Learning Gradient Fields for Shape Generation – Supplementary Material –

Contents

1 Method Details **1**
2 Implementation Details **5**
3 Additional Quantitative Results **7**
4 Additional Ablation Studies **10**
5 Additional Qualitative Results **10**

1 Method Details

In this section, we provide extensive details of our method. We provide detailed explanations for training and inference in Section 1.1. More details about surface extraction are provided in Section 1.2. Finally, a mathematical proof about our objective function is shown in Section 1.3.

1.1 Training and inference

We provide algorithm blocks to better illustrate the training and inference procedures. See Algorithm 1 and Algorithm 2 for training and inference, respectively. Please refer to Section 2 for hyper-parameters and neural network architectures.

1.2 Surface extraction

We use a modified version of the volumetric ray casting algorithm [12] to render the iso-surface produced by the learned gradient field. Algorithm 3 below shows the rendering process of a single pixel. For each pixel in the rendered image, we cast a ray $\langle o_0, u \rangle$ towards the gradient field according to the camera model. We advance the ray for a fixed number of steps $k_{\max}$. For reasonable choices of the step rate γ, the ray will either converge to the iso-surface, or miss the iso-surface and march towards infinity. If the ray does reach the iso-surface, we calculate the RGB values for that pixel based on its 3D location and surface normal. If the ray misses the surface, we assign the pixel with background color y_{bg}. In this paper, we use $\gamma = 1.0$, $k_{\max} = 64$ and $\delta = 0.005$.

Algorithm 1 Training.

Require: Noise levels $\{\sigma_i\}_{i=1}^k$; Weight for noise levels' loss $\lambda(\sigma_i)$; Point cloud encoder f_ϕ; A neural network s_θ; Total number of training iterations T; Point cloud X_t

for $t \leftarrow 1$ to T **do**
  $z \leftarrow f_\phi(X_t)$
  for $\sigma \in \{\sigma_i\}_{i=1}^k$ **do**
    for $x_i \in X_t$ **do**
      $\tilde{x}_i \leftarrow x_i + N(0, \sigma^2 I)$
    end for
    $\ell(\sigma, X_t) \leftarrow \frac{1}{|X_t|} \sum_{x_i \in X_t} \| g_\theta(\tilde{x}_i, \sigma, z) - \frac{x_i - \tilde{x}_i}{\sigma^2} \|_2^2$
  end for
  $\mathcal{L}(\{\sigma_i\}_{i=1}^k, X_t) \leftarrow \sum_{i=1}^k \lambda(\sigma_i) \ell(\sigma_i, X_t)$
  $\phi, \theta \leftarrow Adam(\mathcal{L}, \phi, \theta)$
end for
return f_ϕ, s_θ

Algorithm 2 Annealed Langevin dynamics.

Require: Noise levels $\{\sigma_i\}_{i=1}^k$; Step size α; Number of steps T

Initialize x_0
for $i \leftarrow 1$ to k **do**
  for $t \leftarrow 0$ to $T - 1$ **do**
    $\epsilon_t \sim \mathcal{N}(0, I)$
    $x'_{t+1} \leftarrow x_t + \frac{\sqrt{\alpha} \sigma_i \epsilon_t}{\sigma_k}$
    $x_{t+1} \leftarrow x'_{t+1} + \frac{\alpha \sigma_i^2}{2 \sigma_k^2} g_\theta(x'_{t+1}, \sigma_i)$
  end for
  $x_0 \leftarrow x_T$
end for
return x_T

Algorithm 3 Ray Casting for Rendering the Iso-surface.

Require: Neural network s_θ; Minimum noise level σ_k; Initial ray origin o_0; Ray direction u; Maximum ray travel $d_{\max}$; Step rate γ; Number of steps $k_{\max}$; Iso-surface level δ; Background color y_{bg}

$d = 0$
for $k \leftarrow 1$ to $k_{\max}$ **do**
  $x \leftarrow o_0 + du$
  $d \leftarrow d + \gamma(\|s_\theta(x, \sigma_k)\| - \delta)$
end for
if $d < d_{\max}$ **then**
  $n \leftarrow -\frac{s_\theta(x,\sigma_k)}{\|s_\theta(x,\sigma_k)\|}$
  $y \leftarrow \text{Shading}(x, n)$
else
  $y \leftarrow y_{\text{bg}}$
end if
return y

1.3 Objective Function

Here, we provide a proof to show that optimizing $\ell_{\text{direct}}(\sigma, S)$ (Equation 4 in the main paper) is equivalent to optimizing $\ell_{\text{denoising}}(\sigma, S)$ (Equation 5 in the main paper). The proof is largely similar to the one in the Appendix in Vincent 2010 [15]. We will re-visit the prove here using the notation from our paper for convenience of the readers.

Theorem Let $\theta^*_{\text{direct}} = \text{argmin}_\theta \ell_{\text{direct}}(\sigma, S)$, and let $\theta^*_{\text{denoise}} = \text{argmin}_\theta \ell_{\text{denoising}}(\sigma, S)$, then $\theta^*_{\text{direct}} = \theta^*_{\text{denoise}}$. In other words, optimizing $\ell_{\text{denoise}}(\sigma, S)$ leads to the same θ as optimizing $\ell_{\text{denoise}}(\sigma, S)$.

