# A POINT CLOUD GENERATIVE MODEL BASED ON NONEQUILIBRIUM THERMODYNAMICS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

We present a probabilistic model for point cloud generation, which is critical for various 3D vision tasks such as shape completion, upsampling, synthesis and data augmentation. Inspired by the diffusion process in non-equilibrium thermodynamics, we view points in point clouds as particles in a thermodynamic system in contact with a heat bath, which diffuse from the original distribution to a noise distribution. Point cloud generation thus amounts to learning the reverse diffusion process that transforms the noise distribution to the distribution of a desired shape. Specifically, we propose to model the reverse diffusion process for point clouds as a Markov chain conditioned on certain shape latent. We derive the variational bound in closed form for training and provide implementations of the model. Experimental results demonstrate that our model achieves the state-of-the-art performance in point cloud generation and auto-encoding.

## 1 INTRODUCTION

With recent advances in depth sensing and laser scanning, point clouds have attracted increasing attention as a popular representation for modeling 3D shapes. Significant progress has been made in developing methods for point cloud analysis, such as classification and segmentation (Qi et al., 2017a;b; Wang et al., 2019). On the other hand, learning generative models for point clouds is powerful in unsupervised representation learning to characterize the data distribution, which lays the foundation for various tasks such as shape completion, upsampling, synthesis, *etc.*.

Generative models such as variational auto-encoders (VAEs), generative adversarial networks (GANs), normalizing flows, *etc.*, have shown great success in image generation (Kingma & Welling, 2013; Goodfellow et al., 2014; Chen et al., 2016; Dinh et al., 2016). However, these powerful tools cannot be directly generalized to point clouds, due to the irregular sampling patterns of points in the 3D space in contrast to regular grid structures underlying images. Hence, learning generative models for point clouds is quite challenging. Prior research has explored point cloud generation via GANs (Achlioptas et al., 2018; Valsesia et al., 2018; Shu et al., 2019), auto-regressive models (Sun et al., 2020), flow-based models (Yang et al., 2019) and so on. While remarkable progress has been made, these methods have some inherent limitations for modeling point clouds. For instance, the training procedure could be unstable for GANs due to the adversarial losses. Auto-regressive models depend on the generation order, and thus lack flexibility. In flow-based models, the constraint of invertibility might limit the expressive power of the model, and the training and inference are slow especially when the flow is continuous.

In this paper, we propose a probabilistic generative model for point clouds inspired by non-equilibrium thermodynamics, exploiting the *reverse diffusion process* to learn the point distribution. As a point cloud is composed of discrete points in the 3D space, we regard these points as particles in a non-equilibrium thermodynamic system in contact with a heat bath. Under the effect of the heat bath, the position of particles evolves stochastically in the way that they diffuse and eventually spread over the space. This process is termed the *diffusion process* that converts the initial distribution of the particles to a simple noise distribution by adding noise at each time step (Jarzynski, 1997; Sohl-Dickstein et al., 2015). Analogously, we connect the point distribution of point clouds to a noise distribution via the diffusion process. Naturally, in order to model the point distribution for point cloud generation, we consider *the reverse diffusion process*, which recovers the target point distribution from the noise distribution.

In particular, we model this reverse diffusion process as a Markov chain that converts the noise distribution into the target distribution. Our goal is to learn its transition kernel such that the Markov chain can reconstruct the desired shape. Further, as the purpose of the Markov chain is modeling the point distribution, the Markov chain alone is incapable to generate point clouds of various shapes. To this end, we introduce a shape latent as the condition for the transition kernel. In the setting of generation, the shape latent follows a prior distribution which we parameterize via normalizing flows (Chen et al., 2016; Dinh et al., 2016) for better model expressiveness. In the setting of auto-encoding, the shape latent is learned end-to-end. Finally, we formulate the training objective as maximizing the variational lower bound of the likelihood of the point cloud conditional on the shape latent, which is further formulated into tractable expressions in closed form. We apply our model to point cloud generation, auto-encoding and unsupervised representation learning, and results demonstrate that our model achieves the state-of-the-art performance on point cloud generation and auto-encoding and comparable results on unsupervised representation learning.

Our main contributions include: (1) We propose a novel probabilistic generative model for point clouds, inspired by the diffusion process in non-equilibrium thermodynamics. (2) We derive a tractable training objective from the variational lower bound of the likelihood of point clouds conditioned on some shape latent. (3) Extensive experiments show that our model achieves the state-of-the-art performance in point cloud generation and auto-encoding.

## 2 Related Works

**Point Cloud Generation** Early point cloud generation methods (Achlioptas et al., 2018; Gadelha et al., 2018) treat point clouds as $N \times 3$ matrices, where $N$ is the fixed number of points, converting the point cloud generation problem to a matrix generation problem, so that existing generative models are readily applicable. For example, Gadelha et al. (2018) apply variational auto-encoders (Kingma & Welling, 2013) to point cloud generation. Achlioptas et al. (2018) employ generative adversarial networks (Goodfellow et al., 2014) based on a pre-trained auto-encoder. The main defect of these methods is that they are restricted to generating point clouds with a fixed number of points, and lack the property of permutation invariance. FoldingNet and AtlasNet (Yang et al., 2018; Groueix et al., 2018) mitigate this issue by learning a mapping from 2D patches to the 3D space, which deforms the 2D patches into the shape of point clouds. These two methods allow generating arbitrary number of points by first sampling some points on the patches and then applying the mapping on them. In addition, the points on the patches are inherently invariant to permutation.

The above methods rely on heuristic set distances such as the Chamfer distance (CD) and the Earth Mover's distance (EMD). As pointed out in (Yang et al., 2019), CD has been shown to incorrectly favor point clouds that are overly concentrated in the mode of the marginal point distribution, and EMD is slow to compute while approximations could lead to biased gradients.

Alternatively, point clouds can be regarded as samples from a point distribution. This viewpoint inspires exploration on applying likelihood-based methods to point cloud modeling and generation. PointFlow (Yang et al., 2019) employs continous normalizing flows (Chen et al., 2018; Grathwohl et al., 2018) to model the distribution of points. PointGrow (Sun et al., 2020) is an auto-regressive model with exact likelihoods. More recently, Cai et al. (2020) proposed a score-matching energy-based model ShapeGF to model the distribution of points.

Our method also regards point clouds as samples from a distribution, but differs in the probabilistic model compared to prior works. We leverage the reverse diffusion Markov chain to model the distribution of points, achieving both simplicity and flexibility. Specifically, the training process of our model involves learning the Markov transition kernel, whose training objective has a simple function form. By contrast, GAN-based models involve complex adversarial losses, flow-based methods involve expensive ODE integration and score-matching-based methods involve second-order gradients. In addition, our model is flexible, because it does not require invertibility in contrast to flow-based models, and does not assume ordering compared to auto-regressive models.

