# DiffFacto: Controllable Part-Based 3D Point Cloud Generation with Cross Diffusion — Supplementary Materials

George Kiyohiro Nakayama[1] Mikaela Angelina Uy[1] Jiahui Huang[2] Shi-Min Hu[2]
Ke Li[3] Leonidas Guibas[1]
[1]Stanford University [2]Tsinghua University [3]Simon Fraser University

This document supplements our submission DiffFacto: Controllable Part-Based 3D Point Cloud Generation with Cross Diffusion. In particular, we provide detailed derivations and proofs of our training objective (Sec 1) and generalized forward kernel (Sec 2), additional experiment comparisons and results (Sec 3), and additional details on our network (Sec 4), implementation (Sec 5) and experiment set-ups (Sec 6).

## Contents

## 1. Objective Derivation

We derive the evidence lower bound presented in Sec. 4.1 of our main paper. Let $Q_{\boldsymbol{\varphi}}(\mathbf{Z}, \mathbf{T}|S)$ be a variational family with the following factorization

$$Q_{\boldsymbol{\varphi}}\left(\mathbf{Z}, \mathbf{T}|S\right) = Q_{\boldsymbol{\varphi}}\left(\mathbf{Z}|S\right) Q\left(\mathbf{T}|S\right) = Q\left(\mathbf{T}|S\right) \prod_{j=1}^{m} Q_{\varphi_j}\left(Z_j|S_j\right) = Q\left(\mathbf{T}|S\right) \prod_{j=1}^{m} Q_{\varphi_j}\left(Z_j|\hat{S}_j\right), \tag{1}$$

where we learn independent $Q_{\varphi_j}\left(Z_j|\hat{S}_j\right) \forall j$, *i.e.* our part stylizers, as a variational encoder that is parameterized as a Gaussian distribution with learnable mean and diagonal variance. Since given an input segmented shape $S$, its corresponding part transformations $\mathbf{T}$ is known, then the distribution $Q\left(\mathbf{T}|S\right)$ is deterministic. Furthermore, we highlight that each of the encoders $Q_{\varphi_j}$ only takes in the canonicalized part $\hat{S}_j$ as input, which enables us to model the prior distribution $P(Z_j)$ to only encode information about the part style and not of its size and scale. With this variational family, we can write the evidence lower bound as:

$$\mathbb{E}_S\left[\log P_{\phi,\boldsymbol{\psi},\theta}\left(S\right)\right]$$

$$= \mathbb{E}_S\left[\log \iint P_\phi\left(S|\boldsymbol{z},\boldsymbol{\tau}\right) P_{\boldsymbol{\psi},\theta}\left(\boldsymbol{z},\boldsymbol{\tau}\right)\, \mathrm{d}\boldsymbol{z}\mathrm{d}\boldsymbol{\tau}\right]$$

$$= \mathbb{E}_S\left[\log \iint \frac{P_\phi\left(S|\boldsymbol{z},\boldsymbol{\tau}\right) P_{\boldsymbol{\psi}}(\boldsymbol{z}) P_\theta(\boldsymbol{\tau}|\boldsymbol{z})}{Q_{\boldsymbol{\varphi}}(\boldsymbol{z},\boldsymbol{\tau}|S)} Q_{\boldsymbol{\varphi}}(\boldsymbol{z},\boldsymbol{\tau}|S)\mathrm{d}\boldsymbol{z}\mathrm{d}\boldsymbol{\tau}\right]$$

$$= \mathbb{E}_S\left[\log \iint \frac{P_\phi\left(S|\boldsymbol{z},\boldsymbol{\tau}\right) P_\theta(\boldsymbol{\tau}|\boldsymbol{z}) \prod_{j=1}^m P_{\psi_j}(z_j)}{Q\left(\boldsymbol{\tau}|S\right)\prod_{j=1}^m Q_{\varphi_j}\left(z_j|\hat{S}_j\right)} Q\left(\boldsymbol{\tau}|S\right) \prod_{j=1}^{m} Q_{\varphi_j}\left(z_j|\hat{S}_j\right) \mathrm{d}\boldsymbol{z}\mathrm{d}\boldsymbol{\tau}\right]$$

$$= \mathbb{E}_S\left[\log \mathbb{E}_{z_j\sim Q_{\varphi_j},\boldsymbol{\tau}\sim Q(\boldsymbol{T}|S)}\left[\frac{P_\phi\left(S|\boldsymbol{z},\boldsymbol{\tau}\right) P_\theta(\boldsymbol{\tau}|\boldsymbol{z}) \prod_{j=1}^m P_{\psi_j}(z_j)}{Q\left(\boldsymbol{\tau}|S\right)\prod_{j=1}^m Q_{\varphi_j}\left(z_j|\hat{S}_j\right)}\right]\right]$$

By Jensen's Inequality,

$$\geq \mathbb{E}_{S\sim P(S), z_j\sim Q_{\varphi_j},\boldsymbol{\tau}\sim Q(\boldsymbol{T}|S)}\left[\log \frac{P_\phi\left(S|\boldsymbol{z},\boldsymbol{\tau}\right) P_\theta(\boldsymbol{\tau}|\boldsymbol{z}) \prod_{j=1}^m P_{\psi_j}(z_j)}{Q\left(\boldsymbol{\tau}|S\right)\prod_{j=1}^m Q_{\varphi_j}\left(z_j|\hat{S}_j\right)}\right]$$

$$= \mathbb{E}_{S,z_j,\boldsymbol{\tau}}\left[\log P_\phi\left(S|\boldsymbol{z},\boldsymbol{\tau}\right) + \sum_{j=1}^m \log \frac{P_{\psi_j}(z_j)}{Q_{\varphi_j}\left(z_j|\hat{S}_j\right)} + \log \frac{P_\theta(\boldsymbol{\tau}|\boldsymbol{z})}{Q(\boldsymbol{\tau}|S)}\right]$$

Since $Q(\boldsymbol{T}|S)$ is a deterministic distribution, $Q(\boldsymbol{\tau}|S) = 1$ for all $\boldsymbol{\tau} \sim Q(\boldsymbol{\tau}|S)$. Thus,

$$= \mathbb{E}_{S,z_j,\boldsymbol{\tau}}\left[\log P_\phi\left(S|\boldsymbol{z},\boldsymbol{\tau}\right) + \sum_{j=1}^m \log \frac{P_{\psi_j}(z_j)}{Q_{\varphi_j}\left(z_j|\hat{S}_j\right)} + \log P_\theta(\boldsymbol{\tau}|\boldsymbol{z})\right]$$

$$= \mathbb{E}_{S,\boldsymbol{z},\boldsymbol{\tau}}\left[\log P_\phi\left(S|\boldsymbol{z},\boldsymbol{\tau}\right)\right] - \sum_{j=1}^m \mathbb{E}_S\left[\mathrm{KL}\left(P_{\psi_j}(z_j)\|Q_{\varphi_j}\left(z_j|\hat{S}_j\right)\right)\right] + \mathbb{E}_{S,\boldsymbol{z},\boldsymbol{\tau}}\left[\log P_\theta(\boldsymbol{\tau}|\boldsymbol{z})\right]. \tag{2}$$

As detailed in the main paper, each term in the final objective corresponds to the training loss for each of our components (part stylizers, transformation sampler, and cross diffusion network). The first term is the reconstruction error given prior information $\mathbf{z}, \boldsymbol{\tau}$, and its maximization is done by the cross diffusion network, which is a Denoising Diffusion Probabilistic Model with a generalized forward diffusion kernel. We elaborate on the derivation of the generalized forward kernel in Sec. 2, and the cross diffusion network in Sec. 4.3. The second term is the KL divergence regularization loss for each of the part style latent distribution $P_{\psi_j}(Z_j)$. We implement each of the part style latent distributions with a prior flow model. For specifics, please refer to Sec. 4.1. Lastly, the third/final term corresponds to the part transformations, where the cIMLE training strategy maximizes this the objective, as explained in Sec. 4.2.

## 2. Generalized Forward Diffusion Kernel

We derive the forward diffusion process with the proposed generalized forward kernel in our Cross Diffusion Network (Sec. 4.3 of the main paper). Given a probability distribution $Q(X^{(0)})$ with $X^{(0)} \in \mathbb{R}^d$ that we want to model, we can define a forward diffusion process with a mean $\mu \in \mathbb{R}^d$ and a variance $\Sigma \in \mathbb{R}^{d\times d}_{>0}$ such that

$$Q(X^{(0:T)}|\mu,\Sigma) = Q(X^{(0)}) \prod_{t=1}^{T} Q(X^{(t)}|X^{(t-1)},\mu,\Sigma)$$

where

$$Q(X^{(t)}|X^{(t-1)},\mu,\Sigma) = \mathcal{N}\left(\sqrt{\alpha_t}X^{(t-1)} + (1-\sqrt{\alpha_t})\,\mu, (1-\alpha_t)\,\Sigma\right)$$

for each $t = 1,\ldots,T$. $\alpha_t = 1-\beta_t$ are variance scheduling parameters. Notice that by setting $\Sigma = \boldsymbol{I}$ and $\mu = \mathbf{0}$ we recover the classical diffusion kernel. Similar to the classical kernel, we can efficiently sample from $Q\left(X^{(t)}|x^{(0)},\mu,\Sigma\right)$ using the reparameterization trick:

$$Q\left(X^{(t)}|X^{(0)},\mu,\Sigma\right) = \mathcal{N}\left(\sqrt{\overline{\alpha}_t}X^{(0)} + \left(1-\sqrt{\overline{\alpha}_t}\right)\mu, (1-\overline{\alpha}_t)\Sigma\right).$$

Here, $\overline{\alpha_t} = \prod_{t'=1}^{t}\alpha_{t'}$. As $t \to \infty$ increases, $\overline{\alpha_t}$ decreases to zero. Thus, for large $T$, the distribution $Q\left(X^{(t)}|X^{(0)},\mu,\Sigma\right)$ approaches the Gaussian distribution $\mathcal{N}(\mu,\Sigma)$ with mean $\mu$ and variance $\Sigma$. By doing so, we can incorporate prior information in the forward diffusion process in terms of $\mu$ and $\Sigma$. We can also show that for each $t > 1$, the posterior distribution $Q(X^{(t-1)}|X^{(t)},X^{(0)},\mu,\Sigma)$ is also a Gaussian distribution, given by

$$Q\left(X^{(t-1)}|X^{(t)},X^{(0)},\mu,\Sigma\right) = \mathcal{N}\left(\Xi_t, \eta_t^2\Sigma\right), \tag{3}$$

where

$$\begin{aligned}\Xi_t &= \frac{\beta_t\sqrt{\overline{\alpha}_{t-1}}}{1-\overline{\alpha}_t}X^{(0)} + \frac{(1-\overline{\alpha}_{t-1})\sqrt{\alpha_t}}{1-\overline{\alpha}_t}X^{(t)} + \left(1 + \frac{\left(\sqrt{\overline{\alpha}_t}-1\right)\left(\sqrt{\alpha_t}+\sqrt{\overline{\alpha}_{t-1}}\right)}{1-\overline{\alpha}_t}\right)\mu,\\ \eta_t^2 &= \frac{\beta_t\left(1-\overline{\alpha}_{t-1}\right)}{1-\overline{\alpha}_t}.\end{aligned}$$

We can further simplify the expression for $\Xi_t$ by substituting

$$X^{(t)} = \sqrt{\overline{\alpha}_t}X^{(0)} + \left(1-\sqrt{\overline{\alpha}_t}\right)\mu + (1-\overline{\alpha}_t)\boldsymbol{\varepsilon}\sqrt{\Sigma}, \tag{4}$$

where $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})$ and $\sqrt{\Sigma}$ is the Cholesky decomposition of $\Sigma$. Thus, similar to the classical diffusion parameterizations, we can define a reverse diffusion process $P_\phi(X^{(0:T)}|\mu,\Sigma)$ that is also a Markov chain so that

$$P_\phi(X^{(0:T)}|\mu,\Sigma) = P(X^{(T)}|\mu,\Sigma)\prod_{t=1}^{T}P_\phi(X^{(t-1)}|x^{(t)},\mu,\Sigma),$$

where each $P_\phi(X^{(t-1)}|x^{(t)},\mu,\Sigma)$ is a Gaussian distribution parameterized by weights $\phi$ and tries to approximate the ground truth posterior distribution $Q\left(X^{(t-1)}|X^{(t)},X^{(0)},\mu,\Sigma\right)$. Specifically, as in classical diffusion models [6], we can parameterize each reverse kernel as a Gaussian distribution with learnable mean and fixed variance:

$$P_\phi(X^{(t-1)}|x^{(t)},\mu,\Sigma) = \mathcal{N}\left(\Xi_\phi\left(t,x^{(t)},\mu,\Sigma\right), \eta_t\sqrt{\Sigma}\right)$$

for each $t > 1$. In practice, instead of directly learning $\Xi_\phi$, we can parameterize $\Xi_\phi$ using Eq (3) and learn a noise approximator

|  | Chair |  |  | Airplane |  |  |
|---|---|---|---|---|---|---|
|  | MMD-P (↓) | COV-P (↑) | 1NNA-P | MMD-P (↓) | COV-P (↑) | 1NNA-P |
| SP-GAN [12] | 4.43 | 31.7 | 87.77 | 5.27 | 26.2 | 91.03 |
| SPAG [5] | 4.53 | 36.3 | 78.94 | 5.73 | 26.3 | 90.17 |
| NW [7] | 4.77 | 34.0 | 83.36 | 4.77 | 34.0 | 83.36 |
| **DiffFacto** (Ours) | **3.27** | **42.5** | **65.23** | **3.20** | **46.2** | **68.72** |

Table 1: **Intra-part evaluation with additional baselines.** MMD-P score is multiplied by $10^{-2}$. COV-P and 1NNA-P are reported in $\%$

.

$\varepsilon_\phi$ that approximate random noises. At sampling time, we can recover $P_\phi(X^{(t-1)}|x^{(t)}, \mu, \Sigma)$ using $\varepsilon_\phi$ by writing

$$\begin{aligned}\Xi_\phi\left(t, x^{(t)}, \mu, \Sigma\right) &= \frac{\beta_t\sqrt{\overline{\alpha}_{t-1}}}{1-\overline{\alpha}_t}X^{(0)} + \frac{(1-\overline{\alpha}_{t-1})\sqrt{\alpha_t}}{1-\overline{\alpha}_t}X^{(t)} + \left(1 + \frac{\left(\sqrt{\overline{\alpha}_t}-1\right)\left(\sqrt{\alpha_t}+\sqrt{\overline{\alpha}_{t-1}}\right)}{1-\overline{\alpha}_t}\right)\mu \\ &= \frac{1}{\sqrt{\alpha_t}}\boldsymbol{x}^{(0)} - \frac{1-\sqrt{\alpha_t}}{\sqrt{\alpha_t}}\mu - \frac{\beta_t\boldsymbol{\varepsilon}_\phi(t, x^{(t)}, \mu, \Sigma)}{\sqrt{\alpha_t\left(1-\overline{\alpha}_t\right)}}\sqrt{\Sigma}.\end{aligned} \tag{5}$$

Doing so, the training objective of the generalized diffusion process simplifies to an L2 losss with noise, as we discussed in our main paper.

## 3. Additional Experimental Results

We provide additional experiment results in this section. We show additional quantitative comparisons in Sec. 3.1. In Sec. 3.2, we evaluate DiffFacto's autoencoding quality compared with state-of-the-art point cloud reconstruction networks, as well as demonstrate the advantage of the generalized forward kernel in our Cross Diffusion Network in modeling complex part geometry and topology. In Sec. 3.3, we ablate the prior flows in our part style sampler. In Sec. 3.4 and Sec. 3.5, we showcase shape mixing and part-level interpolation results compared with control-enabled baselines. In Sec. 3.6 and Sec. 3.7, we showcase additional qualitative results on part-level and transformation sampling. Lastly, in Sec. **??**, we qualitatively evaluate our transformation sampler trained with conditional Maximum Likelihood Estimation (cIMLE) against other types of implicit methods.

### 3.1. Additional Quantitative Comparison

We provide additional quantitative comparisons for DiffFacto on part-level generation as shown in Tab. 1. Specifically, we include two state-of-the-art mesh-based generative networks SPAGHETTI [5] and Neural Wavelets [7]. As shown, although these methods require 3D mesh supervision, our approach is able to achieve better part-level generative scores compared to these works. Additionally, we also compare with SP-GAN [12], which is a GAN-based point cloud generative network. We also note that unlike the other point cloud baselines, SP-GAN does not allow for encoding of an existing shape, making shape editing non-trivial if not infeasible. We used pre-trained weights for the three networks and follow the same evaluation procedure as for the other baselines.