Proof: We want to show that $\ell_{\text{direct}}(\sigma, S) = \ell_{\text{denoise}}(\sigma, S) + C$ for some constant C that doesn't depend on θ. Note that ℓ_{denoise} can be decomposed as follows:

$$\begin{aligned}\ell_{\text{denoise}}(\sigma, S) = \mathbb{E}_{x\sim P_S, \tilde{x}\sim q_\sigma(\tilde{x}|x)} \left[\frac{1}{2}\|g_\theta(\tilde{x}, \sigma)\|^2\right] \\ - \mathbb{E}_{x\sim P_S, \tilde{x}\sim q_\sigma(\tilde{x}|x)} \left[g_\theta(\tilde{x}, \sigma)^T \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x)\right] + C_1, \end{aligned} \tag{1}$$

where $C_1 = \mathbb{E}_{x\sim P_S, \tilde{x}\sim q_\sigma(\tilde{x}|s)} \left[\frac{1}{2}\left\|\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x)\right\|^2\right]$. Similarly, we can decompose ℓ_{direct} as follows:

$$\begin{aligned}\ell_{\text{direct}}(\sigma, S) = \mathbb{E}_{x\sim Q_{\sigma,S}} \left[\frac{1}{2}\|g_\theta(x, \sigma)\|^2\right] \\ - \mathbb{E}_{x\sim Q_{\sigma,S}} \left[g_\theta(x, \sigma)^T \nabla_x \log Q_{\sigma,S}(x)\right] + C_2, \end{aligned} \tag{2}$$

where $C_2 = \mathbb{E}_{x\sim Q_{\sigma,S}} \left[\frac{1}{2}\|\nabla_x \log Q_{\sigma,S}(x)\|^2\right]$. Now we will compare the first two terms

of Equation 1 and Equation 2 to show they are the same. Looking at the first terms:

$$
\begin{aligned}
&\mathbb{E}_{s\sim P_S,\tilde{x}\sim q_\sigma(\tilde{x}|s)}\left[\frac{1}{2}\|g_\theta(\tilde{x},\sigma)\|^2\right] \\
&= \frac{1}{2}\int_x P_S(x)\int_{\tilde{x}} q_\sigma(\tilde{x}|x)\|g_\theta(\tilde{x},\sigma)\|^2 d\tilde{x}dx \\
&= \frac{1}{2}\int_{\tilde{x}}\int_x P_S(x)q_\sigma(\tilde{x}|x)\|g_\theta(\tilde{x},\sigma)\|^2 dxd\tilde{x} \\
&= \frac{1}{2}\int_{\tilde{x}} \|g_\theta(\tilde{x},\sigma)\|^2 \int_x P_S(x)q_\sigma(\tilde{x}|x)dxd\tilde{x} \\
&= \frac{1}{2}\int_{\tilde{x}} \|g_\theta(\tilde{x},\sigma)\|^2 Q_{\sigma,S}(\tilde{x})d\tilde{x} \quad \left(\text{since } Q_{\sigma,S}(x) = \int_y P_S(y)q_\sigma(x|y)dy\right) \\
&= \mathbb{E}_{x\sim Q_{\sigma,S}}\left[\frac{1}{2}\|g_\theta(x,\sigma)\|^2\right].
\end{aligned}
$$

Looking at the second terms:

$$
\begin{aligned}
&\mathbb{E}_{x\sim P_S,\tilde{x}\sim q_\sigma(\tilde{x}|x)}\left[g_\theta(\tilde{x},\sigma)^T\nabla_{\tilde{x}}\log q_\sigma(\tilde{x}|x)\right] \\
&= \int_x P_S(x)\int_{\tilde{x}} q_\sigma(\tilde{x}|x)g_\theta(\tilde{x},\sigma)^T\nabla_{\tilde{x}}\log q_\sigma(\tilde{x}|x)d\tilde{x}dx \\
&= \int_{\tilde{x}}\int_x P_S(x)q_\sigma(\tilde{x}|x)g_\theta(\tilde{x},\sigma)^T\nabla_{\tilde{x}}\log q_\sigma(\tilde{x}|x)dxd\tilde{x} \\
&= \int_{\tilde{x}} g_\theta(\tilde{x},\sigma)^T\int_x P_S(x)q_\sigma(\tilde{x}|x)\nabla_{\tilde{x}}\log q_\sigma(\tilde{x}|x)dxd\tilde{x} \\
&= \int_{\tilde{x}} g_\theta(\tilde{x},\sigma)^T\int_x P_S(x)q_\sigma(\tilde{x}|x)\frac{\nabla_{\tilde{x}}q_\sigma(\tilde{x}|x)}{q_\sigma(\tilde{x}|x)}dxd\tilde{x} \\
&= \int_{\tilde{x}} g_\theta(\tilde{x},\sigma)^T\int_x P_S(x)\nabla_{\tilde{x}}q_\sigma(\tilde{x}|x)dxd\tilde{x} \\
&= \int_{\tilde{x}} g_\theta(\tilde{x},\sigma)^T\int_x P_S(x)\nabla_{\tilde{x}}q_\sigma(\tilde{x}|x)dxd\tilde{x}.
\end{aligned}
$$