**Diffusion Probabilistic Models** The diffusion process considered in this work is related to the diffusion probabilistic model (Sohl-Dickstein et al., 2015; Ho et al., 2020). Diffusion probabilistic models are a class of latent variable models, which also use a Markov chain to convert the noise distribution to the data distribution. Prior research on diffusion probabilistic models focuses on the unconditional generation problem for toy data and images. In this work, we focus on point

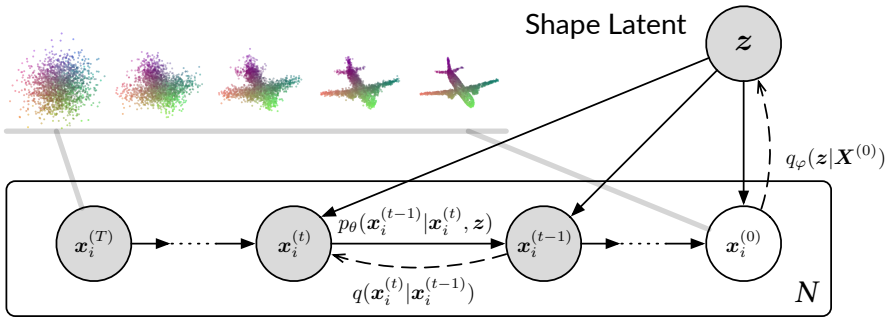


Figure 1: The directed graphical model of the diffusion process for point clouds. $N$ is the number of points in the point cloud $\boldsymbol{X}^{(0)}$.

cloud generation, which is a *conditional* generation problem, because the Markov chain considered in our work generates points of a point cloud conditioned on some shape latent. This conditional adaptation leads to significantly different training and sampling schemes compared to prior research on diffusion probabilistic models.

## 3 The Diffusion Model for Point Clouds

In this section, we first formulate the probabilistic model of both the forward and the reverse diffusion processes for point clouds. Then, we formulate the objective for training the model. The implementation of the model will be provided in the next section.

### 3.1 Formulation

We regard a point cloud $\boldsymbol{X}^{(0)} = \{\boldsymbol{x}_i^{(0)}\}_{i=1}^N$ consisting of $N$ points as a set of particles in an evolving thermodynamic system. As discussed in Section 1, each point $\boldsymbol{x}_i$ in the point cloud can be treated as being sampled independently from a point distribution, which we denote as $q(\boldsymbol{x}_i^{(0)}|\boldsymbol{z})$. Here, $\boldsymbol{z}$ is the shape latent that determines the distribution of points.

Physically, as time goes by, the points gradually diffuse into a chaotic set of points. This process is termed the *diffusion process*, which converts the original meaningful point distribution into a noise distribution. The forward diffusion process is modeled as a Markov chain (Jarzynski, 1997):

$$q(\boldsymbol{x}_i^{(1:T)}|\boldsymbol{x}_i^{(0)}) = \prod_{t=1}^{T} q(\boldsymbol{x}_i^{(t)}|\boldsymbol{x}_i^{(t-1)}), \tag{1}$$

where $q(\boldsymbol{x}_i^{(t)}|\boldsymbol{x}_i^{(t-1)})$ is the Markov diffusion kernel. The kernel adds noise to points at the previous time step and models the distribution of points at the next time step. Following the convention of (Sohl-Dickstein et al., 2015), we define the diffusion kernel as:

$$q(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t-1)}) = \mathcal{N}(\boldsymbol{x}^{(t)}|\sqrt{1-\beta_t}\boldsymbol{x}^{(t-1)}, \beta_t \boldsymbol{I}), t = 1, ..., T, \tag{2}$$

where $\beta_1 \dots \beta_T$ are variance schedule hyper-parameters that control the diffusion rate of the process.

Our goal is to generate point clouds with a meaningful shape, encoded by the latent representation $\boldsymbol{z}$. We treat the generation process as the reverse of the diffusion process, where points sampled from a simple noise distribution $p(\boldsymbol{x}_i^{(T)})$ that approximates $q(\boldsymbol{x}_i^{(T)})$ are given as the input. Then, the points are passed through the reverse Markov chain and finally form the desired shape. Unlike the forward diffusion process that simply adds noise to the points, the reverse process aims to recover the desired shape from the input noise, which requires training from data. We formulate the reverse diffusion process for generation as:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(0:T)}|\boldsymbol{z}) = p(\boldsymbol{x}^{(T)}) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(t-1)}|\boldsymbol{x}^{(t)}, \boldsymbol{z}), \tag{3}$$

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(t-1)}|\boldsymbol{x}^{(t)}, \boldsymbol{z}) = \mathcal{N}\big(\boldsymbol{x}^{(t-1)}\big|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(t)}, t, \boldsymbol{z}), \beta_t \boldsymbol{I}\big), \tag{4}$$

where $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ is the estimated mean implemented by a neural network parameterized by $\boldsymbol{\theta}$. $\boldsymbol{z}$ is the latent encoding the target shape of the point cloud. The starting distribution $p(\boldsymbol{x}_i^{(T)})$ is set to a standard normal distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Given a shape latent $\boldsymbol{z}$, we obtain the point cloud with the target shape by passing a set of points sampled from $p(\boldsymbol{x}_i^{(T)})$ through the reverse Markov chain.

For the sake of brevity, in the following sections, we use the distribution with respect to the entire point cloud $\boldsymbol{X}^{(0)}$. Since the points in a point cloud are independently sampled from a distribution, the probability of the whole point cloud is simply the product of the probability of each point:

$$q(\boldsymbol{X}^{(1:T)}|\boldsymbol{X}^0) = \prod_{i=1}^{N} q(\boldsymbol{x}_i^{(1:T)}|\boldsymbol{x}_i^{(0)}) \quad \text{and} \quad p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(0:T)}|\boldsymbol{z}) = \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(0:T)}|\boldsymbol{z}). \tag{5}$$

Having formulated the forward and reverse diffusion processes for point clouds, we will formalize the training objective of the reverse diffusion process for point cloud generation as follows.

### 3.2 Training Objective

The goal of training the reverse diffusion process is to maximize the log-likelihood of the point cloud: $\mathbb{E}[\log p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(0)})]$. However, since directly optimizing the exact log-likelihood is intractable, we instead *maximize* its variational lower bound:

$$\begin{aligned}\mathbb{E}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(0)})\big] &\geq \mathbb{E}_q\Big[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(0:T)}, \boldsymbol{z})}{q(\boldsymbol{X}^{(1:T)}, \boldsymbol{z}|\boldsymbol{X}^{(0)})}\Big] \\ &= \mathbb{E}_q\Big[\log p(\boldsymbol{X}^{(T)}) + \sum_{t=1}^{T} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z})}{q(\boldsymbol{X}^{(t)}|\boldsymbol{X}^{(t-1)})} - \log \frac{q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})}{p(\boldsymbol{z})}\Big].\end{aligned} \tag{6}$$

The above variational bound can be adapted into the training objective $L$ to be *minimized* (the detailed derivation is provided in Appendix A):

$$\begin{aligned}L(\boldsymbol{\theta}, \boldsymbol{\varphi}) &= \mathbb{E}_q\Big[\log p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(0)}|\boldsymbol{X}^{(1)}, \boldsymbol{z}) + \sum_{t=2}^{T} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z})}{q(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{X}^{(0)})} - \log \frac{q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})}{p(\boldsymbol{z})}\Big] \\ &= \mathbb{E}_q\Big[\sum_{t=2}^{T} D_{\mathrm{KL}}\big(q(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{X}^{(0)}) \| p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z})\big) \\ &\qquad - \log p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(0)}|\boldsymbol{X}^{(1)}, \boldsymbol{z}) - D_{\mathrm{KL}}\big(q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)}) \| p(\boldsymbol{z})\big)\Big].\end{aligned} \tag{7}$$

The objective $L$ allows efficient training as all the terms have closed-form expressions, which we will show later.