### 3.2. Autoencoding

We report autoencoding performance of DiffFacto in Tab. 2 compared to DPM [13] and ShapeGF [1]. We further modify the two baselines to model part latents by also encoding each segmented part. We also report their reconstruction score denoted by † superscript. We see that DiffFacto outperforms all baselines by a margin due to our novel cross diffusion network and generalized forward kernel. We also note that there is an increase in performance for both baselines with the additional part latent modification †. We also report the reconstruction score for our model without the generalized forward kernel, marked as *Ours*$^*$. Notice that there is a larger gap between our approach with and without the generalized forward kernel in the lamp category in which parts exhibit more complex geometrical and topological structures.

| Dataset | Metrics | DPM | DPM$^\dagger$ | ShapeGF | ShapeGF$^\dagger$ | Ours$^*$ | Ours (DiffFacto) |
|---|---|---|---|---|---|---|---|
| Airplane | CD (↓) | 2.266 | 1.70 | 2.082 | 1.599 | 1.460 | **1.413** |
| | EMD (↓) | 1.96 | 1.72 | 2.258 | 1.722 | 1.337 | **1.333** |
| Car | CD (↓) | 7.686 | 7.013 | 7.016 | 6.735 | 6.559 | **6.538** |
| | EMD (↓) | 3.229 | 3.179 | 3.283 | 3.323 | **2.782** | 2.788 |
| Chair | CD (↓) | 5.297 | 4.39 | 4.521 | 3.928 | 3.746 | **3.701** |
| | EMD (↓) | 2.83 | 2.64 | 3.181 | 2.646 | 2.222 | **2.214** |
| Lamp | CD (↓) | 8.334 | 5.55 | 8.243 | 5.100 | 3.787 | **3.481** |
| | EMD (↓) | 3.395 | 2.899 | 4.504 | 2.983 | 1.959 | **1.951** |

Table 2: Comparison of point cloud auto-encoding performance. † represents a modification of baselines to incorporate part latents. $*$ represents our framework without modification of the forward kernel. Reported CD and EMD are multiplied by $10^4$ and $10^2$ respectively.

| | MMD-P | COV-P | 1NNA |
|---|---|---|---|
| VAE | 3.47 | **44.3** | 72.59 |
| CNF (Ours) | **3.27** | 42.5 | **65.23** |

Table 3: **Ablation on Part Stylizer.** Individual-part level metrics ablating our part stylizer (MMD-P $\times 10^{-2}$).

### 3.3. Ablation on Part Stylizer

We replace the CNF in our part stylizer by directly using a standard Gaussian as the prior, *i.e*. VAE. Tab. 3 shows intra-part evaluation metrics comparing CNF with standard VAE. With the lower 1NNA and MMD-P, we note that CNF gives an overall better performance in capturing the part-style distribution than with vanilla VAE. We also observe that shapes generated with CNF are overall much clearer and with less noise than with VAE. This is because the prior flow relaxes the latent space regularization constraint by allowing the prior distribution to be transformed into a more complex distribution. For details of the prior flow, see Sec. 4.1.

### 3.4. Additional Part Style Mixing Results

Since we sample from independent part-style distributions to generate coherent shapes, shape mixing is a direct application where modeling the factorized latent distribution has a direct advantage. We showcase additional part style mixing results in chairs, airplanes, and lamps categories in Fig. 2, Fig. 1, and Fig. 3. We also further evaluate our mixing quality compared to our control-enabled baselines, Ctrl-LION [19] and Ctrl-ShapeGF [1], for the chair and airplane categories. Because we model a distribution of parts' size and location given a set of part style latent codes, across all the examples, we are able to produce coherent shapes despite the input parts having different sizes and locations in their original source shapes. In particular, when compared to the control-enabled baselines that do not explicitly model parts sizes, the baselines' mixing results often produce shapes with detached or interpenetrating parts as a result of incorrect part configuration.

### 3.5. Additional Part Level Interpolation Results

Moreover, since we explicitly model a part-style latent space, we can also interpolate shapes on the part level. Given two samples of part style codes $z_j, z'_j \sim P_{\psi_j}(Z_j)$, we use linear interpolation from $z_j$ to $z'_j$ obtain intermediate part latents uniformly spaced in between: $z^k_j = z_j + \frac{k}{10}(z'_j - z_j)$. We showcase qualitative results in all four categories in Fig. 4, 5, 6, 7 and 10. Notice that in all four categories, our interpolation results show smooth transitions from the source to the target of the interpolation, as most highlighted in the chair leg interpolation in Fig. 7. Moreover, the intermediate shapes are also plausible, where discrete jumps can be observed when no intermediate configuration is available. An example of this can be seen in the interpolation of engines in Fig. 4, where the number of engines jumps from four to two to allow a discrete transition. Moreover, we see that the unreferenced parts (colored in grey) remain unchanged during the interpolation process because we model independent distributions of part styles. Overall, the part-level interpolation results showcase our models' ability to generate plausible shapes with fine-grained control.

We also qualitatively compare DiffFacto with the control-enabled baselines, Ctrl-LION [19] and Ctrl-ShapeGF [1], on part-level interpolation quality. Fig. 11 and 13 show part level interpolation result for Ctrl-LION and Fig. 12 and 14 show part level interpolation result for Ctrl-ShapeGF, for the chair and airplane categories. We follow the same part-level interpolation procedure on the control-enabled baselines as described earlier for our method. Since the control enabled baselines do not explicitly model the distribution of part transformations given their styles, the interpolation of each part may result in detached parts. See the interpolation on chairs in Fig. 13. Moreover, since Ctrl-ShapeGF generates shapes by training a cGAN directly on its autoencoding latent space, its interpolation is less smooth as the autoencoding latent space is trained on discrete training examples. This can also be seen from the chair interpolation in Fig. 14 (see the last row where the intermediate geometry exhibits extensive artifacts).

## 3.6. Additional Part Level Sampling Results

We showcase that DiffFacto enables part-based controllable generation through part-level sampling. We include additional results on part-style sampling to supplement the left and middle column of Figure 3 in the main paper. We showcase both *sampling* a part style and *fixing* a part style for each shape. Fig. 15, 17 and 19 show additional qualitative results for *sampling* individual part styles (Figure 3 main paper -left). Fig. 16, 18 and 20 show additional qualitative results for *fixing* individual part styles (Figure 3 main paper -middle).

## 3.7. Additional Part-Configuration Sampling Results

We further show additional qualitative examples on our ability to enable part-configuration sampling, *i.e*. generate various plausible configurations of the shape with fixed part styles. Fig. 21 to 24 show additional qualitative results. This is supplement the right column in Figure 3 of the main paper.

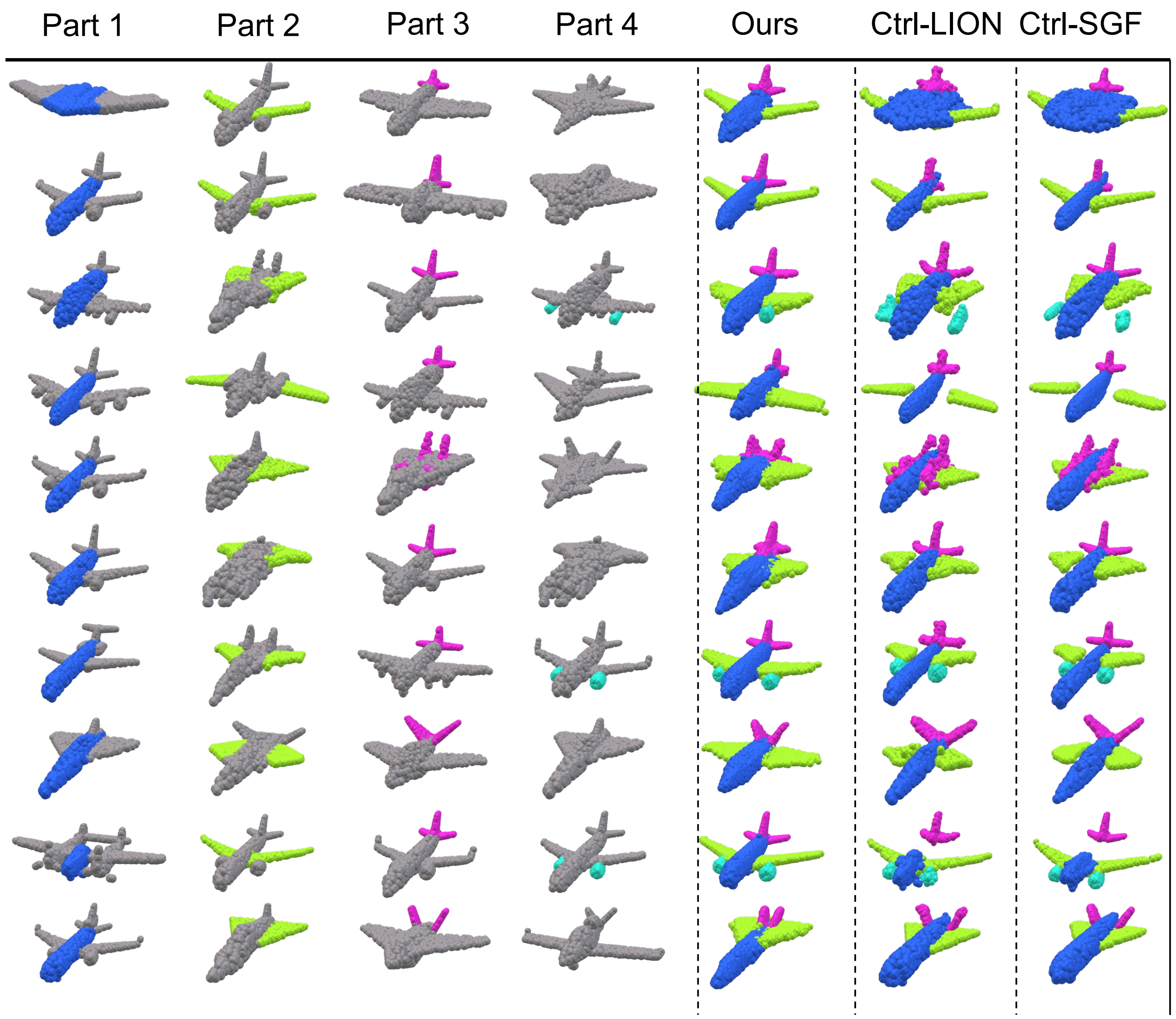


Figure 1: **Shape Mixing: Airplanes.** For each row, parts from different shapes are encoded and mixed for both our method (**Ours**) and the control-enabled baselines (**Ctrl-LION**, **Ctrl-SGF**). Notice that the control-enabled baselines fail to produce coherent shapes because it does not model the distribution of valid transformations while our method transforms each part globally into a coherent configuration so that the generated shapes are plausible.

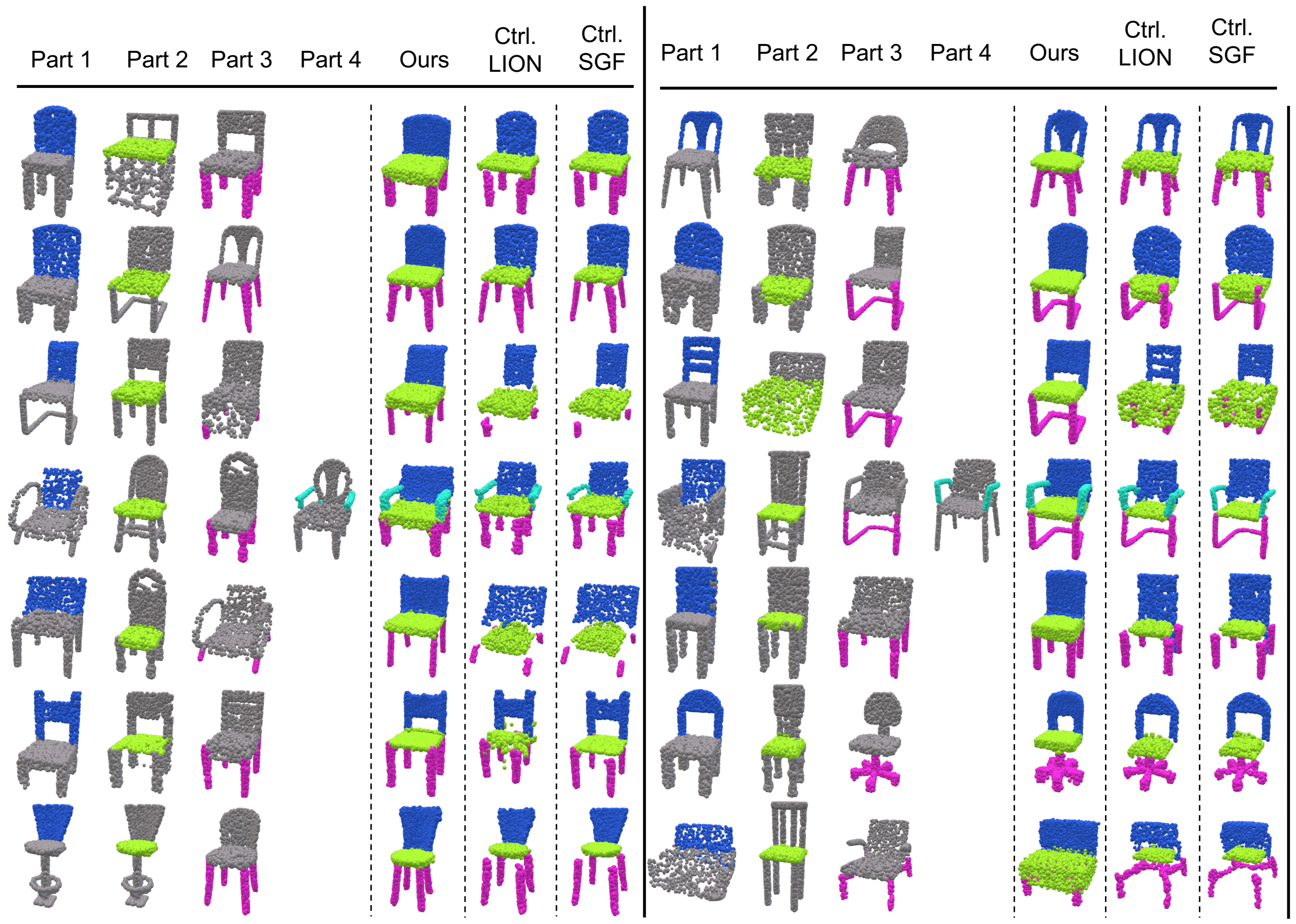


Figure 2: **Shape Mixing: Chairs.** For each example (two examples per row, separated by the solid line), parts from different shapes are encoded and mixed for both our method (**Ours**) and the control-enabled baselines (**Ctrl-LION**, **Ctrl-SGF**). Notice that the control-enabled baselines fail to produce coherent shapes because it does not model the distribution of valid transformations while our method transforms each part globally into a coherent configuration so that the generated shapes are plausible.

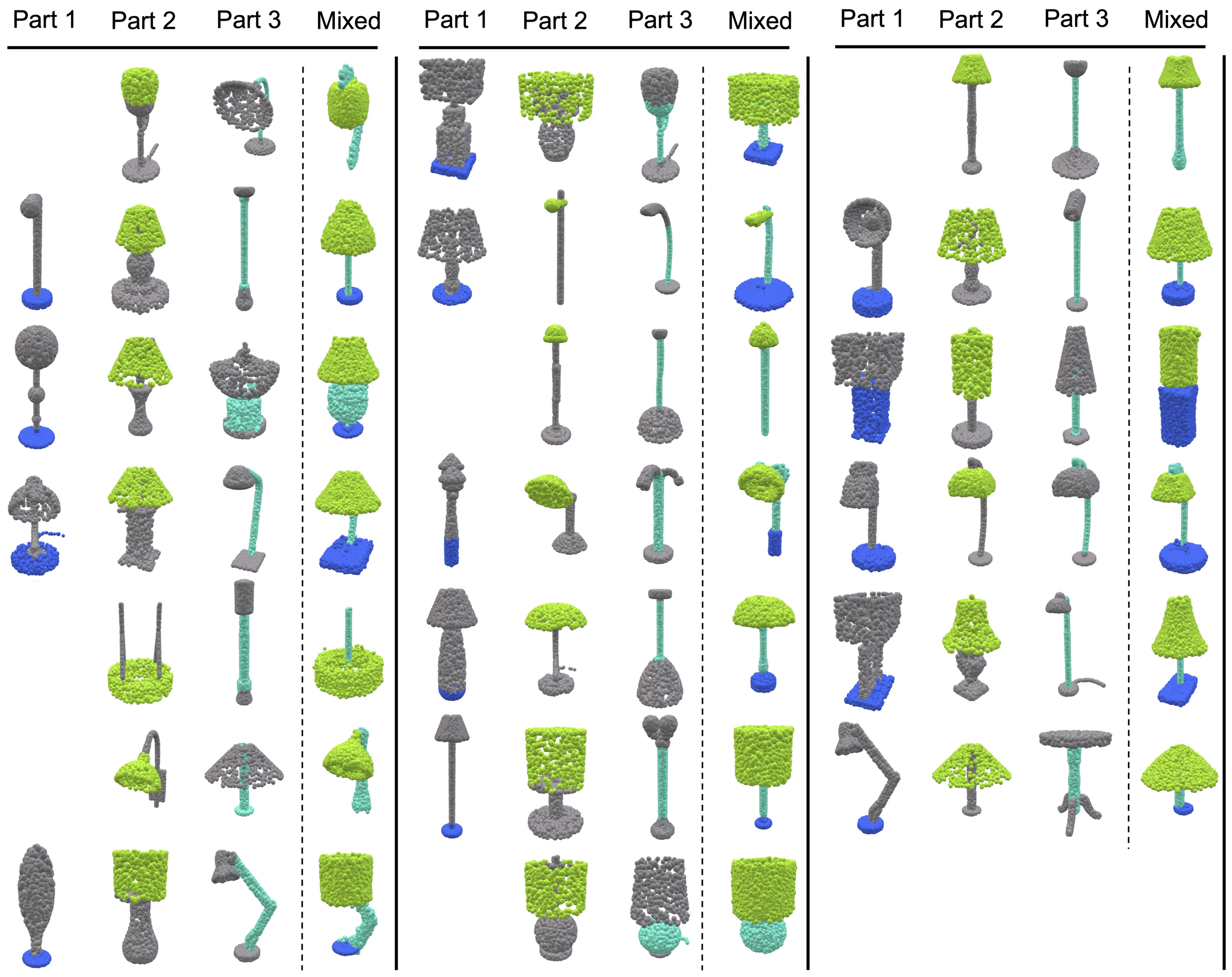


Figure 3: **Shape Mixing: Lamps.** Each row shows three examples of shape mixing for lamps. Notice that our model can adjust the parts' size and location so that the mixed shape is still coherent.

