeGF [3]    | 4.77   | 0.214   | 3.29      | 47.64   | 45.17   | 73.85   | 81.44   |
|            | PointFlow [47] | 4.92   | 0.217   | 3.24      | 46.91   | 46.91   | 75.68   | 75.06   |
|            | ChartPointFlow (proposed) | 3.01 | 0.214 | 3.06 | 50.20 | 51.79 | 69.39 | 65.62 |
| Chair      |                |        |         |           |         |
|            | r-GAN [1]      | 11.5   | 2.57    | 12.8      | 33.99   | 9.97    | 71.75   | 99.47   |
|            | l-GAN (CD) [1] | 4.59   | 2.46    | 8.91      | 41.39   | 25.68   | 64.43   | 85.27   |
|            | l-GAN (EMD) [1]| 2.27   | 2.61    | 7.85      | 40.79   | 41.69   | 64.73   | 65.56   |
|            | PC-GAN [26]    | 3.90   | 2.75    | 8.20      | 36.50   | 38.98   | 76.03   | 78.37   |
|            | ShapeGF [3]    | 1.75   | 2.51    | 7.82      | 48.06   | 48.28   | 56.97   | 59.60   |
|            | PointFlow [47] | 1.74   | 2.24    | 7.87      | 46.83   | 46.98   | 60.88   | 59.89   |
|            | ChartPointFlow (proposed) | 1.83 | 2.52 | 7.84 | 45.61 | 45.85 | 57.89 | 58.31 |
| Car        |                |        |         |           |         |
|            | r-GAN [1]      | 12.8   | 1.27    | 8.74      | 15.06   | 9.38    | 97.87   | 99.86   |
|            | l-GAN (CD) [1] | 4.43   | 1.55    | 6.25      | 38.64   | 18.47   | 63.07   | 88.07   |
|            | l-GAN (EMD) [1]| 2.21   | 1.48    | 5.43      | 39.20   | 39.77   | 69.74   | 68.32   |
|            | PC-GAN [26]    | 5.85   | 1.12    | 5.83      | 23.56   | 30.29   | 92.19   | 90.87   |
|            | ShapeGF [3]    | 0.98   | 0.94    | 5.22      | 47.96   | 47.27   | 59.65   | 60.31   |
|            | PointFlow [47] | 0.87   | 0.91    | 5.22      | 44.03   | 46.59   | 60.65   | 62.36   |
|            | ChartPointFlow (proposed) | 0.86 | 0.91 | 5.13 | 46.41 | 48.81 | 58.13 | 58.80 |

Table A5: Generation performances on PartDataset, ShapeNet. The scores are multiplied by $10^2$ for JSD and MMD-EMD, and by $10^3$ for MMD-CD. ↑ denotes that a higher score is better. ↓ denotes that a lower score is better.

| Category   | Model          | JSD(↓) | MMD(↓) | COV(%, ↑) |
|------------|----------------|--------|---------|-----------|
| Airplane   |                |        |         |           |
|            | GCN-GAN [43]   | 8.3    | 0.8     | 7.1       | 31       | 14      |
|            | tree-GAN [39]  | 9.7    | 0.4     | 6.8       | 61       | 20      |
|            | Spectral-GAN [36] | —   | 0.2     | 5.7       | —        |         |
|            | ChartPointFlow (proposed) | 3.14 | 0.50 | 3.86 | 44.56 | 46.77 |
| Chair      |                |        |         |           |
|            | GCN-GAN [43]   | 10.0   | 2.9     | 9.7       | 30       | 26      |
|            | tree-GAN [39]  | 11.9   | 1.6     | 10.1      | 58       | 30      |
|            | Spectral-GAN [36] | —   | 1.2     | 8.0       | —        |         |
|            | ChartPointFlow (proposed) | 1.70 | 1.45 | 6.37 | 43.63 | 43.55 |
| Category | Model                  | CD  | EMD  |
|----------|------------------------|-----|------|
|          | ShapeGF [3]            | 0.98| 2.55 |
|          | AtlasNet [11]          | 1.01| 2.95 |
| Airplane | AtlasNet V2 (PD) [5]   | 1.15| 3.28 |
|          | AtlasNet V2 (FT) [5]   | 1.01| 3.57 |
|          | PointFlow [47]         | 1.21| 2.77 |
|          | SoftFlow [18]          | 1.19| 2.60 |
|          | ChartPointFlow         | 1.07| 2.23 |
|          | ShapeGF [3]            | 6.32| 5.22 |
|          | AtlasNet [11]          | 7.38| 6.68 |
| Chair    | AtlasNet V2 (PD) [5]   | 5.72| 5.67 |
|          | AtlasNet V2 (FT) [5]   | 5.17| 5.97 |
|          | PointFlow [47]         | 10.09| 6.42 |
|          | SoftFlow [18]          | 11.04| 6.60 |
|          | ChartPointFlow         | 8.78| 4.62 |
|          | ShapeGF [3]            | 5.67| 4.63 |
|          | AtlasNet [11]          | 5.71| 4.75 |
| Car      | AtlasNet V2 (PD) [5]   | 5.31| 4.51 |
|          | AtlasNet V2 (FT) [5]   | 4.60| 5.13 |
|          | PointFlow [47]         | 6.54| 5.16 |
|          | SoftFlow [18]          | 6.82| 5.08 |
|          | ChartPointFlow         | 6.20| 3.96 |

Table A6: Reconstruction performances evaluated through CD (×10^4) and EMD (×10^2).
Figure A1: Generation examples of airplane by ChartPointFlow.
Figure A2: Generation examples of chair by ChartPointFlow.
Figure A3: Generation examples of car by ChartPointFlow.
Figure A4: Linear interpolation of the feature vector $\mathbf{s}_X$ between two point clouds.
Figure A5: Reconstruction examples of seen airplanes (i.e., super-resolution).
Figure A6: Reconstruction examples of seen chairs (i.e., super-resolution).
Figure A7: Reconstruction examples of seen cars (i.e., super-resolution).
Figure A8: Reconstruction examples of unseen airplanes.
Figure A9: Reconstruction examples of unseen chairs.
Figure A10: Reconstruction examples of unseen cars.