Since the distributions of points are independent of each other as described in Eq. (5), we further expand the training objective:

$$\begin{aligned}L(\boldsymbol{\theta}, \boldsymbol{\varphi}) = \mathbb{E}_q\Big[&\sum_{t=2}^{T}\sum_{i=1}^{N} D_{\mathrm{KL}}\big(q(\boldsymbol{x}_i^{(t-1)}|\boldsymbol{x}_i^{(t)}, \boldsymbol{x}_i^{(0)}) \| p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t-1)}|\boldsymbol{x}_i^{(t)}, \boldsymbol{z})\big) \\ &- \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(0)}|\boldsymbol{x}_i^{(1)}, \boldsymbol{z}) - D_{\mathrm{KL}}\big(q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)}) \| p(\boldsymbol{z})\big)\Big].\end{aligned} \tag{8}$$

Next, we elaborate on each term in the above objective in order to show its tractability.

The $q(\boldsymbol{x}_i^{(t-1)}|\boldsymbol{x}_i^{(t)}, \boldsymbol{x}_i^{(0)})$ in the first term can be computed with the following closed-form Gaussian (Sohl-Dickstein et al., 2015):

$$q(\boldsymbol{x}_i^{(t-1)}|\boldsymbol{x}_i^{(t)}, \boldsymbol{x}_i^{(0)}) = \mathcal{N}(\boldsymbol{x}_i^{(t-1)}|\boldsymbol{\mu}_t(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(0)}), \gamma_t \boldsymbol{I}), \tag{9}$$

where, using the notation $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$:

$$\boldsymbol{\mu}_t(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(0)}) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\boldsymbol{x}^{(0)} + \frac{\sqrt{\alpha_t}\,(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\boldsymbol{x}^{(t)} \quad \text{and} \quad \gamma_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t. \tag{10}$$

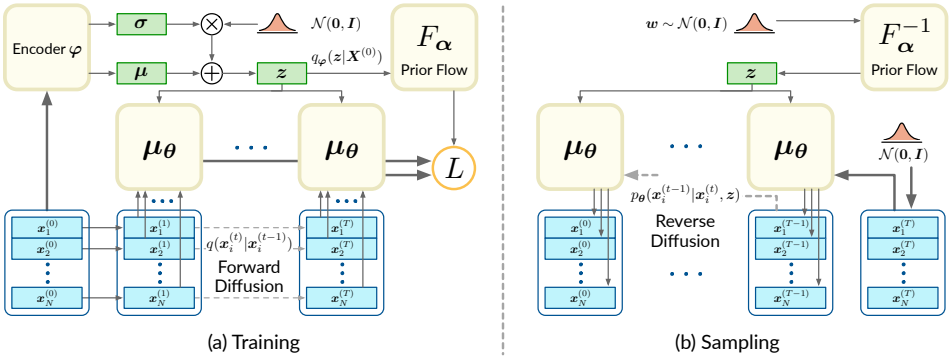


Figure 2: The illustration of the proposed model. (a) illustrates how the objective is computed during the training process. (b) illustrates the generation process.

In both the first and the second term, $\{p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t-1)}|\boldsymbol{x}_i^{(t)}, \boldsymbol{z})\}_{t=1}^{T}$ are trainable Gaussians described in Eq. (4). In the last term, $q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})$ is the approximate posterior distribution. Using the language of variational auto-encoders, $q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})$ is an encoder that encodes the input point cloud $\boldsymbol{X}^{(0)}$ into the distribution of the latent code $\boldsymbol{z}$. We assume it a Gaussian following the convention:

$$q(\boldsymbol{z}|\boldsymbol{X}^{(0)}) = \mathcal{N}\big(\boldsymbol{z}|\boldsymbol{\mu}_{\boldsymbol{\varphi}}(\boldsymbol{X}^{(0)}), \boldsymbol{\Sigma}_{\boldsymbol{\varphi}}(\boldsymbol{X}^{(0)})\big). \tag{11}$$

The last term $p(\boldsymbol{z})$ is the prior distribution. The most common choice of $p(\boldsymbol{z})$ is the isotropic Gaussian $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. In addition to a fixed distribution, the prior can be a trainable parametric distribution, which is more flexible. For example, normalizing flows (Chen et al., 2016; Dinh et al., 2016) can be employed to parameterize the prior distribution.

Consequently, all the terms in Eq. (8) can be computed with closed-form expressions, which enables efficient training. Further details about the derivation and optimization are provided in Appendix A and Appendix B.

## 4 Model Implementations

The general training objective in Eq. (8) lays the foundation for the formulation of specific point cloud tasks. Next, we adapt the training objective to point cloud generation and point cloud auto-encoding, respectively.

### 4.1 Point Cloud Generator

Leveraging on the model in Section 3, we propose a probabilistic model for point cloud generation by employing normalizing flows to parameterize the prior distribution $p(\boldsymbol{z})$, which makes the model more flexible (Rezende & Mohamed, 2015; Chen et al., 2016).

Specifically, we use a stack of affine coupling layers (Dinh et al., 2016) as the normalizing flow (the detail of affine coupling layers is provided in Appendix C). In essence, the affine coupling layers provide a trainable bijector $F_{\boldsymbol{\alpha}}$ that maps an isotropic Gaussian to a complex distribution. Since the mapping is bijective, the exact probability of the target distribution can be computed by the change-of-variable formula:

$$p(\boldsymbol{z}) = p_{\boldsymbol{w}}(\boldsymbol{w}) \cdot \left| \det \frac{\partial F_{\boldsymbol{\alpha}}}{\partial \boldsymbol{w}} \right|^{-1} \quad \text{where} \quad \boldsymbol{w} = F_{\boldsymbol{\alpha}}^{-1}(\boldsymbol{z}), \tag{12}$$

where $p(\boldsymbol{z})$ is the prior distribution in the model, $F_{\boldsymbol{\alpha}}$ is the trainable bijector implemented by the affine coupling layers, and $p_{\boldsymbol{w}}(\boldsymbol{w})$ is the isotropic Gaussian $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

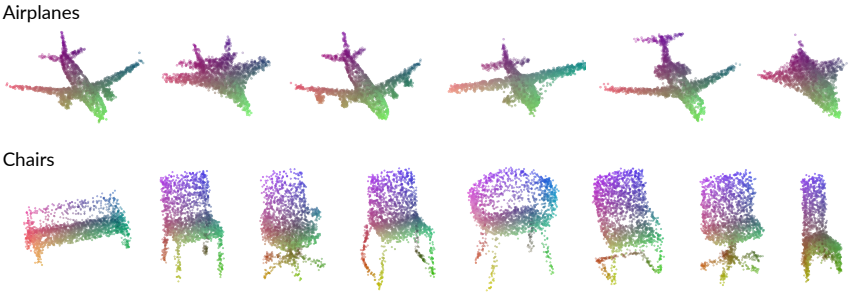


Figure 3: Visualization of our generated samples.

Substituting Eq. (12) into Eq. (8), the training objective for the generative model is:

$$L_G(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{\alpha}) = \mathbb{E}_q \Big[ \sum_{t=2}^{T} \sum_{i=1}^{N} D_{\mathrm{KL}} \big( q(\boldsymbol{x}_i^{(t-1)} | \boldsymbol{x}_i^{(t)}, \boldsymbol{x}_i^{(0)}) \| p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t-1)} | \boldsymbol{x}_i^{(t)}, \boldsymbol{z}) \big) - \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(0)} | \boldsymbol{x}_i^{(1)}, \boldsymbol{z}) - D_{\mathrm{KL}} \Big( q_{\boldsymbol{\varphi}}(\boldsymbol{z} | \boldsymbol{X}^{(0)}) \Big\| p_{\boldsymbol{w}}(\boldsymbol{w}) \cdot \Big| \det \frac{\partial F_{\boldsymbol{\alpha}}}{\partial \boldsymbol{w}} \Big|^{-1} \Big) \Big]. \tag{13}$$

Specifically, we adopt PointNet (Qi et al., 2017a) as the architecture for $\boldsymbol{\mu}_{\boldsymbol{\varphi}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\varphi}}$ of the encoder $q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})$, and employ a modified MLP as the function approximator for the mean $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ of the kernel $p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t-1)}|\boldsymbol{x}_i^{(t)}, \boldsymbol{z})$. The detail of the architecture is presented in Appendix D.