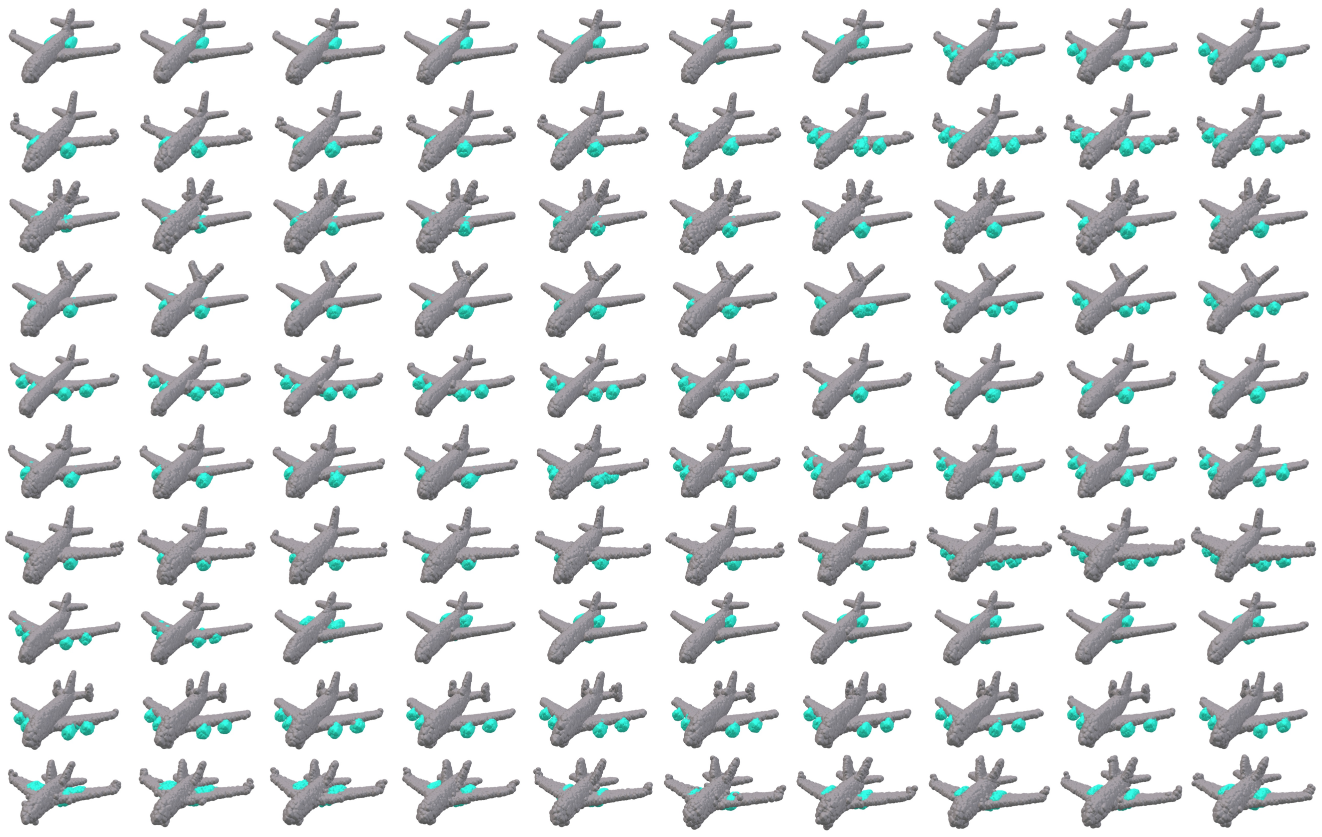


Figure 4: **Airplane Part Level Interpolation: Engines.** Each row is one interpolation path of the engine (colored part). Notice that because of the discrete nature of engine's number and location, the interpolation path between a different number of engines, and different locations of engines shows distinctive jumps from one configuration to another. This implies that the part latent space learned by our model only generates plausible shapes.

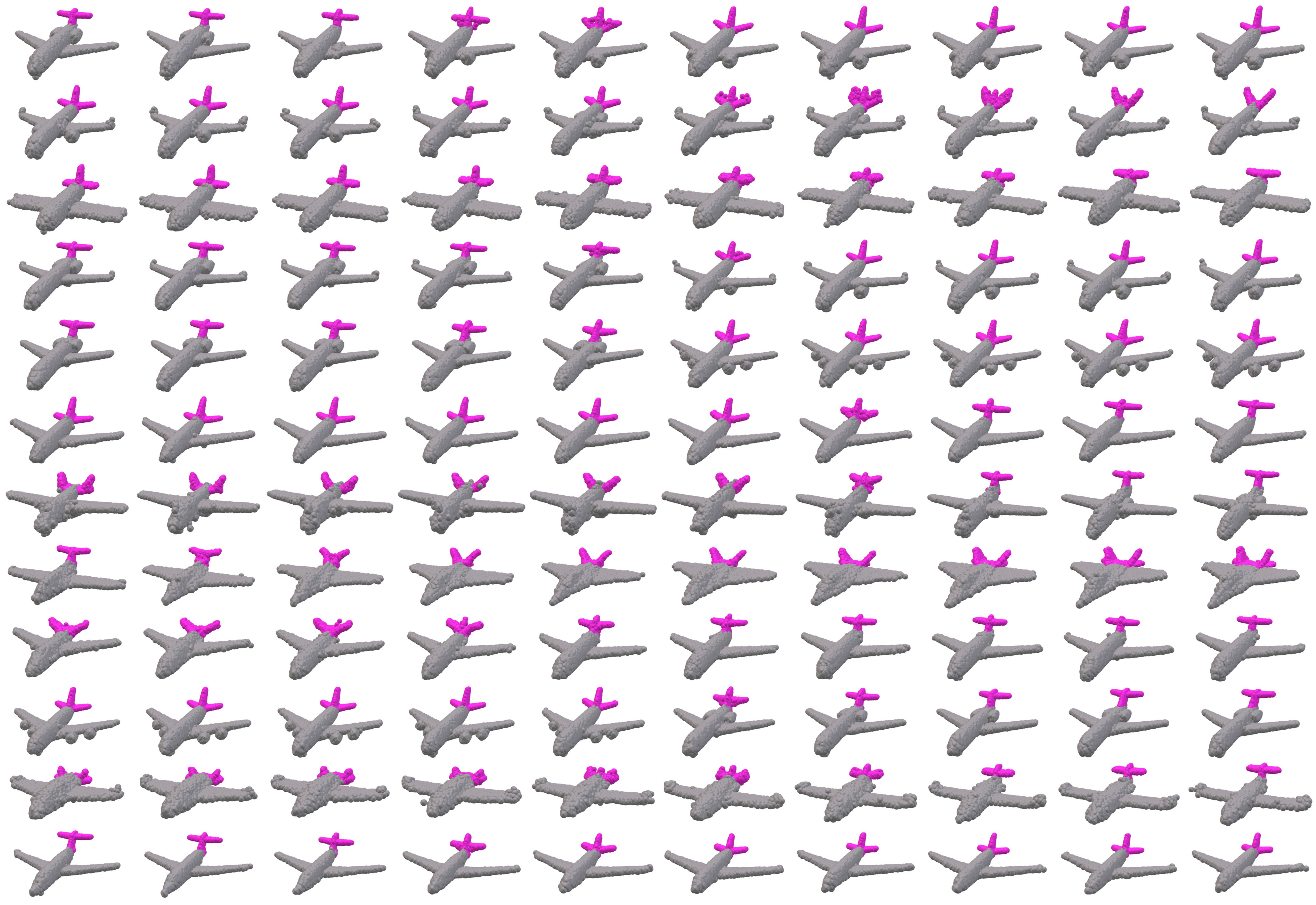


Figure 5: **Airplane Part Level Interpolation: Tail Wings.** Each row is one interpolation path of the tail wing (colored part). Similar to the engine's interpolation, the tail exhibits distinctive jumps from one mode to another so that the intermediate shapes are all plausible and coherent.

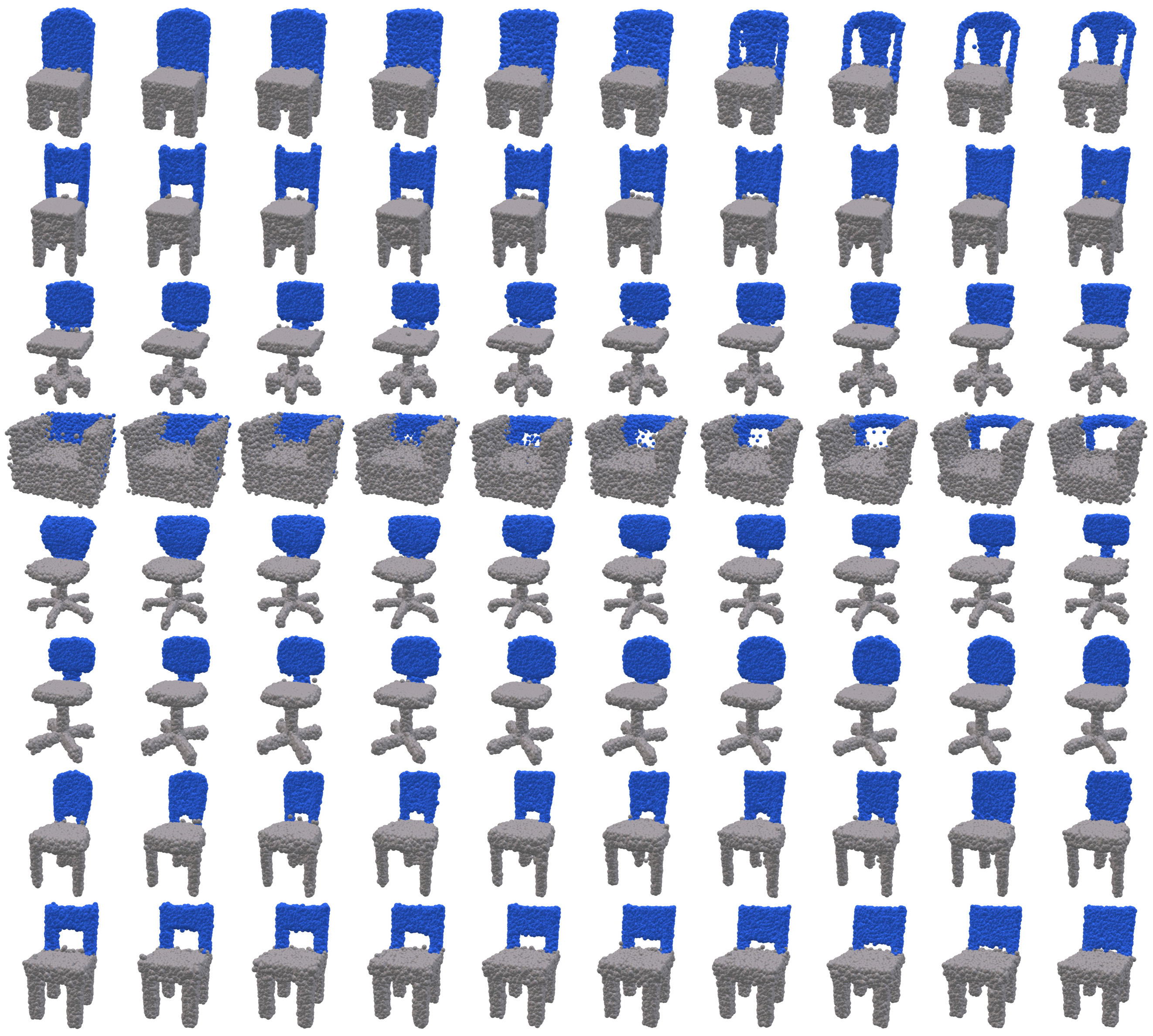


Figure 6: **Chair Part Level Interpolation: Back.** Each row is one interpolation path of the back (colored part). Notice that to adjust for the change in the back's style during interpolation, the unreferenced parts (in grey) adjust their size and location so that the output shapes are still coherent (see rows 3 and 6 for such covariance).

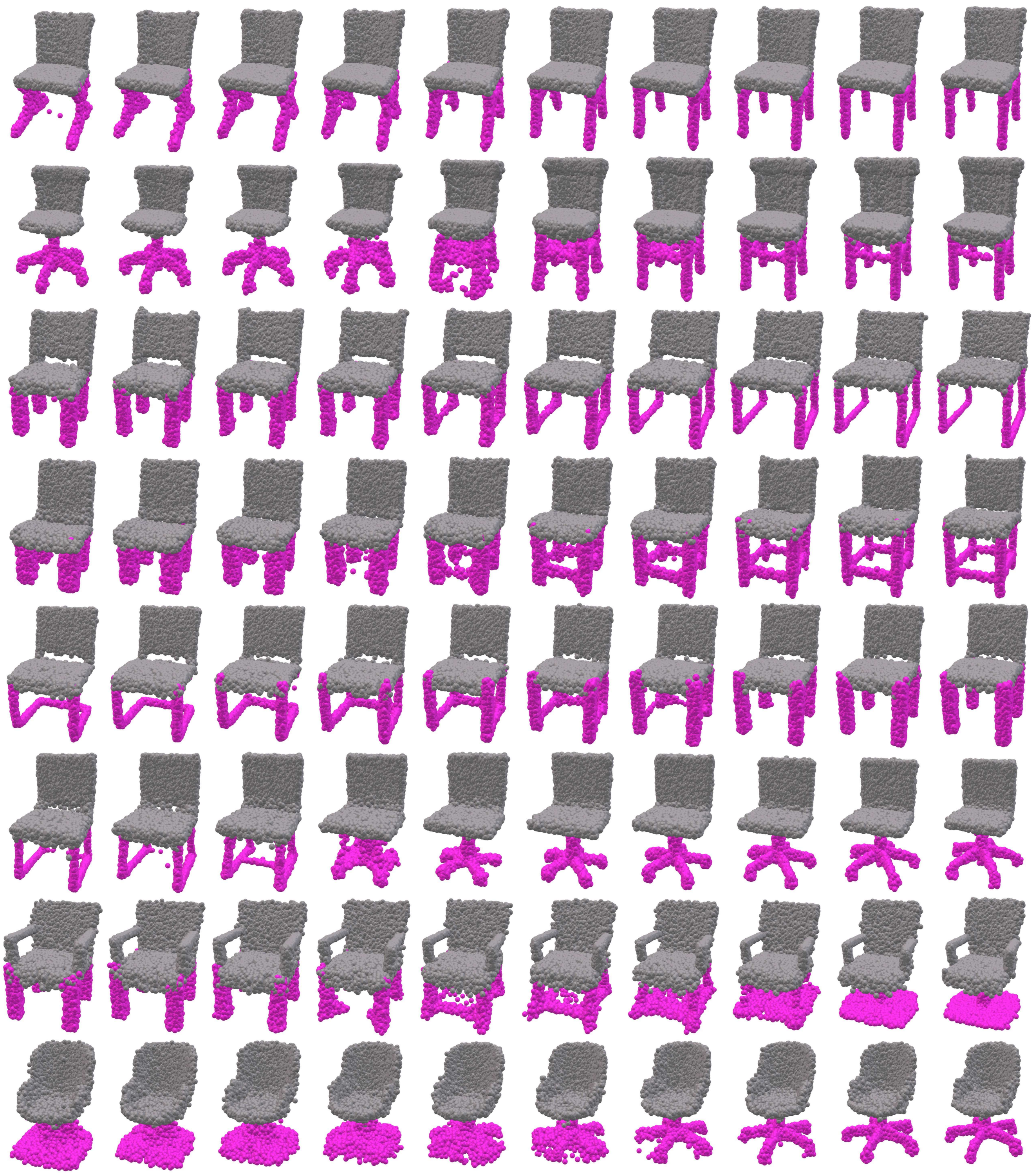


Figure 7: **Chair Part Level Interpolation: Legs.** Each row is one interpolation path of the legs (colored part). Notice that the intermediate shapes exhibit little artifacts and the interpolation path smoothly transitions between two different styles of legs. See, for example, rows 2 and 6.