Since $q_\sigma(\tilde{x}|x) = \mathcal{N}(\tilde{x}; x, \sigma^2 I)$, it is bounded by $q_\sigma(\tilde{x}|x) \leq \mathcal{N}(x; x, \sigma^2 I)$. As a result,

we can take the derivative outside of the integral $\int_x P_S(x)\nabla_{\tilde{x}} q_\sigma(\tilde{x}|x)dx$:

$$\begin{aligned}
&\mathbb{E}_{x\sim P_S, \tilde{x}\sim q_\sigma(\tilde{x}|x)}\left[g_\theta(\tilde{x},\sigma)^T \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x)\right] \\
&= \int_{\tilde{x}} g_\theta(\tilde{x},\sigma)^T \nabla_{\tilde{x}} \left(\int_x P_S(x) q_\sigma(\tilde{x}|x)dx\right) d\tilde{x} \\
&= \int_{\tilde{x}} g_\theta(\tilde{x},\sigma)^T Q_{\sigma,S}(\tilde{x}) \frac{\nabla_{\tilde{x}} \left(\int_x P_S(x) q_\sigma(\tilde{x}|x)dx\right)}{Q_{\sigma,S}(\tilde{x})} d\tilde{x} \\
&= \int_{\tilde{x}} g_\theta(\tilde{x},\sigma)^T Q_{\sigma,S}(\tilde{x}) \frac{\nabla_{\tilde{x}} Q_{\sigma,S}(\tilde{x})}{Q_{\sigma,S}(\tilde{x})} d\tilde{x} \\
&= \int_{\tilde{x}} g_\theta(\tilde{x},\sigma)^T Q_{\sigma,S}(\tilde{x}) \nabla_{\tilde{x}} \log Q_{\sigma,S}(\tilde{x}) d\tilde{x} \\
&= \mathbb{E}_{\tilde{x}\sim Q_{\sigma,S}}\left[g_\theta(\tilde{x},\sigma)^T \nabla_{\tilde{x}} \log Q_{\sigma,S}(\tilde{x})\right]
\end{aligned}$$

At this point, we can conclude that by setting $C = C_1 - C_2$, we will have $\ell_{\text{direct}}(\sigma, S) = \ell_{\text{denoise}}(\sigma, S) + C$. So optimizing either of ℓ_{direct} or ℓ_{denoise} will give the same optimal θ. ■

2 Implementation Details

2.1 Network architecture

Auto-encoding For auto-encoding, our model takes 2D or 3D shape point cloud X as input to the encoder, which follows the architecture proposed by [11], and outputs the 128-dimensional latent code z for each shape.

For the decoder, we train several noise level σ at the same time. To condition on different noise level, we concatenate the noise level σ at the end of latent code z. The input point cloud X has 800 or 2048 points x in total, and each point is concatenated with latent code z and σ yielding a 131 or 132 dimensional input for the decoder (depending if the point cloud is in 2D or 3D).

Following the architecture used by OccNet [9], first the input is scaled with a fully-connected layer to the hidden dimension 256. Then there are 8 pre-activation ResNet-blocks with 256 dimensions for every hidden layer, and each Res-block consists of two sets of Conditional Batch-Normalization (CBN), a ReLU activation layer and a fully-connected layer. The output of these two sets is added to the input of the Res-block. Then the output of all the Res-blocks is passed through another set of CBN, ReLU and FC layer, and this is the final output of the model – a 3-dimensional vector describing the gradient for the input point.

The CBN layer takes the concatenated input as the latent code $\tilde{z} = [x, z, \sigma]$. The input $\tilde{z}$ is passed through two FC layers to output the 256-dimensional vectors $\beta(\tilde{z})$ and $\gamma(\tilde{z})$. The output of the CBN is computed according to:

$$f_{out} = \gamma(\tilde{z}) \frac{f_{in} - \mu}{\sqrt{\sigma'^2 + \epsilon}} + \beta(\tilde{z}), \tag{3}$$

where μ and σ' are the mean and standard deviation of the batch feature data f_{in}. During training, μ and σ' are the running mean with momentum 0.1, and they are replaced with the corresponding running mean at test time. Figure 1 describes the architecture of our decoder s_θ.

Generation For generation, based on the pretrained auto-encoder model, we use l-GAN to train the latent code generator. Specifically, we train our GAN with WGAN-GP [7] objective. We use Adam optimizer ($\beta_1 = 0.5, \beta_2 = 0.9$) with learning rate 10^{-4} for both the discriminator and the generator. The latent-code dimension is set to be 256. The generator takes a 256-dimensional noise vector sampled from $\mathcal{N}(\mathbf{0}, 0.2^2 I_{256})$, and passes it through an MLP with hidden dimensions of $\{256, 256\}$ before outputting the final 256-dimensional latent code. We apply ReLU activation between layers and there is no batch normalization. The discriminator is a three-layers MLP with hidden dimension $\{512, 512\}$. We use LeakyReLU with slope 0.2 between the layers. We fixed both the pretrained encoder and decoder (i.e. set into evaluation mode). The latent-GAN is trained for 5000 epochs for each of the category.