To sample a point cloud, we first draw $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and pass it through $F_{\boldsymbol{\alpha}}$ to acquire the shape latent $\boldsymbol{z}$. Then, with the shape latent $\boldsymbol{z}$, we sample some points $\{\boldsymbol{x}_i^{(T)}\}$ from the noise distribution $p(\boldsymbol{x}^{(T)})$ and pass the points through the reverse Markov chain $p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(0:T)}|\boldsymbol{z})$ defined in Eq. (3) to generate the point cloud $\boldsymbol{X}^{(0)} = \{\boldsymbol{x}_i^{(0)}\}_{i=1}^N$.

### 4.2 Point Cloud Auto-Encoder

Also, we implement a point cloud auto-encoder based on the probabilistic model in Section 3. We employ the PointNet as the representation encoder, denoted as $E_{\boldsymbol{\varphi}}(\boldsymbol{X}^{(0)})$ with parameters $\boldsymbol{\varphi}$, and leverage the reverse diffusion process presented in Section 3.1 for decoding, conditioned on the latent code produced by the encoder.

Leveraging on Eq. (8), we train the auto-encoder by minimizing the following adapted objective:

$$L(\boldsymbol{\theta}, \boldsymbol{\varphi}) = \mathbb{E}_q \Big[ \sum_{t=2}^{T} \sum_{i=1}^{N} D_{\mathrm{KL}} \big( q(\boldsymbol{x}_i^{(t-1)} | \boldsymbol{x}_i^{(t)}, \boldsymbol{x}_i^{(0)}) \| p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t-1)} | \boldsymbol{x}_i^{(t)}, E_{\boldsymbol{\varphi}}(\boldsymbol{X}^{(0)})) \big) - \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(0)} | \boldsymbol{x}_i^{(1)}, E_{\boldsymbol{\varphi}}(\boldsymbol{X}^{(0)})) \Big]. \tag{14}$$

To decode a point cloud that is encoded as the latent code $\boldsymbol{z}$, we sample some points $\{\boldsymbol{x}_i^{(T)}\}$ from the noise distribution $p(\boldsymbol{x}_i^{(T)})$ and pass the points through the reverse Markov chain $p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(0:T)}|\boldsymbol{z})$ defined in Eq. (3) to acquire the reconstructed point cloud $\boldsymbol{X}^{(0)} = \{\boldsymbol{x}_i^{(0)}\}_{i=1}^N$.

## 5 Experiments

In this section, we evaluate our model's performance on three tasks: point cloud generation, auto-encoding, and unsupervised representation learning.

Table 1: Comparison of point cloud generation performance.

| | | MMD (↓) | | COV (%, ↑) | | 1-NNA (%, ↓) | | JSD (↓) |
|---|---|---|---|---|---|---|---|---|
| Shape | Model | CD | EMD | CD | EMD | CD | EMD | - |
| Airplane | PC-GAN | 3.819 | 1.810 | 42.17 | 13.84 | 77.59 | 98.52 | 6.188 |
| | GCN-GAN | 4.713 | 1.650 | 39.04 | 18.62 | 89.13 | 98.60 | 6.669 |
| | TreeGAN | 4.323 | 1.953 | 39.37 | 8.40 | 83.86 | 99.67 | 15.646 |
| | PointFlow | 3.688 | 1.090 | 44.98 | 44.65 | 66.39 | **69.36** | 1.536 |
| | ShapeGF | 3.727 | **1.061** | 46.79 | 44.98 | **62.69** | 71.91 | 1.314 |
| | Ours | **3.276** | **1.061** | **48.71** | **45.47** | 64.83 | 75.12 | **1.067** |
| Chair | PC-GAN | 13.436 | 3.104 | 46.23 | 22.14 | 69.67 | 100.00 | 6.649 |
| | GCN-GAN | 15.354 | 2.213 | 39.84 | 35.09 | 77.86 | 95.80 | 21.708 |
| | TreeGAN | 14.936 | 3.613 | 38.02 | 6.77 | 74.92 | 100.00 | 13.282 |
| | PointFlow | 13.631 | 1.856 | 41.86 | 43.38 | 66.13 | 68.40 | 12.474 |
| | ShapeGF | 13.175 | 1.785 | 48.53 | 46.71 | **56.17** | **62.69** | **5.996** |
| | Ours | **12.276** | **1.784** | **48.94** | **47.52** | 60.11 | 69.06 | 7.797 |

### 5.1 Experimental Setup

**Datasets** For generation and auto-encoding tasks, we employ the ShapeNet dataset (Chang et al., 2015) containing 51,127 shapes from 55 categories. The dataset is randomly split into training, testing and validation sets by the ratio 80%, 15%, 5% respectively. For the representation learning task, we use the training split of ShapeNet to train an encoder. Then we adopt ModelNet10 and ModelNet40 (Wu et al., 2015) to evaluate the representations learned by the encoder. We sample 2048 points from each of the shape to acquire the point clouds and normalize each of them to zero mean and unit variance.

**Evaluation Metrics** Following prior works, we use the Chamfer Distance (CD) and the Earth Mover's Distance (EMD) to evaluate the reconstruction quality of the point clouds (Achlioptas et al., 2018). To evaluate the generation quality, we employ the minimum matching distance (MMD), the coverage score (COV), 1-NN classifier accuracy (1-NNA) and the Jenson-Shannon divergence (JSD) (Yang et al., 2019). The MMD score measures the fidelity of the generated samples and the COV score detects mode-collapse. The 1-NNA score is computed by testing the generated samples and the reference samples by a 1-NN classifier. If the performance of the classifier is close to random guess, *i.e.*, the accuracy is close to 50%, the quality of generated samples can be considered better. The JSD score measures the similarity between the point distributions of the generated set and the reference set.

**Architectures** In order to make fair comparisons, the encoder architecture of our model is identical to that of the PC-GAN, PointFlow and ShapeGF. Other details about the architecture of our model are discussed in Appendix D.

### 5.2 Point Cloud Generation

We quantitatively compare our method with the following state-of-the-art generative models: PC-GAN (Achlioptas et al., 2018), GCN-GAN (Valsesia et al., 2018), TreeGAN (Shu et al., 2019), PointFlow (Yang et al., 2019) and ShapeGF (Cai et al., 2020). Following the convention of prior works, we train each model using point clouds from two categories in ShapeNet: *airplane* and *chair*. Each of the baseline models is trained and tested using the setup reported in their papers. Following ShapeGF (Cai et al., 2020), during the evaluation, we normalize the point clouds in both the generated set and the reference set into a bounding box of $[-1, 1]^3$, so that the metrics focus on the shape of point clouds but not the scale. We evaluate the point clouds generated by the models using the metrics in Section 5.1 and summarize the results in Table 1. Our model outperforms the state-of-the-art methods under most evaluation metrics, and achieves comparable performance to PointFlow and ShapeGF under the 1-NNA score. We also visualize some generated point cloud samples from our method in Fig. 3 and present more visualizations in Appendix E.