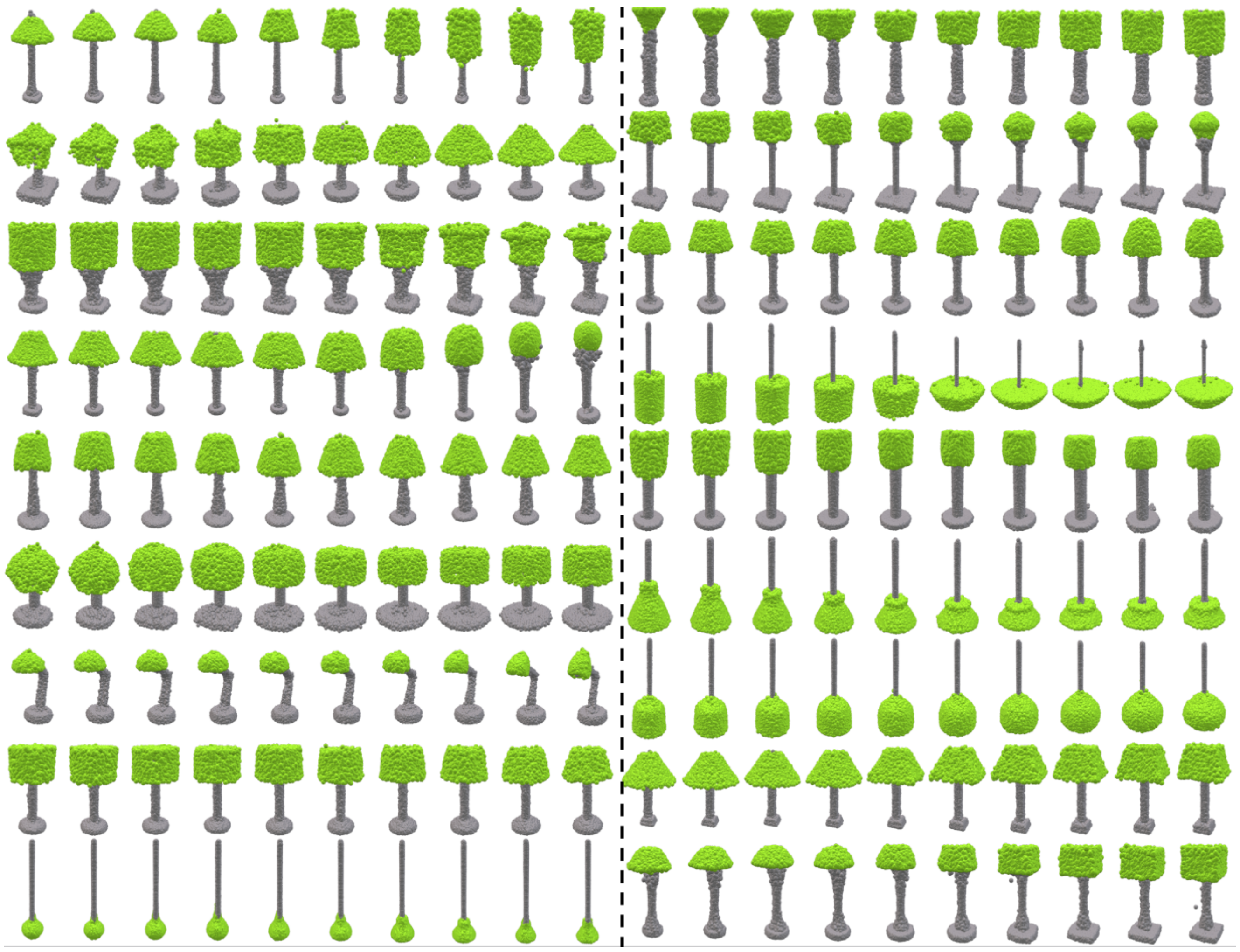


Figure 8: **Lamp Part Level Interpolation: Lamp Cap.** Each row shows two interpolation paths of the lamp cap (colored part) separated by the dashed line. Notice that for some of the interpolations where the starting part geometry is different from the ending geometry, the pole (in grey) needs to adjust its size so that the generated shape is coherent.

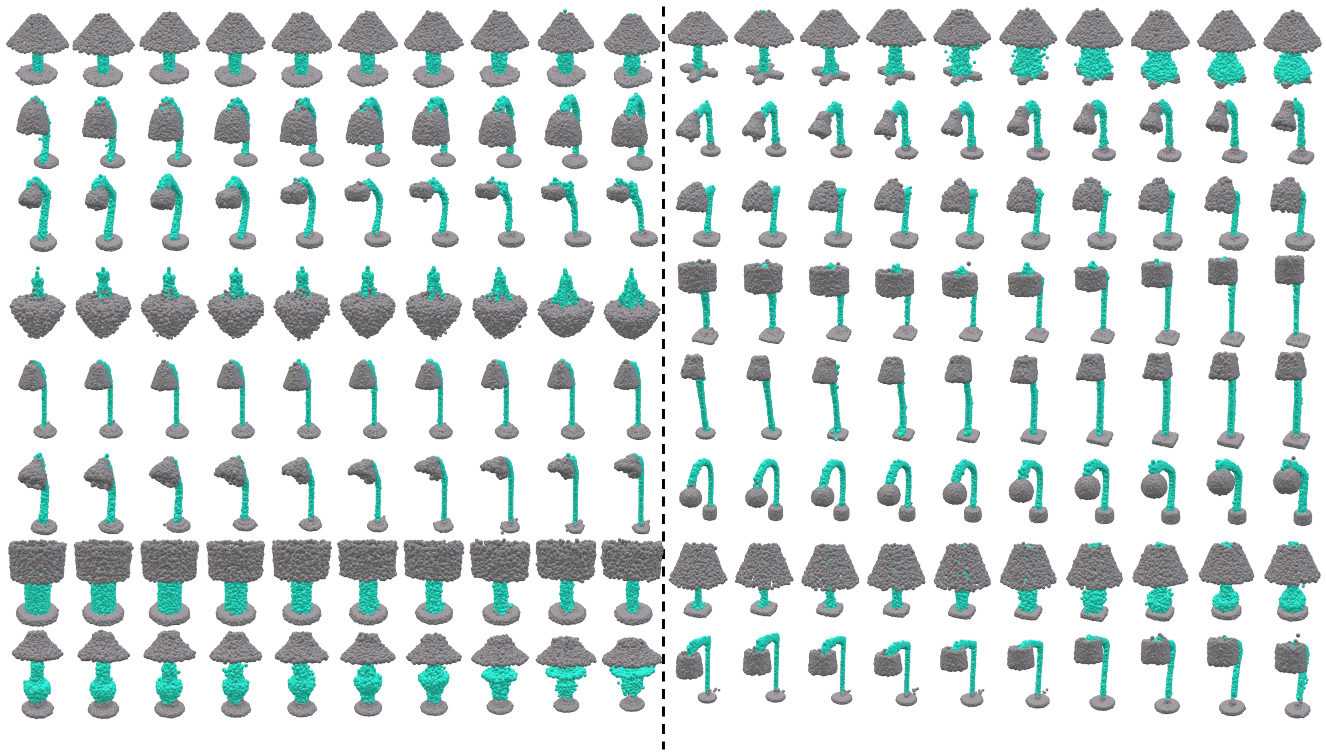


Figure 9: **Lamp Part Level Interpolation: Lamp Pole.** Each row shows two interpolation paths of the lamp pole (colored part) separated by the dashed line. Notice that sometimes the size of the lamp cap needs to vary accordingly to fit the change in the pole's style. See, for example, interpolations on rows 2 and 4.

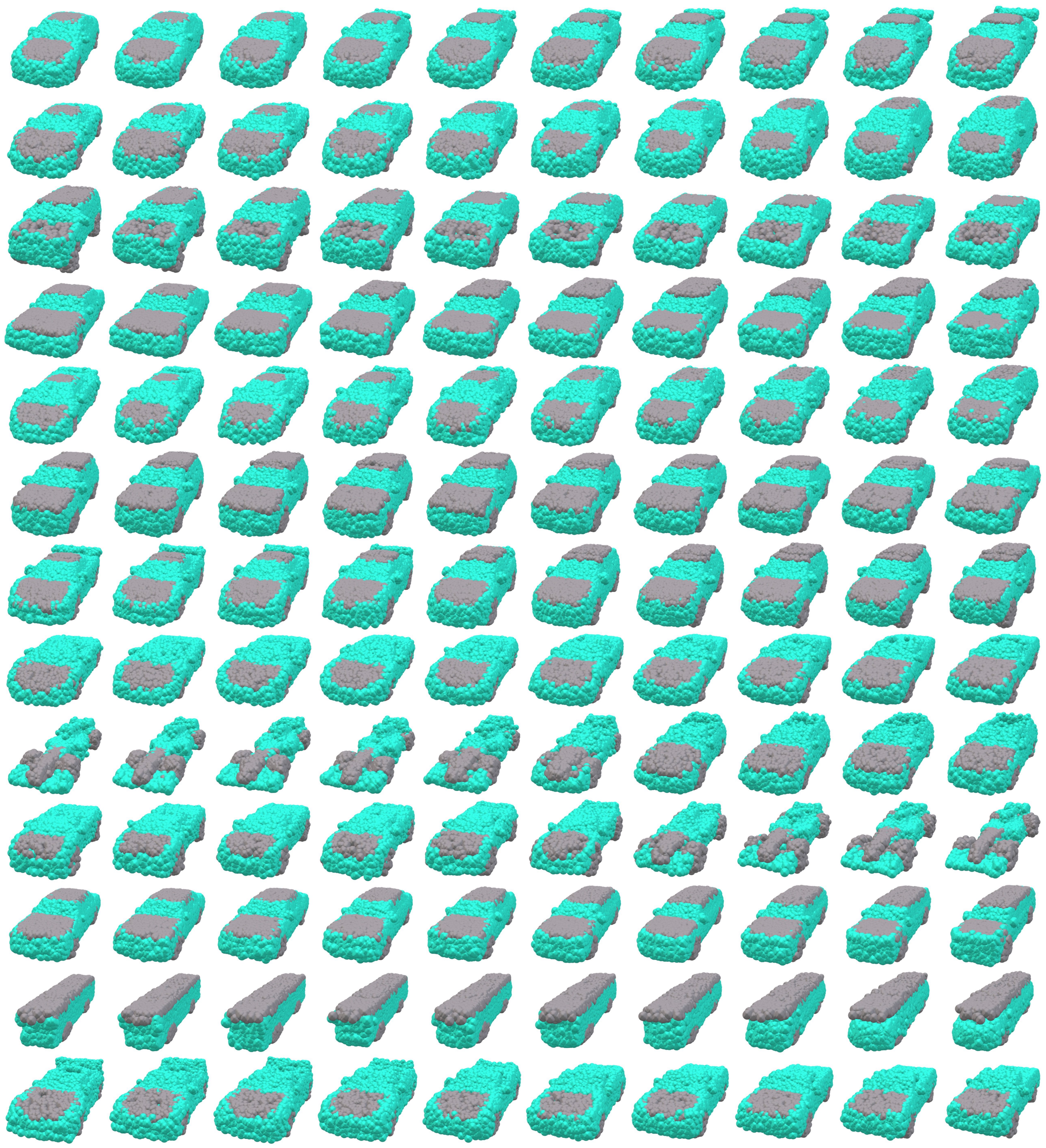


Figure 10: **Car Part Level Interpolation: Body** Each row shows one interpolation path of the car's body (colored part). Notice that the unreferenced parts vary accordingly to adjust to the change in the body's style (See rows 9 and 12).

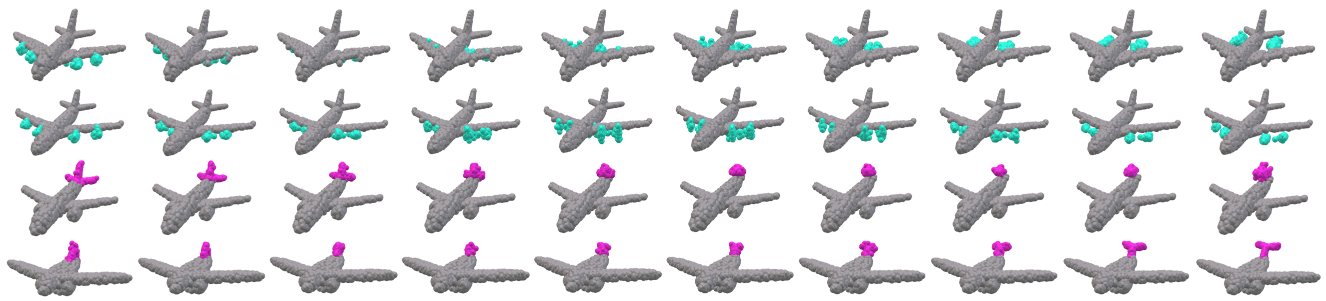


Figure 11: **Control LION Part Level Interpolation: Airplanes.** Each row shows one interpolation path of the colored part. Notice that since the control enabled LION does not model the distribution of part configurations, the intermediate shapes during interpolation often become invalid and not coherent. See, for example, row 3.

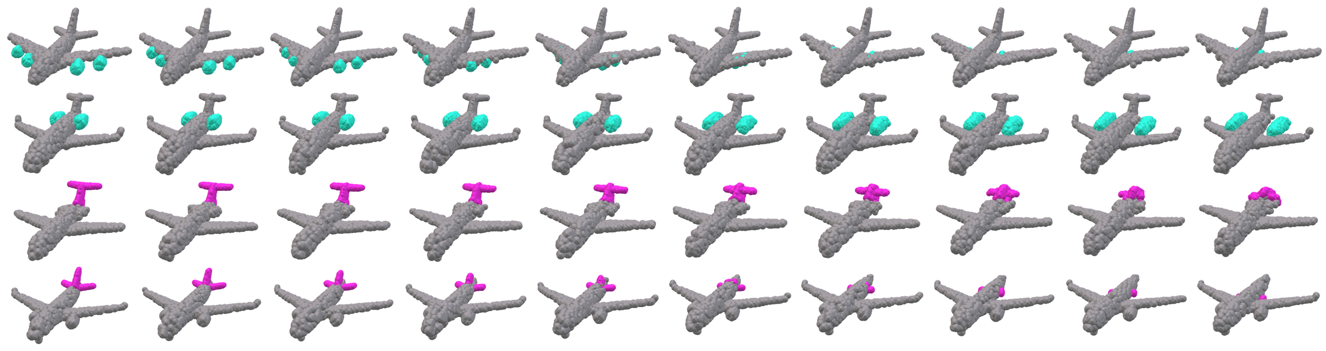


Figure 12: **Control ShapeGF Part Level Interpolation: Airplanes.** Each row shows one interpolation path of the colored part. Since the control enabled ShapeGF samples part latents on its autoencoding latent space, the interpolation path often shows non-smoothness in its intermediate steps. See, for example, the last row.

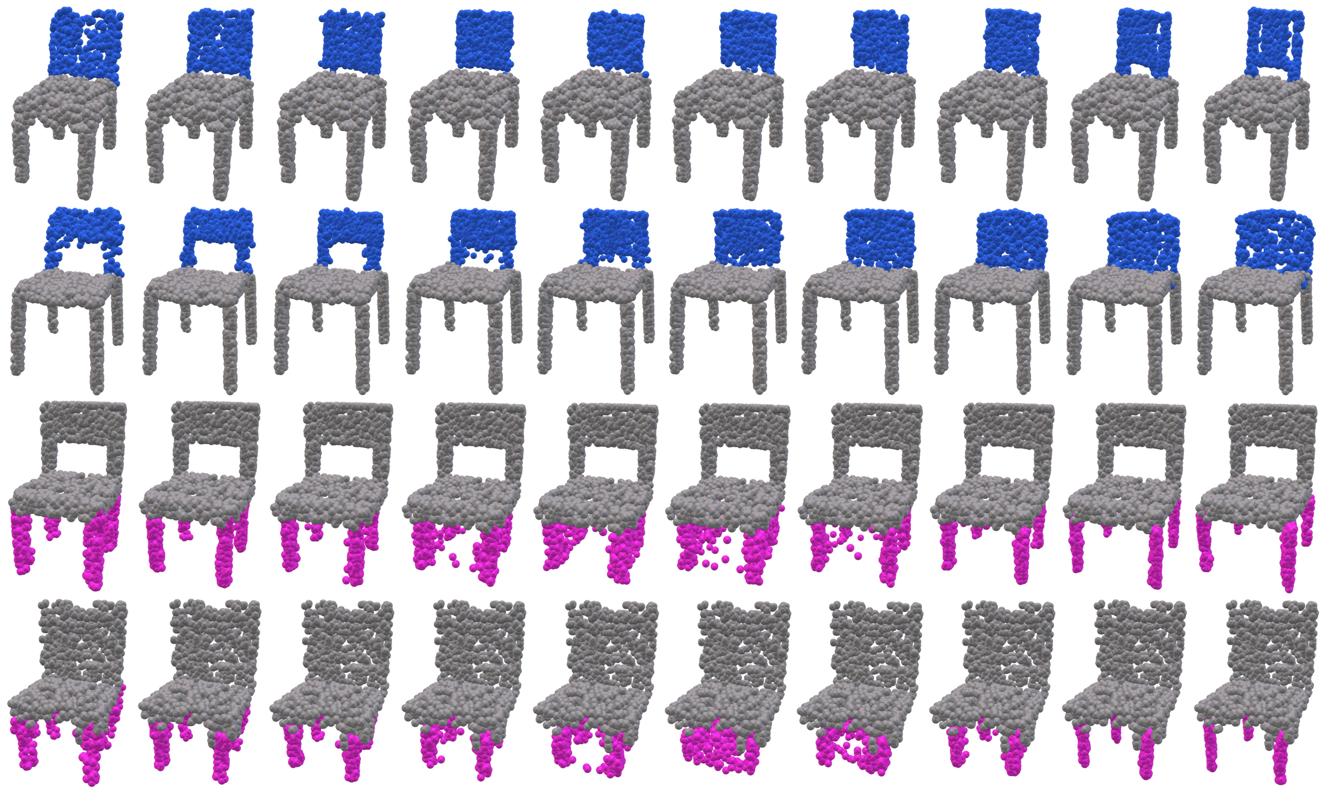


Figure 13: **Control LION Part Level Interpolation: Chairs.** Each row shows one interpolation path of the colored part. Since the control-enabled LION does not model the distribution of part configuration, the intermediate shapes often become detached (see the first row for such an example).