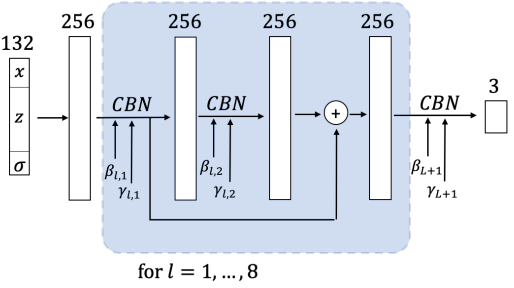

Figure 1: The gradient decoder network s_θ. For each layer, the spatial size is specified on top. The input point x is concatenated to the latent code z and noise level σ. The output of s_θ is the gradient corresponding to the given point.

2.2 Experimental setting

For all the experiments, we use an Adam optimizer. We use ten different σ's ranging from 1 to 0.01. For the ShapeNet dataset, the learning rates are 1×10^{-4} for decoder and 1×10^{-3} for encoder, with linear decay starting at 1000 epoch, reaching 1×10^{-5} and 1×10^{-4} for the encoder and the decoder, respectively. For the MNIST-CP dataset, the learning rate starts with 1×10^{-3} for both the decoder and encoder, with linear decay starting at 1000 epoch, ending at 1×10^{-4}. Each batch consists of 64 shapes (or 200 shapes for the MNIST-CP dataset). We train the model for 2000 epochs. For inference, we set $T = 10$ and $\alpha = 2 \times 10^{-4}$.

2.3 Baselines

In this section, we provide more details on how we obtain the reported scores for alternative methods.

PointFlow [16] To get the results for ShapeNet, we run the pre-trained checkpoint release in the official code repository[1]. The auto-encoding results for MNIST is obtained by running the released code on the pre-processed MNIST-CP dataset.

AtlasNet [6] The code we used for the AtlasNet decoder comes from the official code repository[2]. To enable a fair comparison, we use the same encoder as used in the PointFlow repository, and set the latent code dimension to be the same as our own model. We use the suggested learning rate and optimizer setting from the AtlasNet code-base and paper during training. We train AtlasNet for the same amount of iterations as our method to obtain the reported performance in Table 1 in the main paper.

l-GAN and r-GAN [2] We modify the official released code repository[3] to take our pre-processed point cloud from ShapeNet version 2 [3, 4]. The auto-encoding results (i.e. Table 1 in main paper) for l-GAN and r-GAN are obtained by running the official code for the same number of iterations as our model. The generation results for r-GAN in Table 3 in the main paper is obtained by running the latent-GAN in the official code for the default amount of iterations in the configuration.

GraphCNN-GAN [14] We use the official code released in this repository to obtain the results: `https://github.com/diegovalsesia/GraphCNN-GAN`.

TreeGAN [13] We use the official code released in this repository to obtain the results: `https://github.com/seowok/TreeGAN.git`.

3 Additional Quantitative Results

3.1 Shape generation

We compare our method's performance on shape generation with GraphCNN-GAN [14] and TreeGAN [13] in Table 1 (in addition to the comparisons to r-GAN [1] and PointFlow [16] which we report in the main paper). As discussed in the Related Work section (Section 2 in the main paper), both of these two baselines treat generating point clouds with N points as predicting a fixed dimensional vector (in practice predicting a $N \times 3$ vector but they could potentially use more upsampling layers to predict more points), using the same discriminator as r-GAN [2]. These works report performance for models

[1] `https://github.com/stevenygd/PointFlow`
[2] `https://github.com/ThibaultGROUEIX/AtlasNet`
[3] `https://github.com/optas/latent_3d_points`

trained on smaller collections (i.e. the ShapeNet Benchmark dataset[4]) using different splits and normalization. Therefore, in addition to comparing their publicly available models (in the first rows), we retrain their models on the full ShapeNet collections using the same splits and preprocessing performed on our trained models (in the rows marked with an asterisk ($*$)). For each model, the training is performed over two days with a GeForce GTX TITAN X GPU.

Table 1: Additional shape generation results. Rows marked with an asterisk ($*$) denote retrained models. $\uparrow$ means the higher the better, $\downarrow$ means the lower the better. MMD-CD is multiplied by 10^3 and MMD-EMD is multiplied by 10^2.