Table 2: Comparison of point cloud auto-encoding performance. Atlas (S1) and Atlas (P25) denote 1-sphere and 25-square variants of AtlasNet respectively.

| Dataset | Metric | Atlas (S1) | Altas (P25) | PointFlow | ShapeGF | Ours | Oracle |
|---|---|---|---|---|---|---|---|
| Airplane | CD | 2.000 | **1.795** | 2.420 | 2.102 | 2.118 | 1.016 |
| | EMD | 4.311 | 4.366 | 3.311 | 3.508 | **2.876** | 2.141 |
| Car | CD | 6.906 | 6.503 | 5.828 | **5.468** | 5.493 | 3.917 |
| | EMD | 5.617 | 5.408 | 4.390 | 4.489 | **3.937** | 3.246 |
| Chair | CD | 5.479 | **4.980** | 6.795 | 5.146 | 5.677 | 3.221 |
| | EMD | 5.550 | 5.282 | 5.008 | 4.784 | **4.153** | 3.281 |
| ShapeNet | CD | 5.873 | 5.420 | 7.550 | 5.725 | **5.252** | 3.074 |
| | EMD | 5.457 | 5.599 | 5.172 | 5.049 | **3.783** | 3.112 |

Table 3: Comparison of representation learning in linear SVM classification accuracy.

| Dataset | AtlasNet | PC-GAN(CD) | PC-GAN(EMD) | PointFlow | ShapeGF | Ours |
|---|---|---|---|---|---|---|
| ModelNet-10 | 91.9 | **95.4** | **95.4** | 93.7 | 90.2 | 94.2 |
| ModelNet-40 | 86.6 | 84.5 | 84.0 | 86.8 | 84.6 | **87.6** |

### 5.3 Point Cloud Auto-Encoding

We evaluate the reconstruction quality of the proposed auto-encoder, with comparisons against state-of-the-art point cloud auto-encoders: AtlasNet (Groueix et al., 2018), PointFlow (Yang et al., 2019) and ShapeGF (Cai et al., 2020). Four datasets are used in the evaluation, which include three categories in ShapeNet: *airplane*, *car*, *chair* and the whole ShapeNet. We also report the lower bound "oracle" of the reconstruction errors. This bound is obtained by computing the distance between two different point clouds with the same number of points and the identical shape. As shown in Table 2, our method outperforms other methods when measured by EMD, and greatly pushes closer towards the lower bounded "oracle" performance. The CD score of our method is comparable to others. Notably, when trained and tested on the whole ShapeNet dataset, our model outperforms others in both CD and EMD, which suggests that our model has higher capacity to encode different shapes.

### 5.4 Unsupervised Representation Learning

Further, we evaluate the representation learned by our auto-encoder. Firstly, we train an auto-encoder with the whole ShapeNet dataset. During the training, we augment point clouds by applying random rotations along the gravity axis, which follows previous works. Then, we learn the feature representations of point clouds in ModelNet-10 and ModelNet-40 using the trained encoder, and train a linear SVM using the codes of point clouds in the training split and their categories. Finally, we test the SVM using the testing split and report the accuracy in Table 3. We run the official code of AtlasNet and ShapeGF to obtain the results, since the results are not provided in their papers. For PC-GAN and PointFlow, we use the results reported in the papers. The performance of our encoder is comparable to related state-of-the-art generative models.

## 6 Conclusions

We propose a novel probabilistic generative model for point clouds, taking inspiration from the diffusion process in non-equilibrium thermodynamics. We model the reverse diffusion process for point cloud generation as a Markov chain conditioned on certain shape latent, and derive a tractable training objective from the variational bound of the likelihood of point clouds. Experimental results demonstrate that the proposed model achieves the state-of-the-art performance in point cloud generation and auto-encoding.

## REFERENCES

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pp. 40–49. PMLR, 2018.

Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. *arXiv preprint arXiv:2008.06520*, 2020.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Matheus Gadelha, Rui Wang, and Subhransu Maji. Multiresolution tree networks for 3d point cloud processing. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 103–118, 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 216–224, 2018.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239*, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Christopher Jarzynski. Equilibrium free-energy differences from nonequilibrium measurements: A master-equation approach. *Physical Review E*, 56(5):5018, 1997.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017a.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pp. 5099–5108, 2017b.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pp. 1530–1538, 2015.

Dong Wook Shu, Sung Woo Park, and Junseok Kwon. 3d point cloud generative adversarial network based on tree structured graph convolutions. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3859–3868, 2019.

Jascha Sohl-Dickstein, Eric A Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *arXiv preprint arXiv:1503.03585*, 2015.

Yongbin Sun, Yue Wang, Ziwei Liu, Joshua Siegel, and Sanjay Sarma. Pointgrow: Autoregressively learned point cloud generation with self-attention. In *The IEEE Winter Conference on Applications of Computer Vision*, pp. 61–70, 2020.

Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Learning localized generative models for 3d point clouds via graph convolution. In *International conference on learning representations*, 2018.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019.

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4541–4550, 2019.

Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 206–215, 2018.

# A DETAILED DERIVATIONS

We present the detailed derivations of the training objective in Eq. 7.

$$
\begin{aligned}
\mathbb{E}_{q_{\text{data}}}\Big[\log p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(0)})\Big] &= \int q_{\text{data}}(\boldsymbol{X}^{(0)})\left[\log \int p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(0:T)}, \boldsymbol{z})\,\mathrm{d}\,\boldsymbol{X}^{(1:T)}\,\mathrm{d}\,\boldsymbol{z}\right]\mathrm{d}\,\boldsymbol{X}^{(0)} \\
&\geq \log \int q_{\text{data}}(\boldsymbol{X}^{(0)}) p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(0:T)}, \boldsymbol{z})\,\mathrm{d}\,\boldsymbol{X}^{(1:T)}\,\mathrm{d}\,\boldsymbol{z}\,\mathrm{d}\,\boldsymbol{X}^{(0)} \quad \text{(Jensen's inequality)} \\
&\geq \int q_{\text{data}}(\boldsymbol{X}^{(0)}) q(\boldsymbol{X}^{(1:T)}, \boldsymbol{z}|\boldsymbol{X}^{(0)}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(0:T)}, \boldsymbol{z})}{q(\boldsymbol{X}^{(1:T)}, \boldsymbol{z}|\boldsymbol{X}^{(0)})}\,\mathrm{d}\,\boldsymbol{X}^{(1:T)}\,\mathrm{d}\,\boldsymbol{z}\,\mathrm{d}\,\boldsymbol{X}^{(0)} \quad \text{(ELBO)}
\end{aligned}
$$