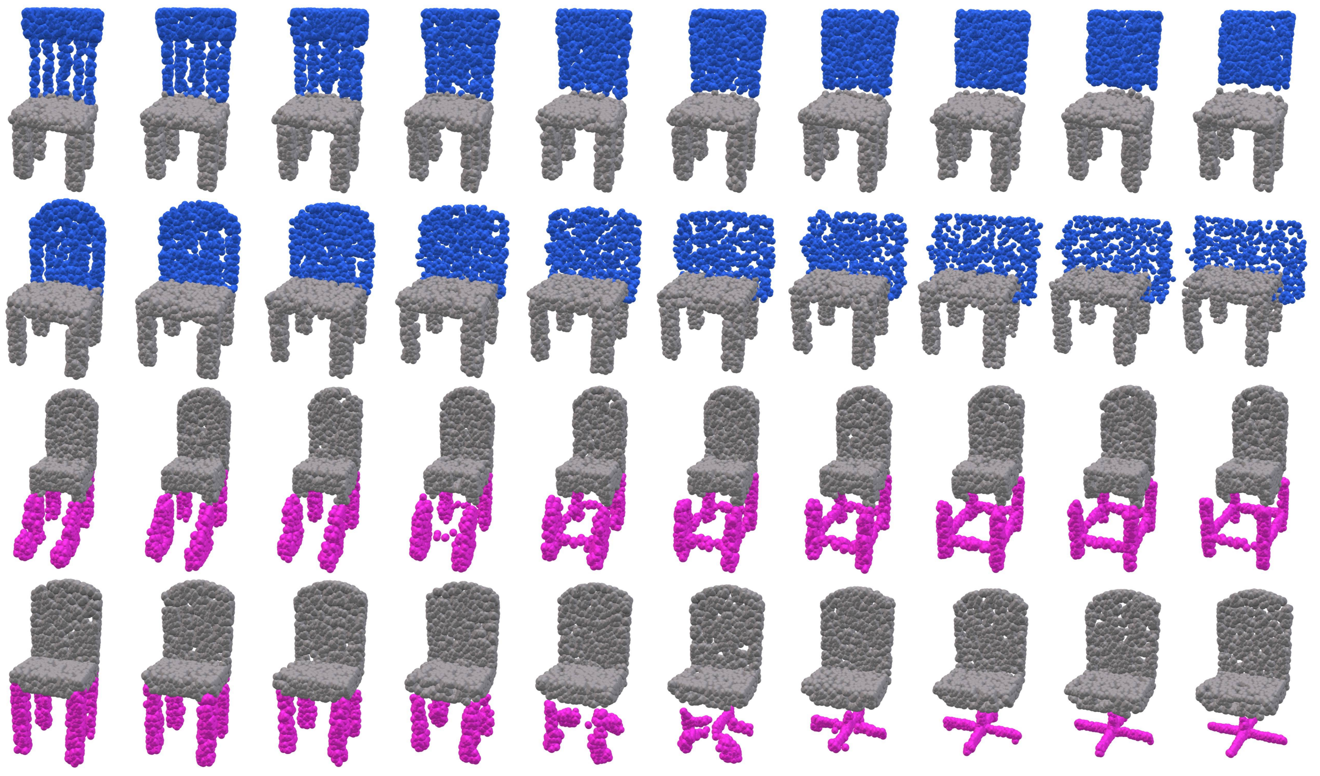


Figure 14: **Control ShapeGF Part Level Interpolation: Chairs.** Each row shows one interpolation path of the colored part. The last row shows the control-enabled ShapeGF exhibits artifacts in its interpolated shapes. Furthermore, parts become detached because it does not model the distribution of part transformation.

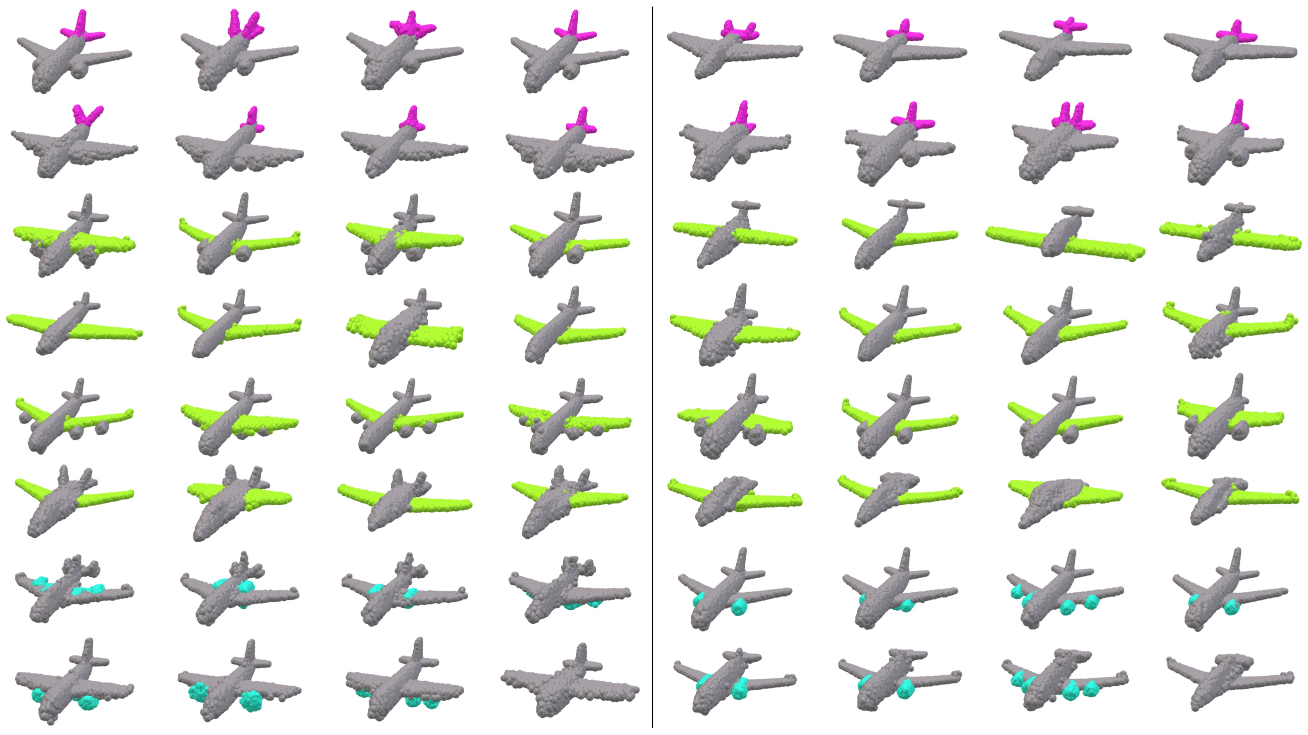


Figure 15: **Part-level Sampling: Airplanes, Sampling one part.** Each row shows two sets of airplanes (separated by the solid line) generated by sampling the part style latent associated with the colored part while keeping the rest of the part style latents for the grey parts fixed. Notice that the grey parts' styles stay fixed while the colored part varies across different samples. This ability to sample on the part level allows for intuitive user control in shapes generated.

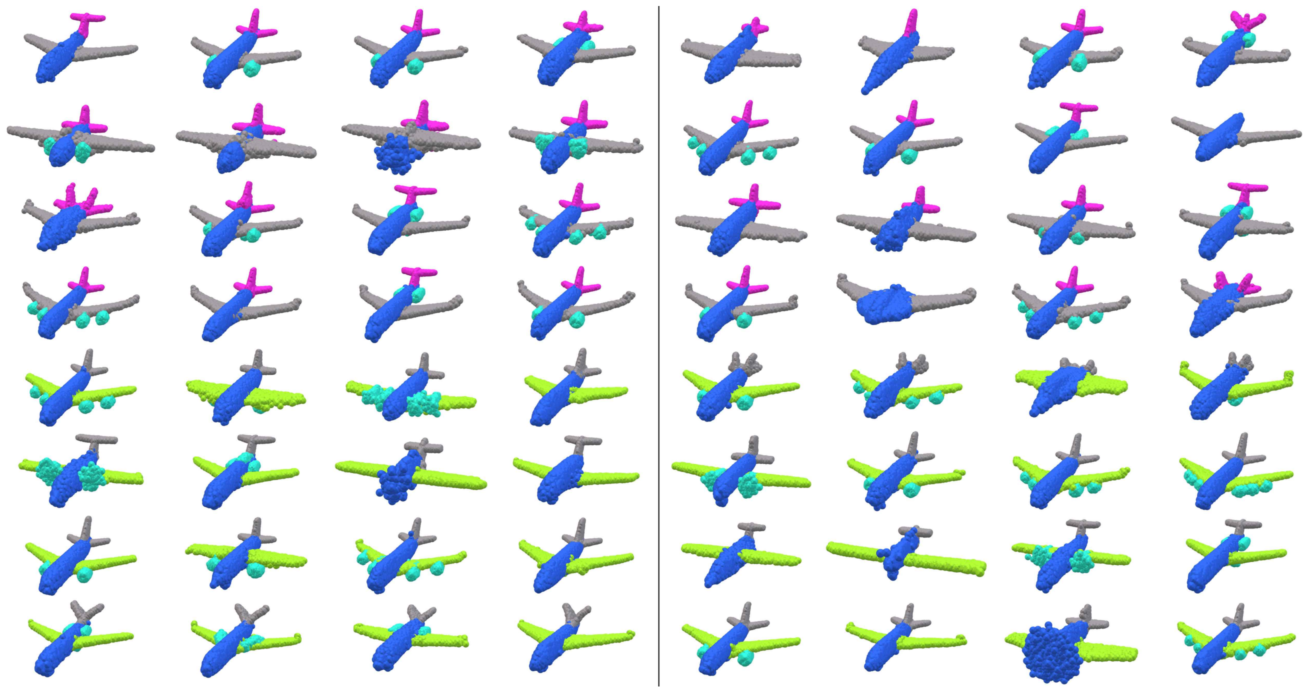


Figure 16: **Part-level Sampling: Airplanes, Fixing one part.** Each row shows two sets of airplanes (separated by the solid line) generated by sampling the set of part style latents associated with the colored parts while keeping the part style latent for the grey part fixed. Notice that the grey part's style stays fixed while the colored parts vary across different samples. The size and location of the grey part adjust to the sampled other part styles to generate coherent shapes.

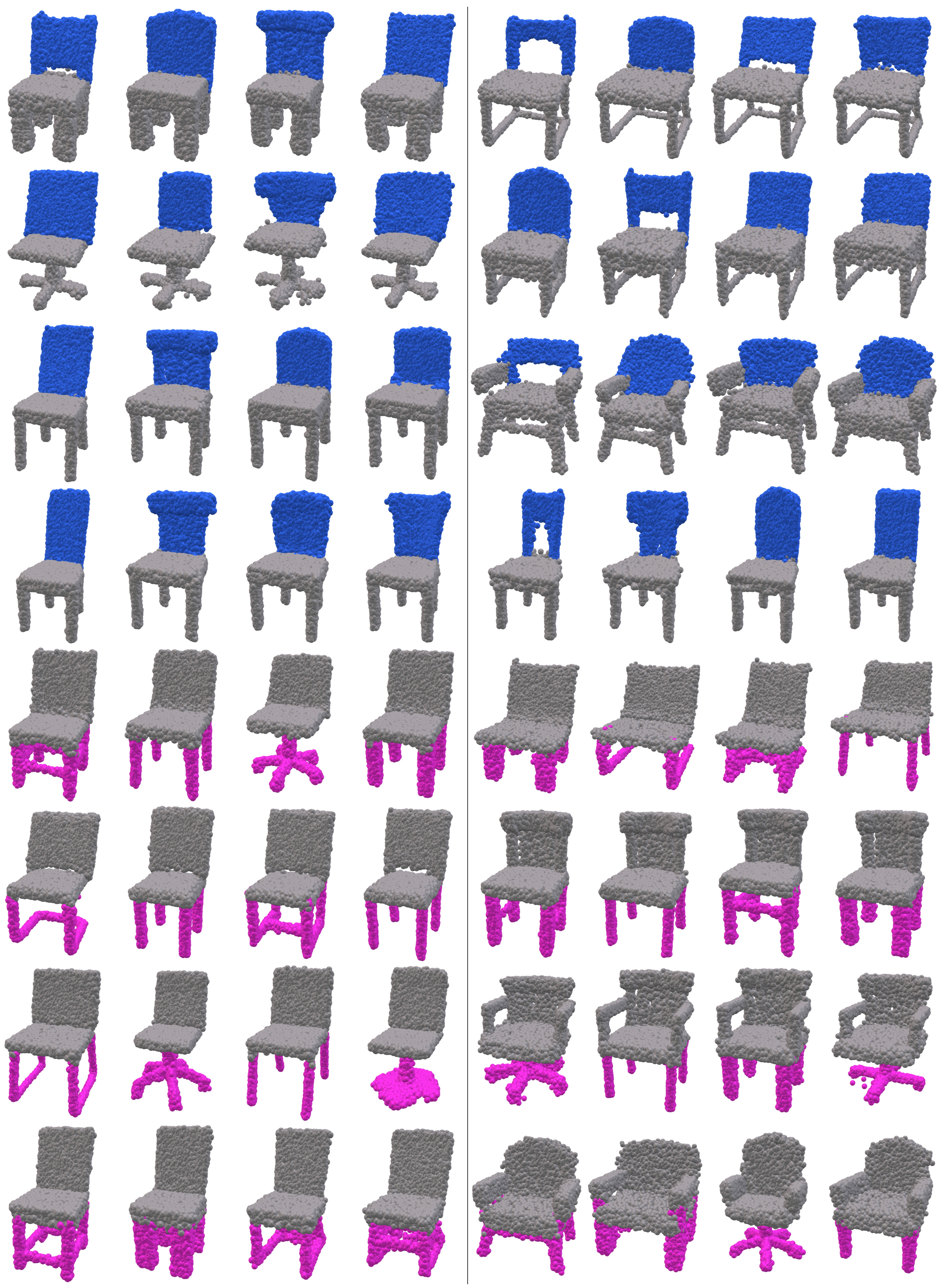


Figure 17: **Part-level Sampling: Chairs, Sampling one part.** Each row shows two examples (separated by the solid line) of sampling the colored part style latent while keeping the other part style latents unchanged. Notice that, based on the sampled part style, the unreferenced part changes their sizes accordingly so that the output shape stays plausible and coherent. See, for example, the change of seat's size on rows 2 and 3.

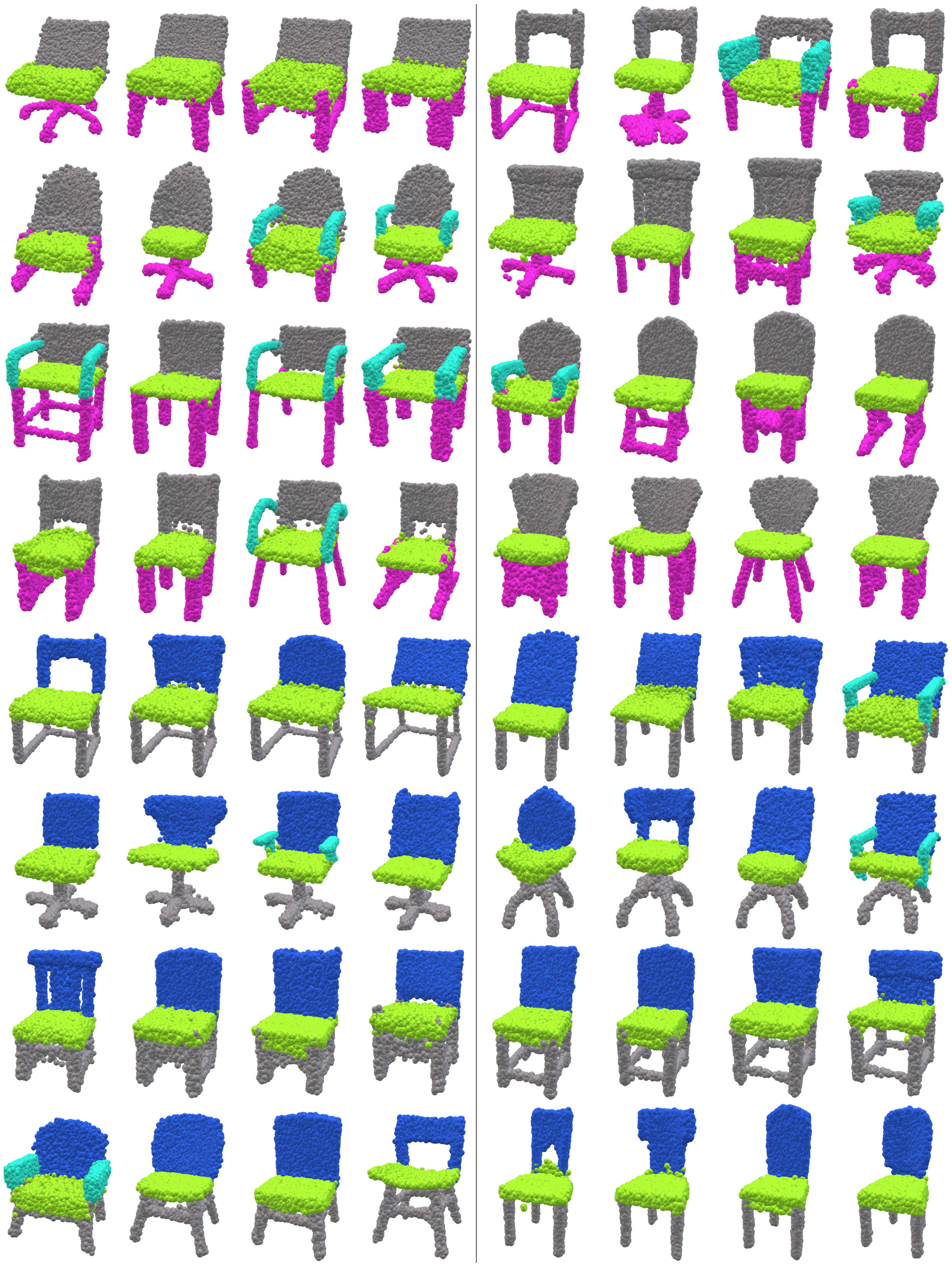


Figure 18: **Part-level Sampling: Chairs, Fixing one part.** Each row shows two sets of chairs (separated by the solid line) generated by sampling the set of part style latents associated with the colored parts while keeping the part style latent for the grey part fixed. Because we model the part configurations independently from their styles, the generated shapes are all plausible and coherent while showing desired controls.