		MMD ($\downarrow$)		COV (%, $\uparrow$)		1-NNA (%, $\downarrow$)	
Category	Model	CD	EMD	CD	EMD	CD	EMD
Airplane	GCN [14]	2.623	15.535	9.38	5.93	95.16	99.12
	GCN [14]*	44.93	35.52	1.98	1.23	99.99	99.99
	Tree [13]	1.466	16.662	44.69	6.91	95.06	100.00
	Tree [13]*	1.798	24.723	31.60	5.43	95.43	99.88
	Ours	**1.285**	**7.364**	**47.65**	**41.98**	**85.06**	**83.46**
	Train	1.288	7.036	45.43	45.43	72.10	69.38
Chair	GCN [14]	23.098	25.781	6.95	6.34	86.52	96.48
	GCN [14]*	140.84	0.5163	1.67	1.06	100	100
	Tree [13]	16.147	36.545	40.33	8.76	74.55	99.92
	Tree [13]*	17.124	26.405	42.90	20.09	77.49	98.11
	Ours	**14.818**	**18.791**	**46.37**	**46.22**	**66.16**	**59.82**
	Train	15.893	18.472	50.45	52.11	53.93	54.15

We also present generation results for the car category in Table 2. Our model achieves performance that's un-par with the state-of-the-arts.

Table 2: Shape generation results. $\uparrow$ means the higher the better, $\downarrow$ means the lower the better. MMD-CD is multiplied by 10^3 and MMD-EMD is multiplied by 10^2.

		MMD ($\downarrow$)		COV (%, $\uparrow$)		1-NNA (%, $\downarrow$)	
Category	Model	CD	EMD	CD	EMD	CD	EMD
Car	rGAN [2]	6.233	18.561	8.24	5.11	99.29	99.86
	PF [16]	4.207	10.631	39.20	44.89	68.75	**62.64**
	Ours	**4.085**	**10.610**	**44.60**	**46.88**	**65.48**	62.93
	Train	4.207	10.631	48.30	54.26	52.98	49.57

In Figure 2, we show convergence curves for our method and for PointFlow [16]

[4] https://shapenet.cs.stanford.edu/ericyi/shapenetcore_partanno_segmentation_benchmark_v0.zip

on the auto-encoding task (over the Airplane category). As the figure illustrates, our method converges much faster and to a better result.

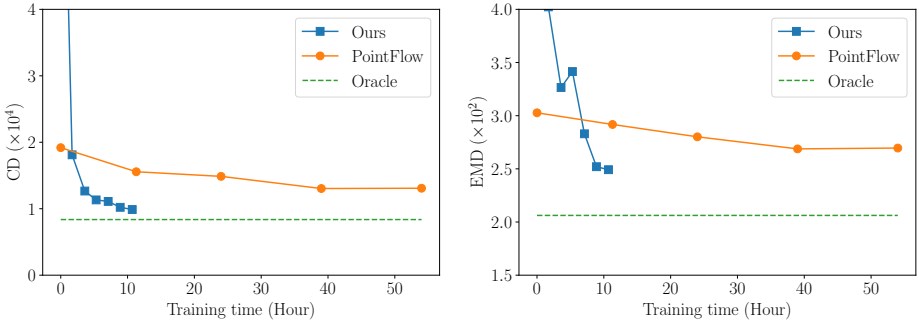

Figure 2: Convergence curves for PointFlow [16] and our method on the auto-encoding task. As illustrated above, our method converges much faster to a better result.

3.2 Implicit surface

Table 3: Implicit surface results on Airplane category. CD is multiplied by 10^4 and EMD is multiplied by 10^2.

Metrics	AtlasNet-Sph.	AtlasNet-25	DeepSDF	Ours
CD	1.88	2.16	1.43	**1.022**
CD (median)	0.79	0.65	**0.36**	0.442
EMD	3.8	4.1	**3.3**	5.545
Mesh acc	0.013	0.013	**0.004**	0.008

In this section, we demonstrate that one can use MISE [9], an octree-based marching cue algorithm, to extract a ground truth mesh that from the learned gradient field. We compute ground truth meshes for the test set of the airplane category following DeepSDF's set-up. As mentioned in Section 1 of main paper, prior implicit representations [5][9][10] require knowing the ground truth meshes in order to provide a supervision signal during training, while our model can be trained solely from sparse point clouds. We conducted a preliminary quantitative comparison between our implicit surface and that of DeepSDF[10]. We follow DeepSDF's experiment set-up to report results on the airplane category in Table3. Our implicif surfaces outperforms AtlasNet in both CD and Mesh accuracy metrics and are competitive with DeepSDF (which uses more supervision) in CD. Failure cases for our extracted meshes usually comes from the bifurcation area (i.e. the local minimums and saddle points) where gradients are close to zero. Another problem with our extracted surface is that marching cue tend to create a double surface around the shape, As our focus is generating point clouds, we will leave the improvement of surface extraction to future work.

Table 4: Architecture ablation study, comparing auto-encoding performance on the Airplane category. CD is multiplied by 10^4 and EMD is multiplied by 10^2. The ablated models are detailed in the text.

Metrics	(a)	(ab)	(abc)	(bd)	(abcd)
CD	1.234±0.007	0.992±0.002	0.998±0.003	1.011±0.008	0.987±0.001
EMD	2.718±0.039	2.513±0.019	2.493±0.010	2.462±0.042	2.524±0.001

4 Additional Ablation Studies

Next we report results of additional ablation studies to evaluating several design considerations and our choice in modeling the distribution of shapes.