Note that $\boldsymbol{X}^{(1:T)}$ and $\boldsymbol{z}$ are conditionally independent on $\boldsymbol{X}^{(0)}$,

$$
\begin{aligned}
&= \int q_{\text{data}}(\boldsymbol{X}^{(0)}) q(\boldsymbol{X}^{(1:T)}|\boldsymbol{X}^{(0)}) q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})\Big[\log p(\boldsymbol{X}^{(T)}) + \log p(\boldsymbol{z}) \\
&\quad + \sum_{t=1}^{T} \log p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z}) - \log q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)}) \\
&\quad - \sum_{t}^{T} \log q(\boldsymbol{X}^{(t)}|\boldsymbol{X}^{(t-1)})\Big]\,\mathrm{d}\,\boldsymbol{X}^{(0:T)}\,\mathrm{d}\,\boldsymbol{z} \\
&= \int q_{\text{data}}(\boldsymbol{X}^{(0)}) q(\boldsymbol{X}^{(1:T)}|\boldsymbol{X}^{(0)}) q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})\Big[\log p(\boldsymbol{X}^{(T)}) + \log p(\boldsymbol{z}) \\
&\quad + \sum_{t=1}^{T} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z})}{q(\boldsymbol{X}^{(t)}|\boldsymbol{X}^{(t-1)})} - \log q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})\Big]\,\mathrm{d}\,\boldsymbol{X}^{(0:T)}\,\mathrm{d}\,\boldsymbol{z}
\end{aligned}
$$

Since $q(\boldsymbol{X}^{(t)}|\boldsymbol{X}^{(t-1)})$ is intractable, we rewrite it using Bayes' rule,

$$
\begin{aligned}
&= \int q_{\text{data}}(\boldsymbol{X}^{(0)}) q(\boldsymbol{X}^{(1:T)}|\boldsymbol{X}^{(0)}) q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})\Big[\log p(\boldsymbol{X}^{(T)}) + \log p(\boldsymbol{z}) \\
&\quad + \sum_{t=2}^{T} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z})}{q(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{X}^{(0)})} \cdot \frac{q(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(0)})}{\boldsymbol{X}^{(t)}|\boldsymbol{X}^{(0)}} \\
&\quad + \log \frac{p(\boldsymbol{X}^{(0)}|\boldsymbol{X}^{(1)}, \boldsymbol{z})}{q(\boldsymbol{X}^{(1)}|\boldsymbol{X}^{(0)})} - \log q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})\Big]\,\mathrm{d}\,\boldsymbol{X}^{(0:T)}\,\mathrm{d}\,\boldsymbol{z} \\
&= \int q_{\text{data}}(\boldsymbol{X}^{(0)}) q(\boldsymbol{X}^{(1:T)}|\boldsymbol{X}^{(0)}) q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})\Big[\log \frac{p(\boldsymbol{X}^{(T)})}{q(\boldsymbol{X}^{(T)}|\boldsymbol{X}^{(0)})} \\
&\quad + \sum_{t=2}^{T} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z})}{q(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{X}^{(0)})} + \log p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(0)}|\boldsymbol{X}^{(1)}, \boldsymbol{z}) \\
&\quad + \log p(\boldsymbol{z}) - \log q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})\Big]\,\mathrm{d}\,\boldsymbol{X}^{(0:T)}\,\mathrm{d}\,\boldsymbol{z} \\
&= \int q_{\boldsymbol{\varphi}}(\boldsymbol{X}^{(0:T)}, \boldsymbol{z})\Big[\log \frac{p(\boldsymbol{X}^{(T)})}{q(\boldsymbol{X}^{(T)}|\boldsymbol{X}^{(0)})} + \sum_{t=2}^{T} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z})}{q(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{X}^{(0)})} \\
&\quad + \log p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(0)}|\boldsymbol{X}^{(1)}, \boldsymbol{z}) + \log \frac{p(\boldsymbol{z})}{q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})}\Big]\,\mathrm{d}\,\boldsymbol{X}^{(0:T)}\,\mathrm{d}\,\boldsymbol{z}
\end{aligned}
$$

On the right hand side, all the terms except $\log p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(0)}|\boldsymbol{X}^{(1)}, \boldsymbol{z})$ can be rewritten into the form of the KL divergence. We show how to do it on one of the terms. For other terms, it is similar.

$$
\begin{aligned}
&\int q_{\boldsymbol{\varphi}}(\boldsymbol{X}^{(0:T)}, \boldsymbol{z}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z})}{q(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{X}^{(0)})} \,\mathrm{d}\, \boldsymbol{X}^{(0:T)} \,\mathrm{d}\, \boldsymbol{z} \\
&\quad = \int q_{\boldsymbol{\varphi}}(\boldsymbol{X}^{(0)}, \boldsymbol{X}^{(t-1)}, \boldsymbol{X}^{(t)}, \boldsymbol{z}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z})}{q(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{X}^{(0)})} \,\mathrm{d}\, \boldsymbol{X}^{(0,t-1,t)} \,\mathrm{d}\, \boldsymbol{z} \\
&\quad = \int q(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(0)}, \boldsymbol{X}^{(t)}) q_{\boldsymbol{\varphi}}(\boldsymbol{X}^{(0)}, \boldsymbol{X}^{(t)}, \boldsymbol{z}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z})}{q(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{X}^{(0)})} \,\mathrm{d}\, \boldsymbol{X}^{(0,t-1,t)} \,\mathrm{d}\, \boldsymbol{z} \\
&\quad = -\int q_{\boldsymbol{\varphi}}(\boldsymbol{X}^{(0)}, \boldsymbol{X}^{(t)}, \boldsymbol{z}) D_{\mathrm{KL}}\left(q(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{X}^{(0)}) \middle\| p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z})\right) \mathrm{d}\, \boldsymbol{X}^{(0,t)} \,\mathrm{d}\, \boldsymbol{z} \\
&\quad = -\mathbb{E}_{\boldsymbol{X}^{(0)}, \boldsymbol{X}^{(t)}, \boldsymbol{z} \sim q_{\boldsymbol{\varphi}}}\left[D_{\mathrm{KL}}\left(q(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{X}^{(0)}) \middle\| p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z})\right)\right].
\end{aligned}
$$

Next, notice that $\log \frac{p(\boldsymbol{X}^{(T)})}{q(\boldsymbol{X}^{(T)}|\boldsymbol{X}^{(0)})}$ has no trainable parameters, so we can ignore it in the training objective. Finally, by negating the variational bound and decomposing the distributions, we obtain the training objective in Eq. 7 as follows:

$$
\begin{aligned}
L(\boldsymbol{\theta}, \boldsymbol{\varphi}) = \mathbb{E}_q\Big[ \sum_{t=2}^{T} \sum_{i=1}^{N} \underbrace{D_{\mathrm{KL}}\big(q(\boldsymbol{x}_i^{(t-1)}|\boldsymbol{x}_i^{(t)}, \boldsymbol{x}_i^{(0)}) \| p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t-1)}|\boldsymbol{x}_i^{(t)}, \boldsymbol{z})\big)}_{L_i^{(t-1)}} \\
- \sum_{i=1}^{N} \underbrace{\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(0)}|\boldsymbol{x}_i^{(1)}, \boldsymbol{z})}_{L_i^{(0)}} - \underbrace{D_{\mathrm{KL}}\big(q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)}) \| p(\boldsymbol{z})\big)}_{L_{\boldsymbol{z}}} \Big].
\end{aligned} \tag{15}
$$

## B Training Algorithm

The training algorithm can be formulated according to the generator's training objective in Eq. 13:

**Algorithm 1** Training

1: **repeat**
2: Sample $\boldsymbol{X}^{(0)} \sim q_{\mathrm{data}}(\boldsymbol{X}^{(0)})$
3: Sample $\boldsymbol{z} \sim q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})$
4: **for** $t = 1 \dots T$ **do**
5: Sample $\boldsymbol{X}^{(t)} \sim q(\boldsymbol{X}^{(t)}|\boldsymbol{X}^{(t-1)})$
6: **end for**
7: Compute $\nabla L_G(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{\alpha})$ using samples $\boldsymbol{X}^{(0:T)}$ and $\boldsymbol{z}$; Then perform gradient descent.
8: **until** converged