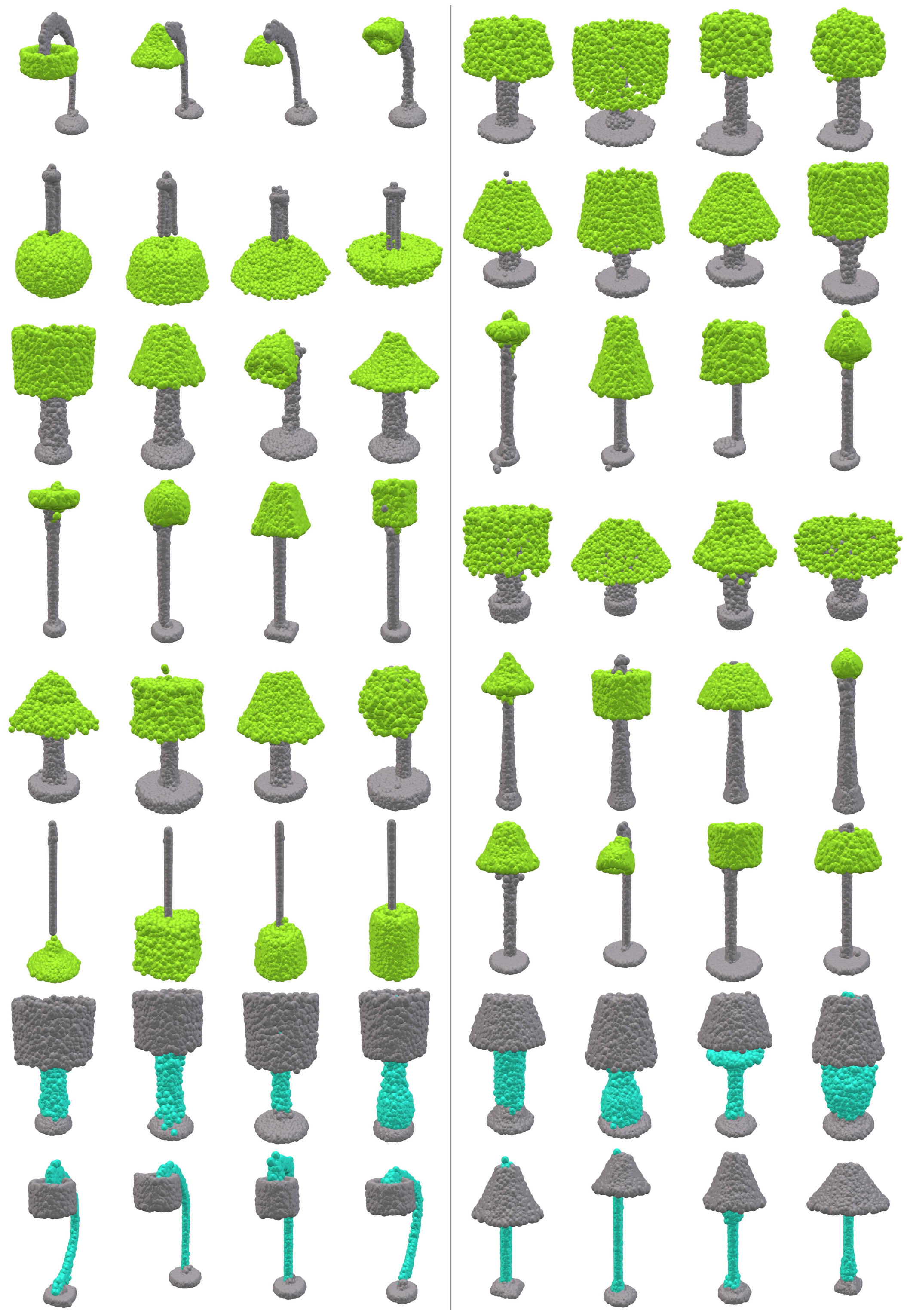


Figure 19: **Part-level Sampling: Lamps, Sampling one part.** Each row shows two sets of lamps (separated by the solid line) generated by sampling the colored part style latent while keeping the other part style latents (colored in grey) unchanged. Notice that the generated samples show diversity in style for colored parts, while the grey part's style stays fixed while changing its size and location to produce coherent shapes.

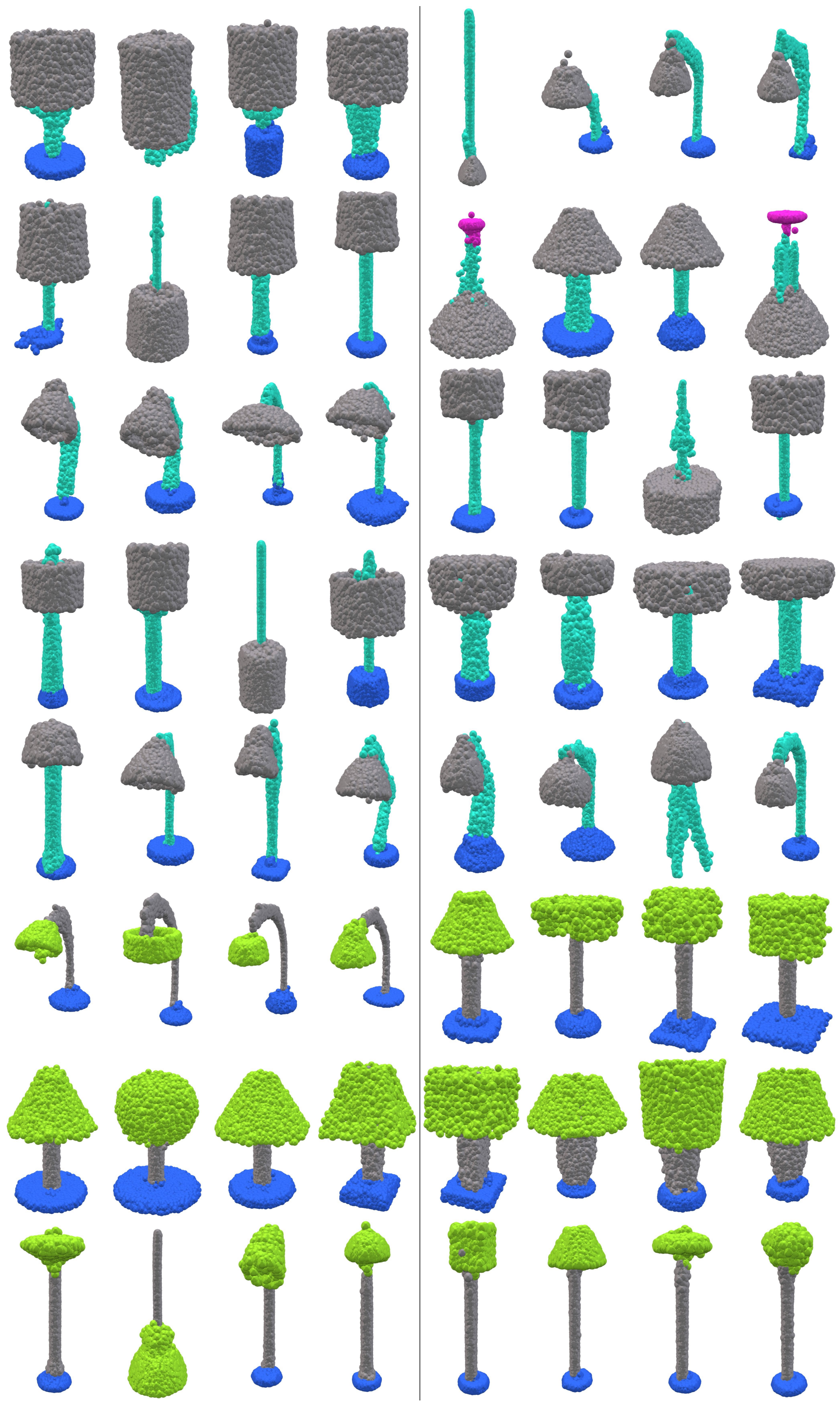


Figure 20: **Part-level Sampling: Lamps, Fixing one part.** Each row shows two sets of lamps (separated by the solid line) generated by sampling the set of part style latents associated with the colored parts while keeping the part style latent for the grey part fixed. Notice that the size and location of the grey part change accordingly to fit the different samples of part styles of the other parts

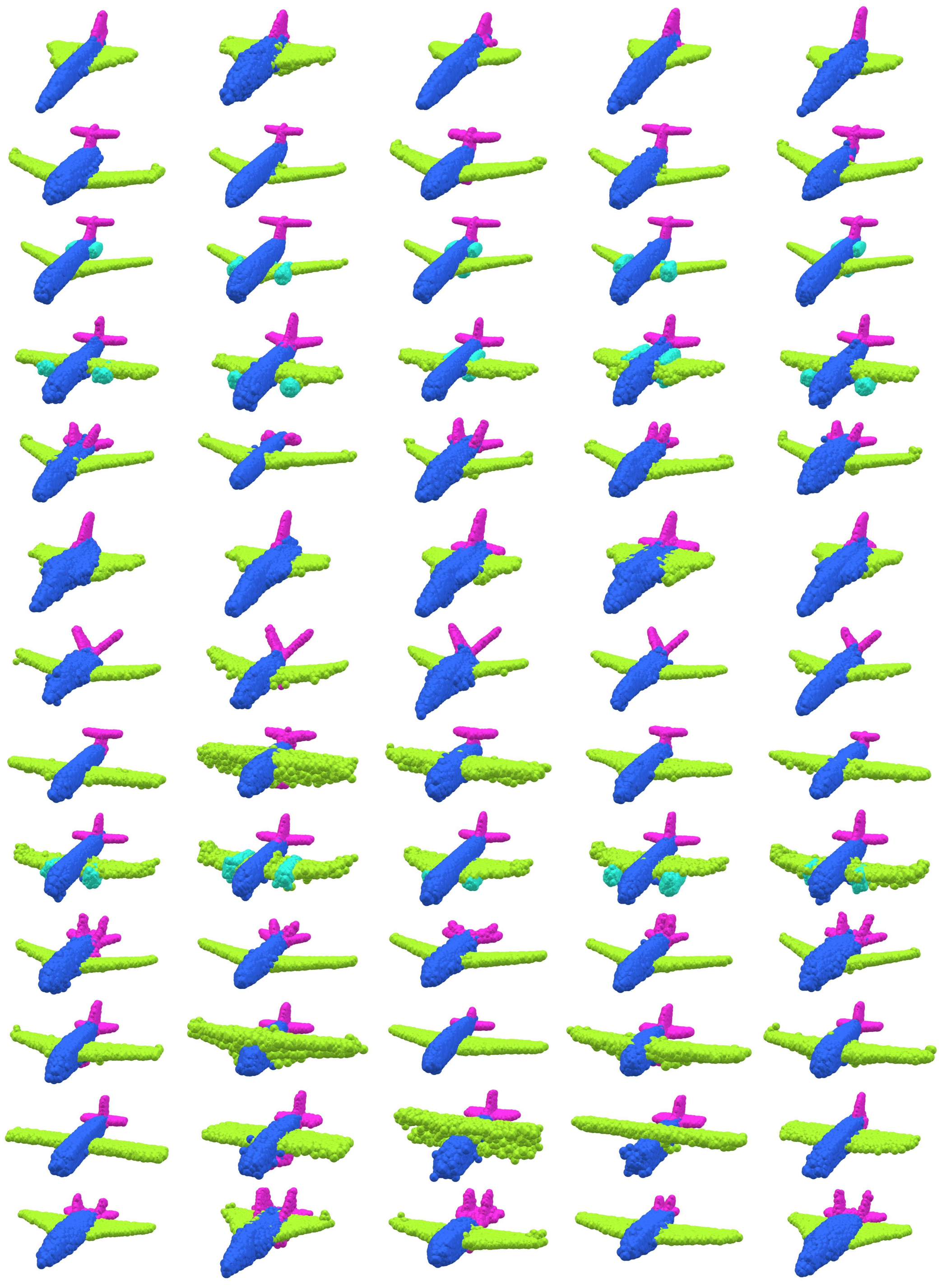


Figure 21: **Part Transformation Sampling: Airplanes.** Each row shows generated airplanes conditioned with different part transformation samples from the transformation sampler. All the part style latent for each row is kept fixed. Notice that the multimodality modeled by cIMLE training method allows for a diverse set of part transformation samples from the transformation sampler.

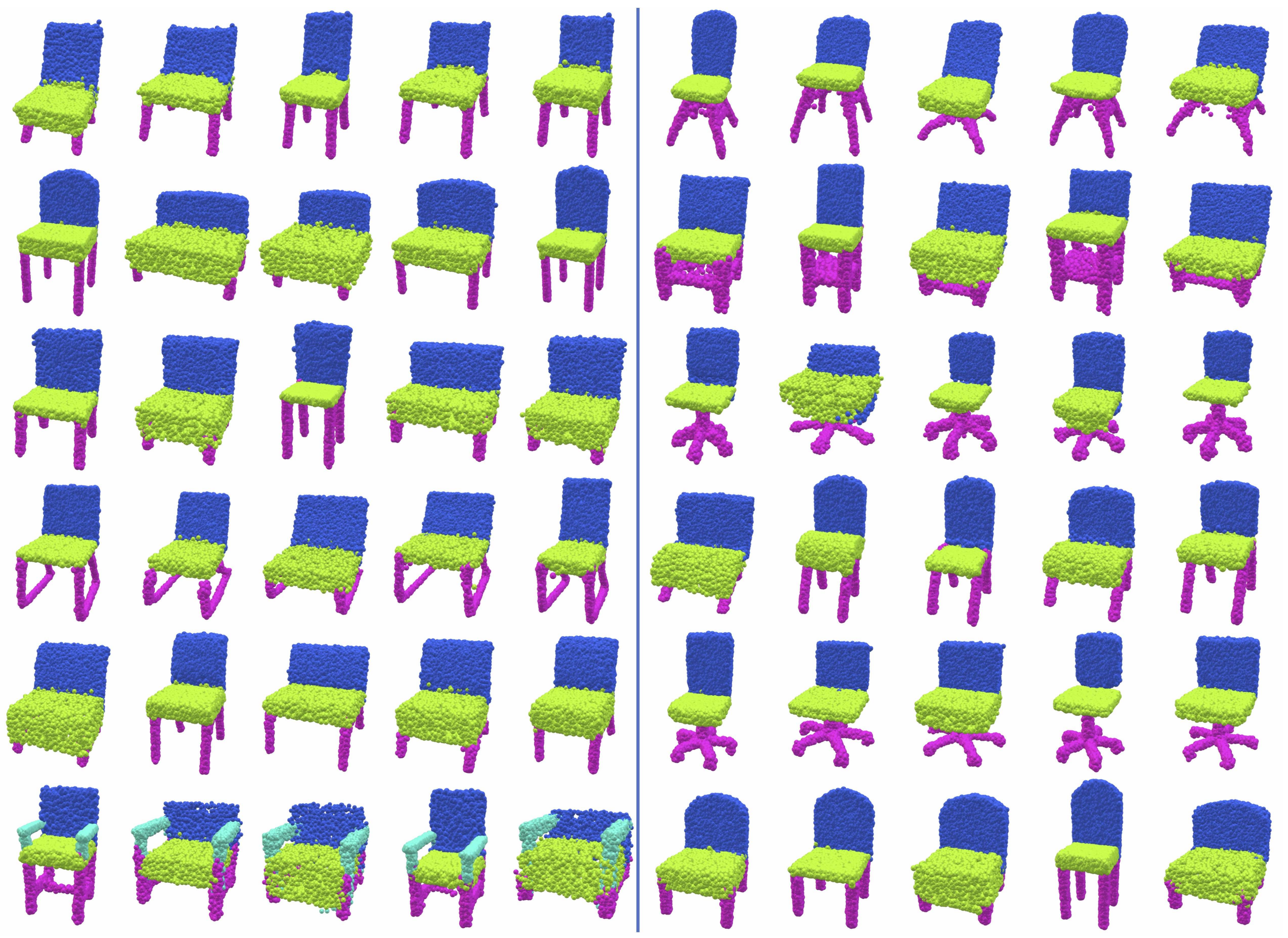


Figure 22: **Part Transformation Sampling: Chairs.** Each row shows generated airplanes conditioned with different part transformation samples from the transformation sampler. All the part style latent for each row is kept fixed.