Network architecture We evaluate different architectures considering the following:

(a) Replacing BN with CBN.

(b) Adding shortcuts.

(c) Replacing the latent code z with $\tilde{z} = [x, z, \sigma]$ for the CBN layer.

(d) Concatenating the latent code z and σ to x as input for the decoder.

In Table 4 we report multiple configurations, with the rightmost one (abcd) corresponding to our full model. For each model, we perform 3 inference runs, and report the average and the standard deviation over these runs. As the table illustrates, our full model yields better performance as well as significantly smaller variance across different runs.

Modeling the distribution of shapes In our work, we propose a new approach of modeling the distribution of points using the gradient of the log density field. To model the distribution of shapes, we use a latent GAN [1]. Next, we explore a different method to model the distribution of shapes. Specifically, we train a VAE using the same encoder and decoder setting as the one used in Section 4.2 in the main paper. We double the output dimension for the encoder so that it can output both μ and σ for the re-parameterization. We add the KL-divergence loss with weight 10^{-3} and train for the same amount of time (in terms of epochs and iterations) as the model reported in the main paper. The results for the Chair and Airplane categories are reported in Table 5. We can see that our reported model, which uses a two-stage training with a latent-GAN, outperforms the model trained with VAE for both the Chair and the Airplane category on all metrics, except for the 1-NNA-EMD metrics on Airplane.

5 Additional Qualitative Results

Scanned data To demonstrate that our technique can also model partial and incomplete shapes, we use scanned point clouds captured using a hand-held 3D scanner, as detailed

Table 5: Modeling the distribution of shapes using different techniques. We compare our model's performance on the generation task against a VAE model. ↑ means the higher the better, ↓ means the lower the better. MMD-CD is multiplied by 10^3 and MMD-EMD is multiplied by 10^2.

		MMD (↓)		COV (%, ↑)		1-NNA (%, ↓)	
Category	Model	CD	EMD	CD	EMD	CD	EMD
Airplane	VAE	1.909	9.004	37.78	38.27	89.14	**86.05**
	Ours	**1.332**	**7.768**	**39.01**	**43.46**	**88.52**	86.91
	Train	1.288	7.036	45.43	45.43	72.10	69.38
Chair	VAE	18.032	20.903	41.99	43.81	74.17	74.92
	Ours	**14.818**	**18.791**	**46.37**	**46.22**	**66.16**	**59.82**
	Train	15.893	18.472	50.45	52.11	53.93	54.15

in Yifan et al.[17]. For each scanned object, we train a separate model, evaluating to what extent we can obtain a high-fidelity point cloud reconstruction for these scanned objects. See Figure 3 for a qualitative comparison of the input scanned objects which contain roughly 600 points (left columns), reconstructed point clouds (middle columns) and extracted implicit surfaces (right columns). While we cannot model the full shape in this case (as we are only provided with a partial, single view), our technique enables reconstructing a denser representation even in this sparse real setting.

Visualization of latent space We visualize the latent code space in Figure 4. We first obtain all latent code $z \in \mathbb{R}^{128}$ by running the encoder on the validation set of Airplane, Car, and Chair. We run T-SNE [8] on the latent code for the same category to reduce the latent-codes' dimensionality down to 2. These 2-dimensional latent codes are then used to place rendered reconstructed point clouds on the figure. The figure shows that the latent code places similar shapes nearby in the latent space, which suggests that we learn a meaningful latent space.

Extended visualizations for 2D and 3D point clouds In Figure 5, we demonstrate our annealed Langevin dynamic procedure for 2D point clouds from MNIST-CP. We also show that our model is insensitive to the choice of the prior distribution in the 2D case in Figure 6. We show more results on the ShapeNet dataset in Figure 7 (auto-encoding shapes), Figure 8(auto-encoding point clouds), Figure 9 (shape generation), and Firgure 10 (shape interpolation).

Figure 3: Autoencoding scanned shapes. Above we demonstrate our technique on sparse point clouds captured with a 3D scanner (left). We sample 10K points to obtain the point clouds illustrated in the middle and also extract the implicit surfaces (right).

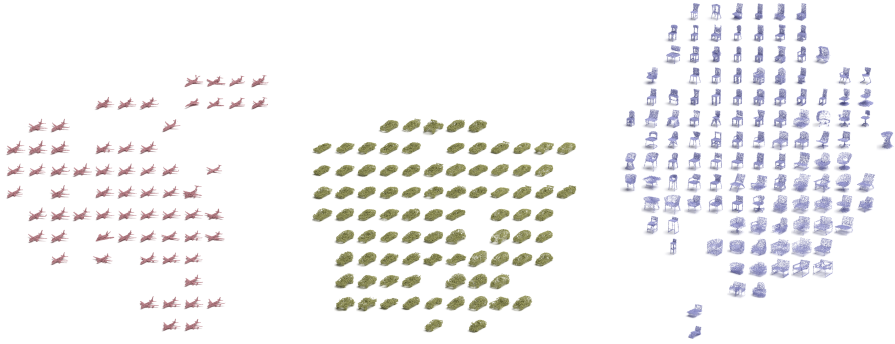

Figure 4: Visualization of the latent space.