**Algorithm 2** Sampling

1: Sample $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
2: $\boldsymbol{z} \leftarrow F_{\boldsymbol{\alpha}}(\boldsymbol{w})$
3: $\boldsymbol{X}^{(T)} \leftarrow \{\boldsymbol{x}_i^{(T)}\} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
4: **for** $t = T \dots 1$ **do**
5: Sample $\boldsymbol{X}^{(t-1)} \sim p_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t-1)}|\boldsymbol{X}^{(t)}, \boldsymbol{z})$
6: **end for**
7: **return** $\boldsymbol{X}^{(0)}$

Ho et al. (2020) proposed a simplified training algorithm. We also adapt the simplified algorithm for our model. Before formulating the simplified algorithm, we should further analyse $L_i^{(t-1)}$. Since both $q(\boldsymbol{x}_i^{(t-1)}|\boldsymbol{x}_i^{(t)}, \boldsymbol{x}_i^{(0)})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t-1)}|\boldsymbol{x}_i^{(t)}, \boldsymbol{z})$ are Gaussians (Eq. 4, Eq. 10), the term $L_i^{(t-1)}$ can be expanded as:

$$
L_i^{(t-1)} = \mathbb{E}_{\boldsymbol{x}_i^{(0)}, \boldsymbol{x}_i^{(t)}, \boldsymbol{z}}\left[\frac{1}{2\beta_t}\left\| \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{x}_i^{(0)} + \frac{\sqrt{\alpha_t}\,(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{x}_i^{(t)} - \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t)}, t, \boldsymbol{z})\right\|^2\right] + C. \tag{16}
$$

Evaluating $L_i^{(t-1)}$ requires sampling $\boldsymbol{x}_i^{(t)}$ from $q(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(0)})$. In principle, it can be done by sampling iteratively through the Markov chain. Fortunately, Ho et al. (2020) showed $q(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(0)})$ is a Gaussian, thus allowing us to sample $\boldsymbol{x}^{(t)}$ efficiently without iterative sampling:

$$
q(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(0)}) = \mathcal{N}(\boldsymbol{x}^{(t)}|\sqrt{\bar{\alpha}_t}\boldsymbol{x}^{(0)}, (1-\bar{\alpha}_t)\boldsymbol{I}). \tag{17}
$$

Using the Gaussian above, we can parameterize $\boldsymbol{x}_i^{(t)}$ as $\boldsymbol{x}_i^{(t)}(\boldsymbol{x}_i^{(0)}, \boldsymbol{\epsilon}) = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_i^{(0)} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$:

$$L_i^{(t-1)} = \mathbb{E}_{\boldsymbol{x}_i^{(0)}, \boldsymbol{\epsilon}, \boldsymbol{z}} \left[ \frac{1}{2\beta_t} \left\| \frac{1}{\sqrt{\alpha_t}} \left( \boldsymbol{x}_i^{(t)} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon} \right) - \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t)}, t, \boldsymbol{z}) \right\|^2 \right] + C. \tag{18}$$

The above equation reveals that $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t)}, t)$ must predict $\frac{1}{\sqrt{\alpha_t}} \left( \boldsymbol{x}_i^{(t)} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon} \right)$ given $\boldsymbol{x}_i^{(t)}$. Thus, $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t)}, t)$ can be parameterized as:

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t)}, t) = \frac{1}{\sqrt{\alpha_t}} \left( \boldsymbol{x}_i^{(t)} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t)}, t, \boldsymbol{z}) \right), \tag{19}$$

where $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t)}, t, \boldsymbol{z})$ is a function approximator (*i.e.*, neural network) intended to predict $\boldsymbol{\epsilon}$ from $\boldsymbol{x}_i^{(t)}$. Finally, $L_i^{(t-1)}$ can be simplified as

$$L_i^{(t-1)} = \mathbb{E}_{\boldsymbol{x}_i^{(0)}, \boldsymbol{\epsilon}, \boldsymbol{z}} \left[ \frac{\beta_t^2}{2\beta_t \alpha_t (1-\bar{\alpha}_t)} \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\boldsymbol{x}_i^{(0)} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t, \boldsymbol{z}) \right\|^2 \right] + C. \tag{20}$$

The simplified algorithm proposed in Ho et al. (2020) suggests choosing a random term from $\{\sum_{i=1}^N L_i^{(t-1)}\}_{t=1}^T$ to optimize at each training step. In addition to that, the prior loss term $L_{\boldsymbol{z}}$ in our objective function should also be considered. Since only one term in $\{\sum_{i=1}^N L_i^{(t-1)}\}_{t=1}^T$ is optimized at each step, we re-weight $L_{\boldsymbol{z}}$ with $\frac{1}{T}$. The adapted simplified training algorithm is as follows:

**Algorithm 3** Training (Simplified)

1: **repeat**
2: Sample $\boldsymbol{X}^{(0)} \sim q_{\text{data}}(\boldsymbol{X}^{(0)})$
3: Sample $\boldsymbol{z} \sim q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)})$
4: Sample $t \sim \text{Uniform}(\{1, \dots, T\})$
5: Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
6: Compute $\nabla \big[ \sum_{i=1}^N \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\boldsymbol{x}_i^{(0)} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t, \boldsymbol{z})\|^2 - \frac{1}{T} D_{\text{KL}}(q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{X}^{(0)}) \| p(\boldsymbol{z})) \big]$; Then perform gradient descent.
7: **until** converged

## C Affine Coupling Layers

In affine coupling layers (Dinh et al., 2016), both the input $\boldsymbol{w}^\ell$ and the output $\boldsymbol{w}^{\ell+1}$ are partitioned into two segments with the same length: $\boldsymbol{w}^\ell = [\boldsymbol{w}_1^\ell, \boldsymbol{w}_2^\ell]$, $\boldsymbol{w}^{\ell+1} = [\boldsymbol{w}_1^{\ell+1}, \boldsymbol{w}_2^{\ell+1}]$. The first segment of the input $\boldsymbol{w}_1^\ell$ is unchanged, and is used to update the other segment $\boldsymbol{w}_2^\ell$ by scaling and translation (affine transform):

$$\begin{aligned} \boldsymbol{w}_1^{\ell+1} &= \boldsymbol{w}_1^\ell, \\ \boldsymbol{w}_2^{\ell+1} &= \boldsymbol{w}_2^\ell \odot F(\boldsymbol{w}_1^\ell) + G(\boldsymbol{w}_1^\ell), \end{aligned} \tag{21}$$

where $F(\cdot)$ and $G(\cdot)$ can be arbitrary (non-invertible) neural networks, and "$\odot$" denotes element-wise multiplication. The inverse of the above affine transform is trivially obtained by subtraction and division, since $\boldsymbol{w}_1^{\ell+1} = \boldsymbol{w}_1^\ell$. Several affine coupling layers can be stacked with alternating partitioning to construct a complex invertible flow.

## D Model Architectures

**PointNet Encoder** The architecture of our encoder follows that of PC-GAN, PointFlow and ShapeGF (Qi et al., 2017a; Achlioptas et al., 2018; Yang et al., 2019; Cai et al., 2020). Specifically, we feed point clouds into a 3-128-256-512 MLP with the ReLU nonlinearity followed by a max-pooling to obtain a global 512-dimension feature. Then, the feature is fed into a 512-256-128-256 MLP with the ReLU nonlinearity and we obtain the latent code of 256-dimension.