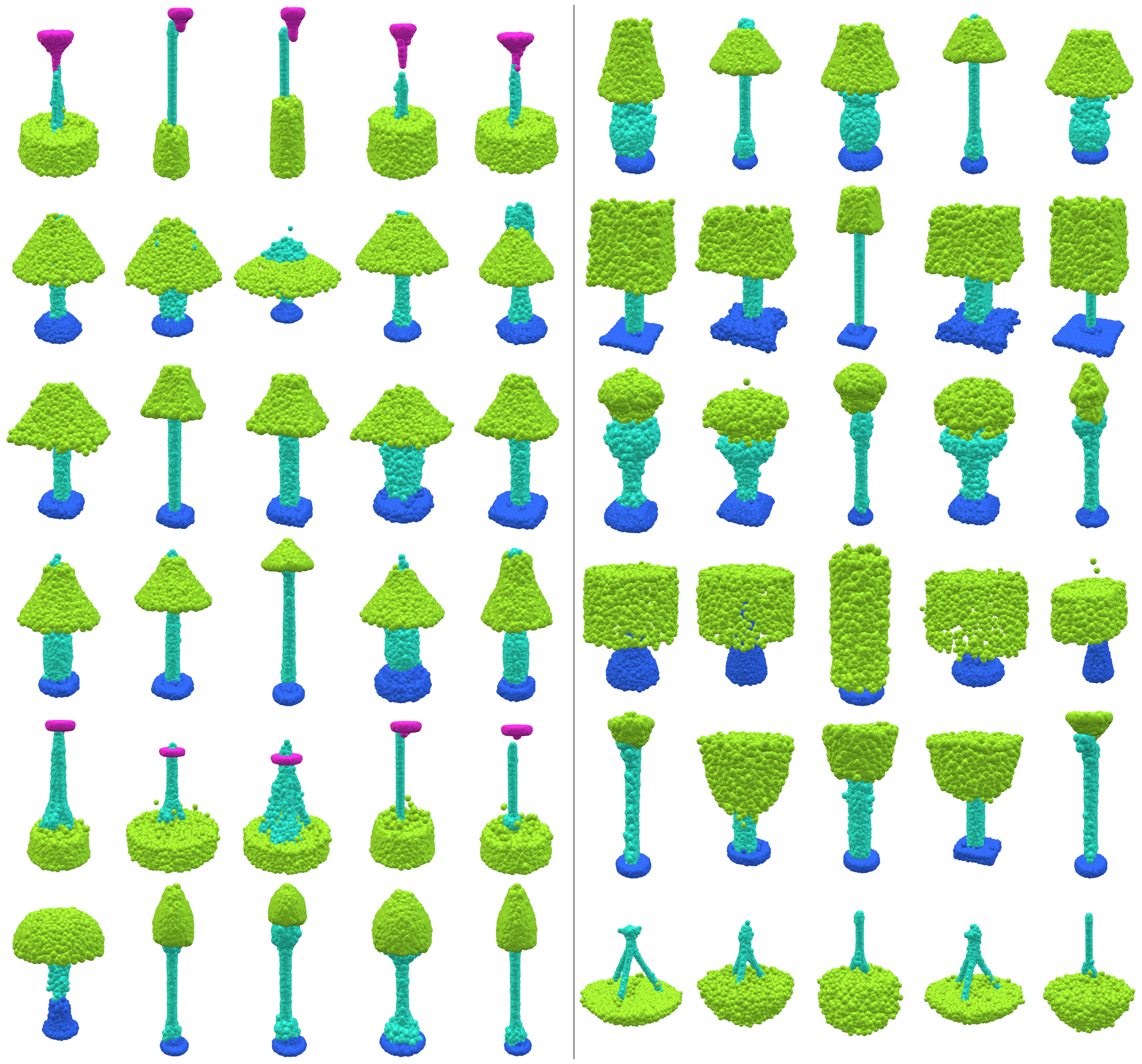


Figure 23: **Part Transformation Sampling: Lamps.** Each row shows generated airplanes conditioned with different part transformation samples from the transformation sampler. All the part style latent for each row is kept fixed.

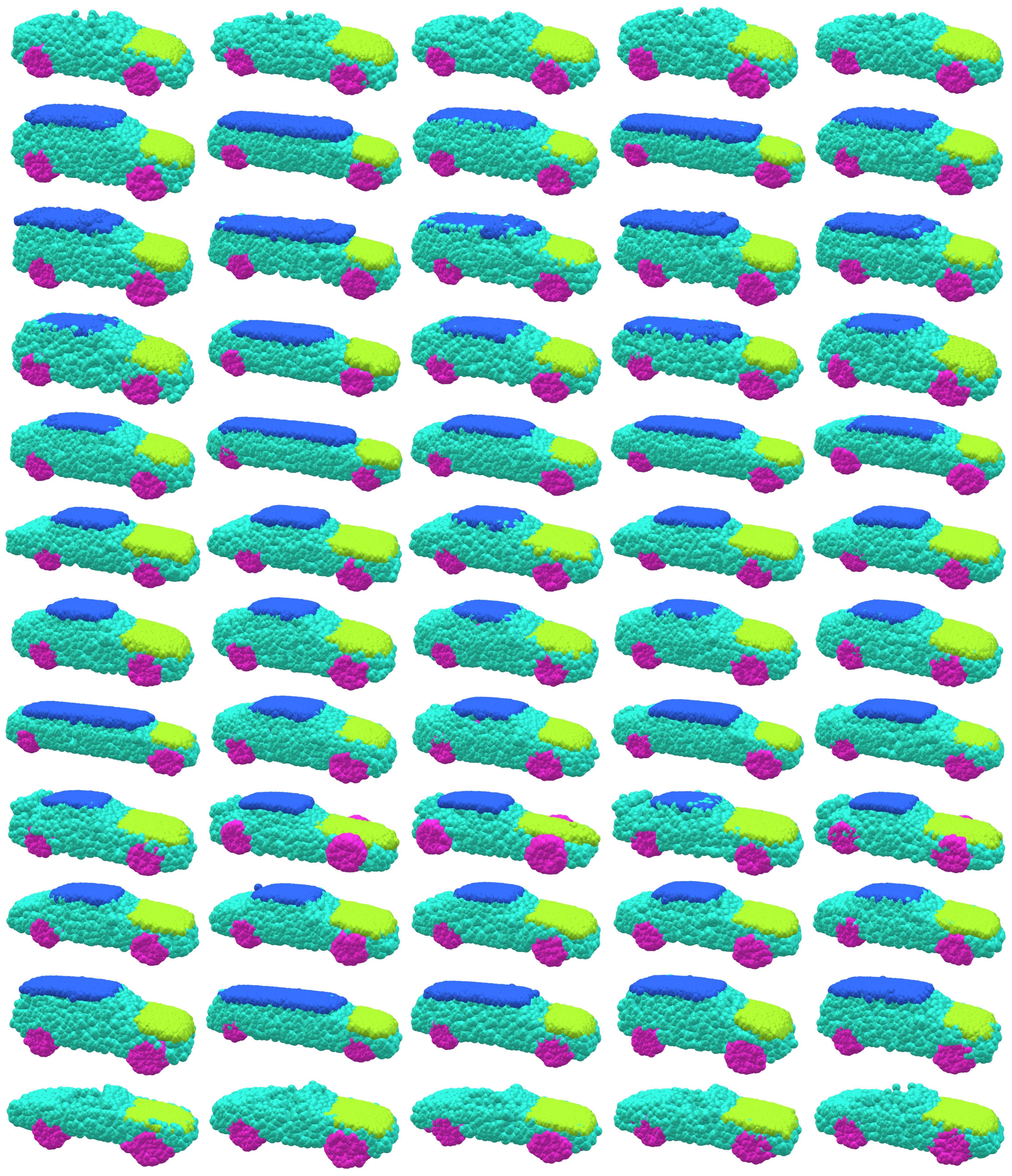


Figure 24: **Part Transformation Sampling: Cars.** Each row shows generated airplanes conditioned with different part transformation samples from the transformation sampler. All the part style latent for each row is kept fixed.

## 4. Network Details

We elaborate on each of our components in this section. Sec. 4.1 presents our part stylizers that learn an independent part style latent space for each part with a continuous normalizing flow model (CNF). Sec. 4.2 elaborates on our transformation sampler that is trained with conditional Maximum Likelihood Estimation (cIMLE) to model multimodality natural to the distribution of valid part configurations. Lastly, Sec. 4.3 details the Cross Diffusion Network using Denoising Diffusion Probabilistic Model (DDPM) with the generalized forward kernel.

### 4.1. Part Style Sampler

To model the part style distribution $P_{\psi_j}(Z_j)$ for each part, we learn a variational encoder $Q_{\varphi_j}(Z_j|\hat{S}_j)$ that models a Gaussian distribution with a learnable mean and diagonal variance given a canonicalized part $\hat{S}_j$. In practice, each part is canonicalized such that $\hat{S}_j$ shifted by the mean and scaled by one standard deviation on each of the axes.

To efficiently learn the variation encoder, we use the reparametrization trick [10] giving us $z_j = \mu_{\varphi_j} + \sigma_{\varphi_j}\varepsilon$ where $\varepsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

Although it is possible to use a standard Gaussian for the prior distribution $P_{\psi_j}(Z_j)$, it has been shown [17, 13, 3] that a simple prior distribution used in VAE models can be less than ideal. To enrich the expressivity of the prior distribution, continuous normalizing flow (CNF) [17] can be used to parameterize the prior distribution as a learnable, invertible network. To this end, we parameterize each per part style distribution $P_{\psi_j}(Z_j)$ as an invertible network and write the KL divergence as [17]

$$\mathrm{KL}\left(P_{\psi_j}(z_j) \| Q_{\varphi_j}\left(z_j|\hat{S}_j\right)\right) = -\mathbb{E}_{z_j \sim Q_{\varphi_j}(Z_j|\hat{S}_j)}\left[\log P_{\psi_j}\left(z_j\right)\right] - H\left(Q_{\varphi_j}\left(Z_j|\hat{S}_j\right)\right). \tag{6}$$

for each part $j = 1, \ldots, m$ and for each shape $S$. Here $H$ is the entropy and $P_{\psi_j}\left(Z_j\right)$ is the learnable part style prior distribution obtained by transforming a standard Gaussian with an invertible network:

$$z_j := F_{\psi_j}\left(\xi(t_0)\right) = \xi(t_0) + \int_{t_0}^{t_1} f_{\psi_j}(\xi(t), t)\, dt$$

where $\xi(t_0) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and $f_{\boldsymbol{\psi}}$ is the continuous-time dynamics of the flow $F_{\psi_j}$. Then, the log probability can be calculated by

$$\log P_{\psi_j}\left(z_i\right) = \log P\left(F_{\psi_j}^{-1}\left(z_j\right)\right) + \int_{t_0}^{t_1} \operatorname{tr}\left(\frac{\partial f_{\psi_j}}{\partial \xi(t)}\right) dt.$$

We finally note that for shapes with missing parts, we replace its part style latent with a dummy latent variance, and mask it out at training time.

### 4.2. Transformation Sampler

As detailed in Sec. 4.2 our the main paper, we train an implicit probabilistic model with conditional Impicit Maximum Likelihood (cIMLE) method. Specifically, we model a multimodal distribution of part transformations $P_\theta(\boldsymbol{T}|\boldsymbol{z})$ given the part style latents by a deep neural network $T_\theta : (\boldsymbol{z}, y) \mapsto \boldsymbol{\tau}$ where $\boldsymbol{z} \sim P_{\boldsymbol{\psi}}(\boldsymbol{Z})$ and $y \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, which is a $32-$dimensional noise latent code. By sampling different noise latent code from the standard Gaussian, the neural network $T_\theta$ is able to produce different valid modes of part transformations for the given set of part style latents. To train the implicit model with cIMLE, we follow [11] and cache the best fitting noise latent code for each set of part style latents $\mathbf{z}$ during training, and optimize the transformation sampler using the cached noise codes for a number of iterations. The best fitting is defined by the cIMLE objective

$$\ell_{\boldsymbol{\tau}} = \sum_{S \in \mathbf{S}} \min_{k=1,\ldots,K} \ell_{\text{fit}}\left(T_\theta\left(\boldsymbol{z}_S, y_k\right), \boldsymbol{\tau}_S\right), \tag{7}$$

where $\boldsymbol{\tau}_S$ is the observed part transformations for shape $S$. $\ell_{\text{fit}}$ is defined to be the distance between the generated transformation $\{\boldsymbol{\tau}_k\}$ and observation $\boldsymbol{\tau}_S$ summed across all parts:

$$\ell_{\text{fit}}\left(\boldsymbol{\tau}, \boldsymbol{\tau}_S\right) = \sum_{j=1}^{m} \|c_j - c_{S,j}\|_2^2 + \|\log s_j - \log s_{S,j}\|_2^2.$$

Same as in the main paper, $c_j, c_{S,j} \in \mathbb{R}^3, s_j, s_{S,j} \in \mathbb{R}^3$ are the shifts and scales associated with $j$-th part of the generated transformation and the observed transformation respectively.

## 4.3. Cross Diffusion Network

To learn the data distribution $P_\phi(S|\boldsymbol{z}, \boldsymbol{\tau})$, we learn a point distribution $P_\phi(X^{(0)}|\boldsymbol{z}, \boldsymbol{\tau}, j)$ for each part conditioned on the part style latents $\boldsymbol{z}$ and the part transformations $\boldsymbol{\tau}$. For each observed shape $S$, represented as a point cloud, we assume that points from each part are sampled independently from $P_\phi(X|\boldsymbol{z}, \boldsymbol{\tau}, j)$. We define a forward diffusion process $Q(X^{(0:T)}|\tau_j)$ for each part using the generalized forward kernel with $\mu = c_j$ and $\Sigma = \mathrm{Diag}(s_j)$ for each part. To learn the denoising process, we define a reverse process as another Markov chain $P_\phi(X^{(0:T)}|\boldsymbol{z}, \boldsymbol{\tau}, j)$ for each part such that

$$P_\phi(X^{(0:T)}|\boldsymbol{z}, \boldsymbol{\tau}, j) = P(X^{(T)}|\tau_j) \prod_{t=1}^{T} P\phi(X^{(t-1)}|X^{(t)}, \boldsymbol{z}, \boldsymbol{\tau}, j)$$

where

$$P(X^{(T)}|\tau_j) = \mathcal{N}\left(c_j, \mathrm{Diag}\, s_j\right)$$

and each reversion kernel is parameterized as a Gaussian Distribution with learnable means to approximate the posterior distribution $q\left(X^{(t-1)}|X^{(t)}, X^{(0)}, c_j, \mathrm{Diag}(s_j)\right)$:

$$P_\phi(X^{(t-1)}|X^{(t)}, \boldsymbol{z}, \boldsymbol{\tau}, j) = \mathcal{N}\left(\Xi_\phi\left(t, X^{(t)}, \boldsymbol{z}, \boldsymbol{\tau}, j\right), \eta_t^2\, \mathrm{Diag}\left(s_j\right)\right).$$

Following [13, 20, 14, 19], instead of directly learning the mean $\Xi_\phi$, we reparameterize the forward posterior distribution according to Eq. 5 to learn $\boldsymbol{\varepsilon}_\phi$ that approximate noises from standard Gaussian. Thus, the final objective becomes

$$\ell_{\text{cross}} = \sum_{j=1}^{m} \sum_{x \in S_j} \mathbb{E}_{\varepsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \boldsymbol{z} \sim Q_{\boldsymbol{\varphi}}, t \in \{1, \ldots, T\}} \left[\left\|\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_\phi\left(x^{(t)}, \boldsymbol{z}, \boldsymbol{\tau}, j, t\right)\right\|_2^2\right]. \tag{8}$$

# 5. Implementation Details

In this section, we provide the implementation details: our training pipeline and hyperparameters, network architecture, and training and sampling algorithms.

## 5.1. Training Details

As detailed in the main paper, our training loss is transformed to $\ell_{\text{total}} = \ell_{\text{recon}} + \lambda_1 \ell_{\mathbf{Z}} + \lambda_2 \ell_{\boldsymbol{\tau}}$. In practice, we adopt a two-stage training strategy. During the first stage, we train the prior flow $P_{\boldsymbol{\psi}}(Z_j)$, variational encoders $Q_{\boldsymbol{\varphi}}(\boldsymbol{Z}|\hat{S})$, and the cross diffusion network $P_\phi(X|\boldsymbol{Z}, \boldsymbol{\tau}_S, j)$. For the Cross Diffusion Network, We use timestep $T = 100$ for all of our models and the variance scheduling parameters $\alpha_t$ are set to linearly decrease from $0.9999$ to $0.08$ following the setting of [13]. Ground truth transformations are used to condition the Cross Diffusion network during the first stage. The loss optimized for the first stage of training is

$$\begin{aligned}\ell_{\text{first stage}} &= \lambda_1 \ell_{\mathbf{Z}} + \ell_{\text{recon}} \\ &= \mathbb{E}_{S \sim P(S)}\Bigg[ - \lambda_1 \mathbb{E}_{z_j \sim Q_{\varphi_j}(Z_j|\hat{S}_j)}\left[\log P_{\psi_j}\left(z_j\right)\right] - H\left(Q_{\varphi_j}\left(Z_j|\hat{S}_j\right)\right) \\ &\quad + \frac{1}{m}\sum_{j=1}^{m} \frac{1}{|S_j|} \sum_{x \in S_j} \mathbb{E}_{\varepsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \boldsymbol{z} \sim Q_{\boldsymbol{\varphi}}, t \in \{1, \ldots, T\}} \left[\left\|\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_\phi\left(x^{(t)}, \boldsymbol{z}, \boldsymbol{\tau}_S, j, t\right)\right\|_2^2\right]\Bigg].\end{aligned} \tag{9}$$

where $\lambda_1$ =5e-4 for chairs and $\lambda_1$ =1e-3 for lamps, airplanes, and cars. For all models, we use a batch size of 128, and the first stage is trained for 8000 epochs using Adam [9] optimizer without weight decay. Momentum parameters $\beta_1$ and $\beta_2$ for Adam optimizer are set to 0.9 and 0.999 respectively. The learning rate is set to 2e-3 at the start of training as is linearly decreased starting from 4000 epochs to 1e-4 at the end of training. The first stage takes around 36 hours for chairs, 30 hours for airplanes, 24 hours for lamps, and 16 hours for cars.