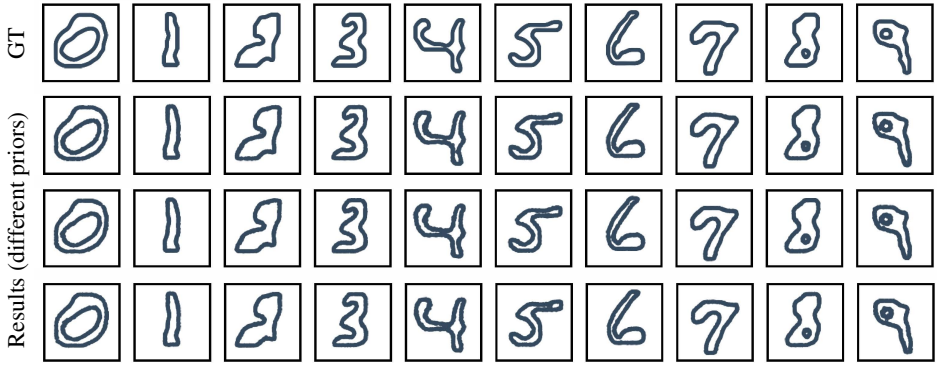

Figure 6: Reconstructing MNIST-CP shapes (illustrated on top), starting from different prior distributions. Above we demonstrate reconstruction results obtained starting from a uniform (second row), Gaussian (third row) or a single point (fourth row) distribution. As the figure illustrates, our method is insensitive to the prior distribution.

Figure 5: Point cloud sampling on the MNIST-CP dataset. Above we illustrate our annealed Langevin dynamics procedure for 2D shapes. Starting with points sampled from a uniform distribution, the points gradually move along the logarithmic density field. As illustrated on the right side, eventually these points are mostly indistinguishable from the ground truth point clouds.

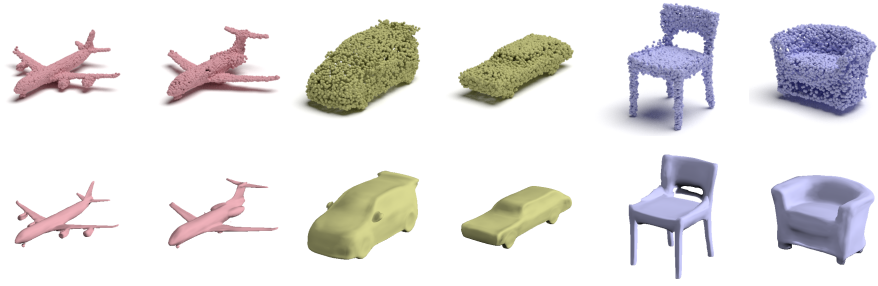

Figure 7: Examples of reconstruction results. The first row depicts reconstructed shapes from the Airplane, Car and Chair categories. The second row is the corresponding implicit surfaces.

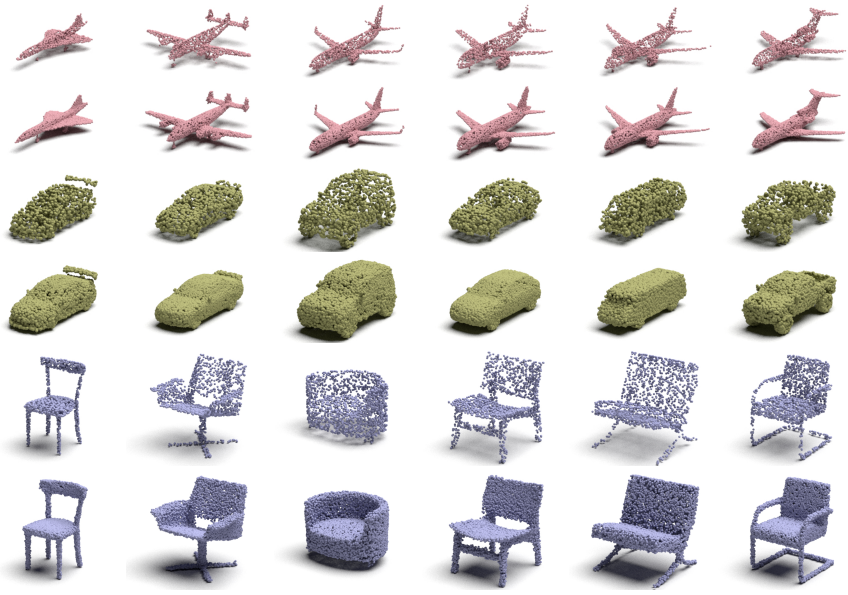

Figure 8: Examples of reconstruction results on the Airplane, Car and Chair categories. For each category, the first row is the input point cloud, and the second row is the up-sampling output point cloud.