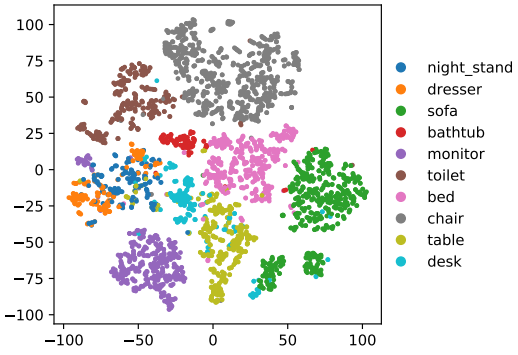


Figure 4: The t-SNE clustering visualization of latent codes obtained from the encoder.

**Prior Flow** We stack 14 affine coupling layers to construct the prior flow. The dimension of hidden states is 256, identical to the dimension of latent codes. Following each of the layers, we apply moving batch normalization (Ioffe & Szegedy, 2015; Grathwohl et al., 2018). Both the scaling and translation networks $F(\cdot)$ and $G(\cdot)$ are 128-256-256-128 MLPs with the ReLU nonlinearity.

**Diffusion Process** The number of steps $T$ in the diffusion process is 200. We set the variance schedules to be $\beta_1 = 0.0001$ and $\beta_T = 0.05$, and $\beta_t$'s $(1 < t < T)$ are linearly interpolated.

**Reverse Diffusion Kernel** The reverse diffusion kernel in Eq. 4 is paramtereized by $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(t)}, t, \boldsymbol{z})$, as derived in Appendix B. We implement it using a variant of MLP, which consists of a series of `concatsquash` layers (Grathwohl et al., 2018) defined as:

$$\boldsymbol{h}^{\ell+1} = \texttt{CS}(\boldsymbol{h}^{\ell}, t, \boldsymbol{z}) = (\boldsymbol{W}_1\boldsymbol{h}^{\ell} + \boldsymbol{b}_1) \odot \sigma(\boldsymbol{W}_2\boldsymbol{c} + \boldsymbol{b}_2) + \boldsymbol{W}_3\boldsymbol{c}, \tag{22}$$

where $\boldsymbol{h}^{\ell}$ is the input to the layer and $\boldsymbol{h}^{\ell+1}$ is the output. The input to the first layer is the 3D positions of points $\boldsymbol{x}_i^{(t)}$. $\boldsymbol{c} = [t, \sin(t), \cos(t), \boldsymbol{z}]$ is the context vector, and $\sigma$ denotes the sigmoid function. $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, $\boldsymbol{W}_3$, $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are all trainable parameters. The dimension of the `concatsquash`-MLP used in our model is 3-128-256-512-256-128-3, and we use the LeakyReLU nonlinearity between the layers.

## E Additional Visual Results

We project the latent codes of ModelNet-10 point clouds produced by the encoder trained by ShapeNet into the 2D plane using t-SNE (Maaten & Hinton, 2008), and present it in Figure 4. It can be observed that there are significant margins between most categories, which indicates that our model manages to learn informative representations. In Figure 5, we visualize the interpolation and extrapolation between latent codes. In Figure 6, we present more generated samples obtained from our model.

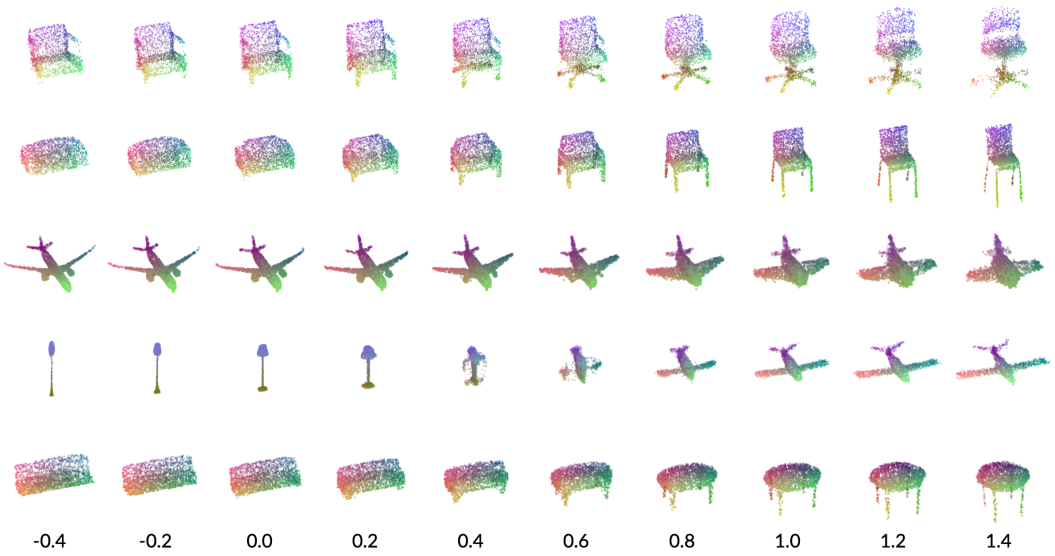


Figure 5: Latent space interpolation and extrapolation.

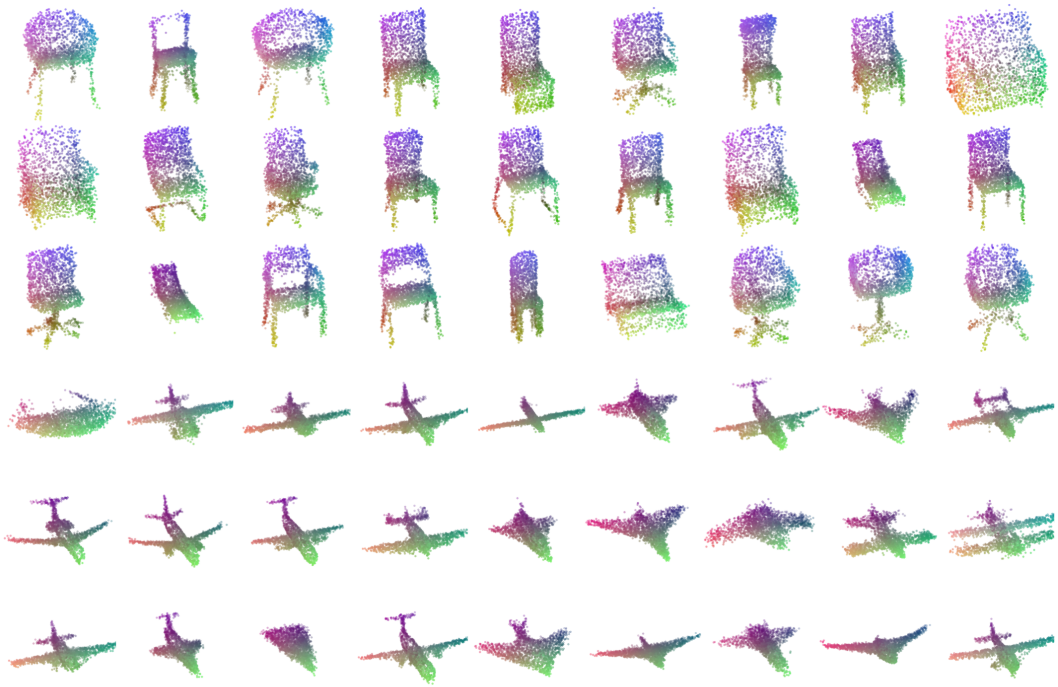

Figure 6: More generated samples from our model.