In the second stage, we freeze the weights of the part stylizers and the cross diffusion network, and only train the transformation sampler using the cIMLE training strategy. The second stage loss is the cIMLE loss plus the reconstruction loss:

$$\ell_{\text{second stage}} = \ell_{\text{recon}} + \lambda_2 \ell_{\boldsymbol{\tau}}, \tag{10}$$

where $\lambda_2 = 1$ for all classes. The training of cIMLE requires recaching of noise latent codes. Specifically, we recache the noise latent codes for each shape in our training data every 50 epoch, during which we sample $K = 20$ noise latent code and associate each set of part style latents $\boldsymbol{z}$ with a best fitting code according to Eq. 7. These pairs are then trained for 50 epochs before the noise codes are recached. The second stage is trained for a maximum of 4000 epochs and the models used for evaluation are selected from the best checkpoint. Similar to the first stage, we use the Adam optimizer, and keep the learning rate fixed at 2e-4 for the entire training process.

## 5.2. Network Parameters

### 5.2.1 Part Style Sampler

Each variation encoder $Q_{\varphi_j}(Z_j|\hat{S}_j)$ is implemented as a PointNet [2] following the architecture of [13, 1] with a shared per point feature regression layer. Specifically, we feed the canonicalized parts through a 3-128-256-512 MLP layer with ReLU nonlinearity and batch normalization. We share the per-point MLP across all parts to reduce parameter count. The per-point features are then max-pooled for each part to obtain a 512 dimensional feature for each part. Then, the feature is fed into a 512-256-128-256 MLP with the ReLU nonlinearity to output a 256 dimensional mean and a diagonal variance which parameterize the Gaussian distribution of $Q_{\varphi_j}(Z_j|\hat{S}_j) = \mathcal{N}\left(\mu_{\varphi_j}\left(\hat{S}_j\right), \sigma_{\varphi_j}\left(\hat{S}_j\right)\right)$. We note that the MLP after max pooling is not shared across different parts as the objective decomposition assumes independence of part style latent distributions. The 256 dimensional part style latent is then sampled from this Gaussian for downstream training.

Because each part style latent distribution is independent of the other, we use one prior flow network for each part. Each prior flow is implemented with the same architecture as [13, 17]. Specifically, we use 14 affine coupling layers with the dimension of hidden states being 256, identical to the dimension of part style latents. Following each of the layers, we apply moving batch normalization [8, 4]. Both the scaling and translation networks $F(\cdot)$ and $G(\cdot)$ are 128-256-256-128 MLPs with ReLU nonlinearity.

### 5.2.2 Transformation Sampler

The network for transformation sampler $T_\theta$ is implemented using a self-attention-only transformer following [14, 16]. The input is $m$ tokens with each token being the concatenation of the part style latent $z_j \in \mathbb{R}^{256}$ and a scaled noise latent code $\lambda y \in \mathbb{R}^{32}$. The $\lambda$ parameter is class specific and controls the degree of transformation variations produced by different modes of the transformation sampler. For the reported models, we use $\lambda = 100$ for chairs, $\lambda = 50$ for airplanes and cars, and $\lambda = 10$ for lamps. Then, each token is projected back to a 256 dimensional vector and the part label associated with each token is added as a learnable embedding vector to the projected token. For the transformer architecture, we use 5 layers of multi-head self-attention followed by a feed-forward layer and layer normalization. Parameters can be found in Tab. 4. The output of $T_\theta$

| parameters | Values |
|---|---|
| layers | 5 |
| head dimension | 32 |
| num heads | 8 |
| drop out rate | 0 |

Table 4: Parameters for Transformation Sampler.

is a three-dimensional scale and shift vector for each part style latent. For missing parts, we mask out its associated token during the self-attention layers.

### 5.2.3 Cross Diffusion Network

As derived in Eq (5), we learn a noise approximator $\varepsilon_\phi$ that estimates pure noises given noisy inputs at time step $t$. We use a cross-attention-only transformer to implement $\varepsilon_\phi$. Specifically, the input consists of $N$ tokens where $N$ is the number of points in a shape $S$. Each input token is the concatenation of $(x^{(t)}, \tau_j, j)$ if the point $x^{(t)}$ belongs to the $j$-th part. Because of computation cost and independence assumption across different point samples, the input tokens are first projected to 128 dimensional vectors and then individually attend to $m$ context tokens each being the concatenation of $(z_j, \tau_j, j, t)$, for $j = 1, ..., m$ followed by a feedforward layer and layer normalization. The transformer architecture is similar to that of the transformation sampler, with parameters listed in Tab. 5.

| parameters | Values |
|---|---|
| layers | 5 |
| head dimension | 16 |
| num heads | 8 |
| drop out rate | 0.2 |

Table 5: Parameters for Cross Diffusion Network.

### 5.3. Algorithm

We provide the training and sampling algorithms for DiffFacto in Algorithm 1 (first stage training), Algorithm 2 (second stage training), and Algorithm 3 (sampling).

**Algorithm 1** DiffFacto Training: Stage 1

```
repeat
    Let $S \in \boldsymbol{S}_{\text{data}}$
    for $j = 1, \ldots, m$ do
        if $S_j \neq \varnothing$ then
            Sample $z_j \sim Q_{\varphi_j}\left(Z_j | \hat{S}_j\right)$.
        else
            $z_j = z_{\text{dummy}}$.
        end if
    end for
    $\boldsymbol{z} \leftarrow \{z_j\}_{j=1}^m$
    Let $\boldsymbol{\tau}_S$ to be the part transformations for $S$.
    Sample $t \sim$ Uniform $(\{1, \ldots, T\})$
    Sample $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    Compute $\nabla_{\phi,\boldsymbol{\varphi},\psi}[\ell_{\text{first stage}}]$; then perform gradient descent.
until converged.
```

## 6. Experiment Set-up Additional Details

### 6.1. Control-Enabled Baselines Additional Details

Because none of the existing networks enable part-level generation, we modify two state-of-the-art baselines (LION [19] and ShapeGF [1]) to enable part-level sampling for a fair comparison with DiffFacto, we refer to these control-enabled modification as Ctrl-LION and Ctrl-ShapeGF. To enable part-level generation, we train their method per part, *i.e*. one model for each part of an object class. We use the same dataset as our method, namely ShapeNet with semantic segmentation labels. As a result, we obtain a generative model for each part for both ShapeGF and LION networks. Then, to obtain the global shape, we simply sample from each of the perpart Ctrl-ShapeGF and Ctrl-LION and concatenate the points together. To sample on the part level, we simply sample from a specific part latent space while fixing the rest of the generated shapes.

### 6.2. Evaluation Protocol Additional Details

We elaborate on the evaluation protocol we use to compare with baselines here. Sec. 6.2.1 details the intro-part evaluation procedure, which we use to evaluate the quality and diversity of our per-part distributions. Sec. 6.2.2 elaborate on the snapping metric, which we use to measure against the control-enabled baselines. Sec. 6.2.3 details the human study that was conducted.

**Algorithm 2** DiffFacto Training: Stage 2

```
repeat
    for $S \in \boldsymbol{S}_{\text{data}}$ do
        Let $\boldsymbol{\tau}_S$ to be the part transformations for $S$.
        for $j = 1, \ldots, m$ do
            if $S_j \neq \varnothing$ then
                Sample $z_j \sim Q_{\varphi_j}\left(Z_j | \hat{S}_j\right)$.
            else
                $z_j = z_{\text{dummy}}$.
            end if
        end for
        $\boldsymbol{z} \leftarrow \{z_j\}_{j=1}^{m}$
        Sample $y_1, \ldots, y_K \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
        Set $y_S^* = \arg\min_{y_i \in \{y_1, \ldots, y_K\}} \ell_{\text{fit}}\left(T_\theta(\boldsymbol{z}, y_i), \boldsymbol{\tau}_S\right)$.
    end for
    for $l = 1, \ldots, 50$ do
        Sample $S \in \boldsymbol{S}_{\text{data}}$
        for $j = 1, \ldots, m$ do
            if $S_j \neq \varnothing$ then
                Sample $z_j \sim Q_{\varphi_j}\left(Z_j | \hat{S}_j\right)$.
            else
                $z_j = z_{\text{dummy}}$.
            end if
        end for
        $\boldsymbol{z} \leftarrow \{z_j\}_{j=1}^{m}$
        Let $\boldsymbol{\tau}_S$ to be the part transformations for $S$.
        Sample $t \sim \text{Uniform}(\{1, \ldots, T\})$
        Sample $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
        Compute $\nabla_\theta \left[\ell_{\text{recon}} + \ell_{\text{fit}}(T_\theta(\boldsymbol{z}, y_S^*), \boldsymbol{\tau}_S)\right]$; then perform gradient descent.
    end for
until converged.
```

### 6.2.1 Intra-part Evaluation Additional Details

As presented in the main paper, we use the standard generation metrics to measure the similarity between the distributions of canonicalized parts of the generated shapes compared to the test set from [18] of segmented shapes. For a set of canonicalized part $\boldsymbol{S}_{t,j}$ from the test set and a set of canonicalized part $\boldsymbol{S}_{g,j}$ from the generated set, the generation metrics are computed as
**Minimum matching distance (MMD-P)**

$$\text{MMD-P}(\boldsymbol{S}_{g,j}, \boldsymbol{S}_{t,j}) = \frac{1}{|\boldsymbol{S}_{t,j}|} \sum_{S_{t,j} \in \boldsymbol{S}_{t,j}} \min_{S_{g,j} \in \boldsymbol{S}_{g,j}} \text{Chamfer}\left(S_{g,j}, S_{t,j}\right).$$

The idea behind MMD-P is to calculate the average distance between the point clouds in the reference set and their closest neighbors in the generated set. A smaller MMD-P implies that the parts from the test set are well represented in the parts from the generated set.
**Coverage (COV-P)**

$$\text{COV-P}(\boldsymbol{S}_{g,j}, \boldsymbol{S}_{t,j}) = \frac{\left|\left\{\arg\min_{S_{t,j} \in \boldsymbol{S}_{t,j}} \text{Chamfer}\left(S_{t,j}, S_{g,j}\right)\right\}\right|}{|S_{t,j}|}.$$

A higher COV-P percentage means that parts in the test set are well covered by the parts in the generated set, so it implies that the generated parts are diverse.

**Algorithm 3** DiffFacto Sampling

```
for $j = 1, \ldots, m$ do
    Sample $\xi \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.
    $z_j = F_{\psi_j}(\xi(t_0))$
end for
$\boldsymbol{z} \leftarrow \{z_j\}_{j=1}^m$
Sample $y \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
Compute $\boldsymbol{\tau} = \{(c_j, s_j)\}_{j=1}^m = T_\theta(\boldsymbol{z}, y)$
for $j = 1, \ldots, m$ do
    $S_j^{(T)} = \left\{x^{(T)}\right\} \sim \mathcal{N}(c_j, \mathrm{Diag}(s_i))$
end for
$S^{(T)} \leftarrow \left\{S_j^{(T)}\right\}_{j=1}^m$
for $t = T, \ldots, 1$ do
    for $i = 1, \ldots, m$ do
        for $x^{(t)} \in S_j^{(t)}$ do
            $x^{(t-1)} \sim P_{\boldsymbol{\theta}}\left(X^{(t-1)} | x^{(t)}, \boldsymbol{z}, \boldsymbol{\tau}, j\right)$
        end for
        $S_i^{(t-1)} \leftarrow \left\{x^{(t-1)}\right\}$
    end for
    $S^{(t-1)} \leftarrow \left\{S_j^{(t-1)}\right\}_{j=1}^m$
end for
return $S^{(0)}$
```

**1-NN classifier accuracy (1NNA-P)**

$$\text{1NNA-P}(\boldsymbol{S}_{g,j}, \boldsymbol{S}_{t,j}) = \frac{\sum_{S_{g,j} \in \boldsymbol{S}_{g,j}} \mathbb{1}[N_{S_{g,j}} \in \boldsymbol{S}_{g,j}] + \sum_{S_{t,j} \in \boldsymbol{S}_{t,j}} \mathbb{1}[N_{S_{t,j}} \in \boldsymbol{S}_{t,j}]}{|S_{t,j}| + |S_{g,j}|}.$$

where $\mathbb{1}[\cdot]$ is the indicator function and $N_{S_j}$ is the nearest neighbor of $S_j$ in the set $S_{t,j} \cup S_{g,j} \setminus \{S_j\}$ (i.e., the union of the sets $S_{g,j}$ and $S_{t,j}$ excluding, $S_j$). With this definition, 1NNA-P represents the leave-one-out accuracy of the 1-NN classifier (measured in Chamfer distance) defined above. If the performance of the 1NN classifier is close to 50 percent, the parts from the test set and the generated set are well mingled. Thus, 1NNA-P directly measures the similarity between the test parts and the generated parts, both in diversity and quality.

To prepare for the sets of canonicalized parts, we use 512 points for each of the part samples for the test set. For our model and all the baselines, we sample 2048 points on the entire shape and obtain for each shape its canonicalized parts from these points. We sample the same number of shapes as in the test set. Since the baselines do not generate shapes with part segmentation, we enable their evaluation using a segmentor with PointNet++ backbone [15] trained on the ShapeNetCore PartSeg dataset. Then, MMD-P. COV-P. and 1NNA-P is computed for each part and in the end, the score is reported as a weighted average based on the number of part samples in each of the part sets of the test shapes. For implicit methods, we sample 2048 points from their generated mesh. The canonicalization of each part is done by fitting all three axes into the unit cube.

### 6.2.2 Inter-Part Evaluation Additional Details

To measure the connectivity of the generated shapes, we select a set of connected parts and compute the Chamfer distance between the closest $N_{\text{SNAP}} = 30$ points between the two parts. Specifically, if the $j$-th part is connected to a set of parts $\{S_k\}$ (note that a part can be connected to multiple parts due to different structures of shapes), we define the snapping distance of connection $j$ to be

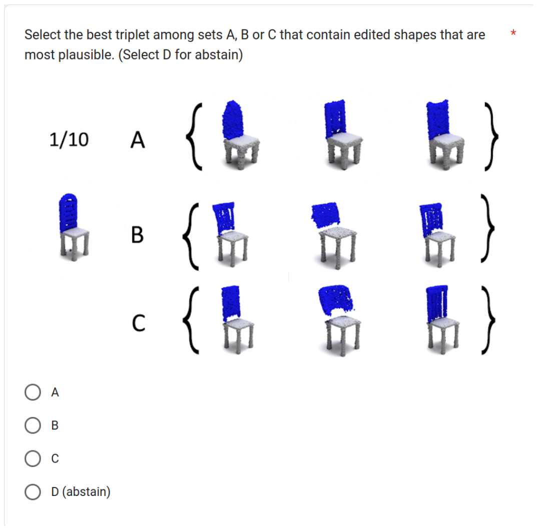
$$\text{SNAP}(S_j) = \min_{S_k \in \{S_k\}} \text{Chamfer}\left(N_{S_k}^{(N_{\text{SNAP}})}(S_j), N_{S_j}^{(N_{\text{SNAP}})}(S_k)\right),$$

where $N_X^{(N_{\text{SNAP}})}(Y)$ are the $N_{\text{SNAP}}$ closest points in shape $Y$ to shape $X$.

For chairs, we define connections of the back, seat, and arms respectively. The back connection is defined to *connect to either the legs or the seat*; the seat connection is defined to be *connected to the legs*, and lastly, the arm connection is defined to *connect to either the back or the seat*.

### 6.2.3 Human Study Additional Details

We provide further details on the user study we conducted to evaluate controllability of our method compared to the control-enabled baselines (Ctrl-ShapeGF and Ctrl-LION). A user is asked to select the triplet of edited shapes that are most plausible, where examples of plausible shapes to be: 1) does not have detachment of parts, 2) does not have interpenetrating parts, 3) the size of the edited parts agrees with the rest of the shape, which was included in the prompt. As described in the main paper, a (controlled) edit is defined as re-sampling a pre-selected part of the given shape. Our user study contained 10 questions, which are randomly selected shapes from the dataset, each with random parts to edit. Fig. 25 shows one example question from our user study.



Select the best triplet among sets A, B or C that contain edited shapes that are most plausible. (Select D for abstain) *

1/10 A { } B { } C { }

- A
- B
- C
- D (abstain)

Figure 25: **User Study Example.** The subject is asked ten questions similar to the example above in which they are asked to pick the most coherent group out of the three.

## References

[1] Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. In *ECCV*, 2020. 4, 5, 32, 33

[2] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017. 32

[3] Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder, 2016. 30

[4] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models, 2018. 32

[5] Amir Hertz, Or Perel, Raja Giryes, Olga Sorkine-Hornung, and Daniel Cohen-Or. Spaghetti: Editing implicit shapes through part aware generation. In *SIGGRAPH Asia*, 2022. 4

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*, 2020. 3

[7] Ka-Hei Hui, Ruihui Li, Jingyu Hu, and Chi-Wing Fu. Neural wavelet-domain diffusion for 3d shape generation, 2022. 4

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. 32

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. 31

[10] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. 30

[11] Ke Li and Jitendra Malik. Implicit maximum likelihood estimation. *arXiv preprint arXiv:1809.09087*, 2018. 30

[12] Ruihui Li, Xianzhi Li, Ka-Hei Hui, and Chi-Wing Fu. Sp-gan: Sphere-guided 3d shape generation and manipulation. *ACM Transactions on Graphics (TOG)*, 40(4):1–12, 2021. 4

[13] Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation. In *CVPR*, June 2021. 4, 30, 31, 32

[14] Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-e: A system for generating 3d point clouds from complex prompts. *arXiv preprint arXiv:2212.08751*, 2022. 31, 32

[15] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017. 35

[16] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021. 32

[17] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *ICCV*, 2019. 30, 32

[18] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *SIGGRAPH Asia*, 2016. 34

[19] Xiaohui Zeng, Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, and Karsten Kreis. Lion: Latent point diffusion models for 3d shape generation. In *NeurIPS*, 2022. 5, 31, 33

[20] Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5826–5835, October 2021. 31