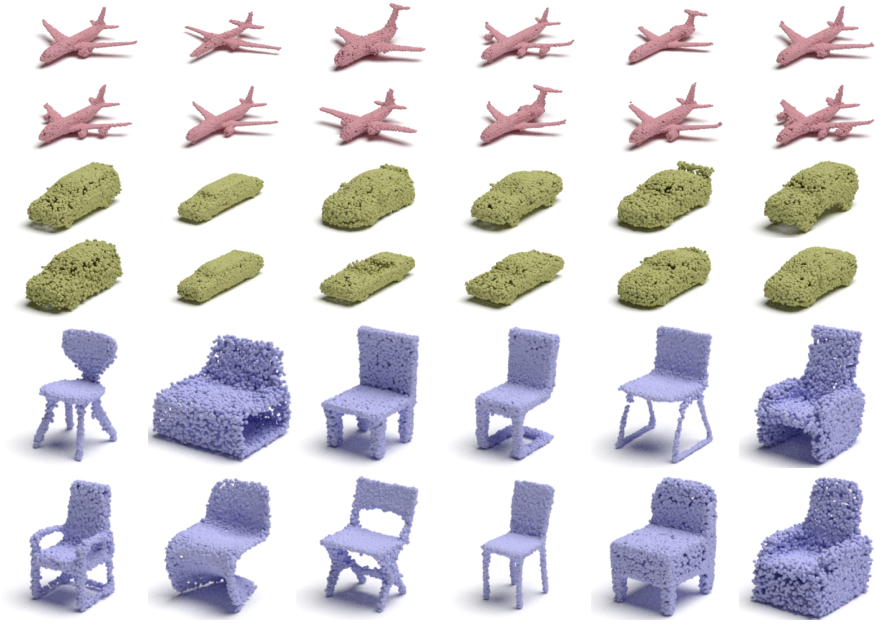

Figure 9: Examples of generation results on the Airplane, Car and Chair categories.

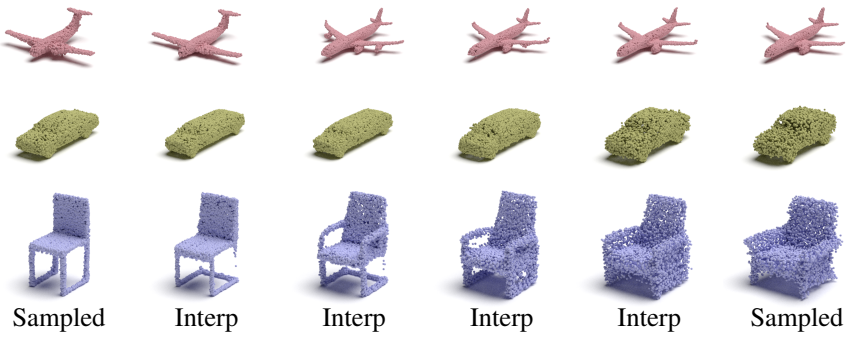

Figure 10: Generation and interpolation results. Generated point clouds (Sampled) and the interpolated results between two generated shapes (Interp) are illustrated.

References

[1] Achlioptas, P., Diamanti, O., Mitliagkas, I., Guibas, L.: Learning representations and generative models for 3d point clouds. arXiv preprint arXiv:1707.02392 (2017) 7, 10

[2] Achlioptas, P., Diamanti, O., Mitliagkas, I., Guibas, L.: Learning representations and generative models for 3d point clouds. In: ICML (2018) 7, 8

[3] Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: ShapeNet: An Information-Rich 3D Model Repository. Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago (2015) 7

[4] Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al.: Shapenet: An information-rich 3d model repository. arXiv preprint arXiv:1512.03012 (2015) 7

[5] Chen, Z., Zhang, H.: Learning implicit fields for generative shape modeling. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5939–5948 (2019) 9

[6] Groueix, T., Fisher, M., Kim, V.G., Russell, B., Aubry, M.: AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In: CVPR (2018) 7

[7] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein gans. In: NeurIPS (2017) 6

[8] Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(Nov), 2579–2605 (2008) 11

[9] Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., Geiger, A.: Occupancy networks: Learning 3d reconstruction in function space. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4460–4470 (2019) 5, 9

[10] Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: Deepsdf: Learning continuous signed distance functions for shape representation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 165–174 (2019) 9

[11] Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: CVPR (2017) 5

[12] Roth, S.D.: Ray casting for modeling solids. Computer graphics and image processing **18**(2), 109–144 (1982) 1

[13] Shu, D.W., Park, S.W., Kwon, J.: 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) 7, 8

[14] Valsesia, D., Fracastoro, G., Magli, E.: Learning localized generative models for 3d point clouds via graph convolution (2018) 7, 8

[15] Vincent, P.: A connection between score matching and denoising autoencoders. Neural computation **23**(7), 1661–1674 (2011) 3

[16] Yang, G., Huang, X., Hao, Z., Liu, M.Y., Belongie, S., Hariharan, B.: Pointflow: 3d point cloud generation with continuous normalizing flows. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 4541–4550 (2019) 7, 8, 9

[17] Yifan, W., Wu, S., Huang, H., Cohen-Or, D., Sorkine-Hornung, O.: Patch-based progressive 3d point set upsampling. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5958–5967 (2019) 11