Neural Point Cloud Diffusion for Disentangled 3D Shape and Appearance Generation – Appendix –

A1. Visualization of the neural point cloud diffusion process

Figure A1 shows a visualization of the neural point cloud diffusion process for unconditional generation on ShapeNet Cars, ShapeNet Chairs, and PhotoShape Chairs.

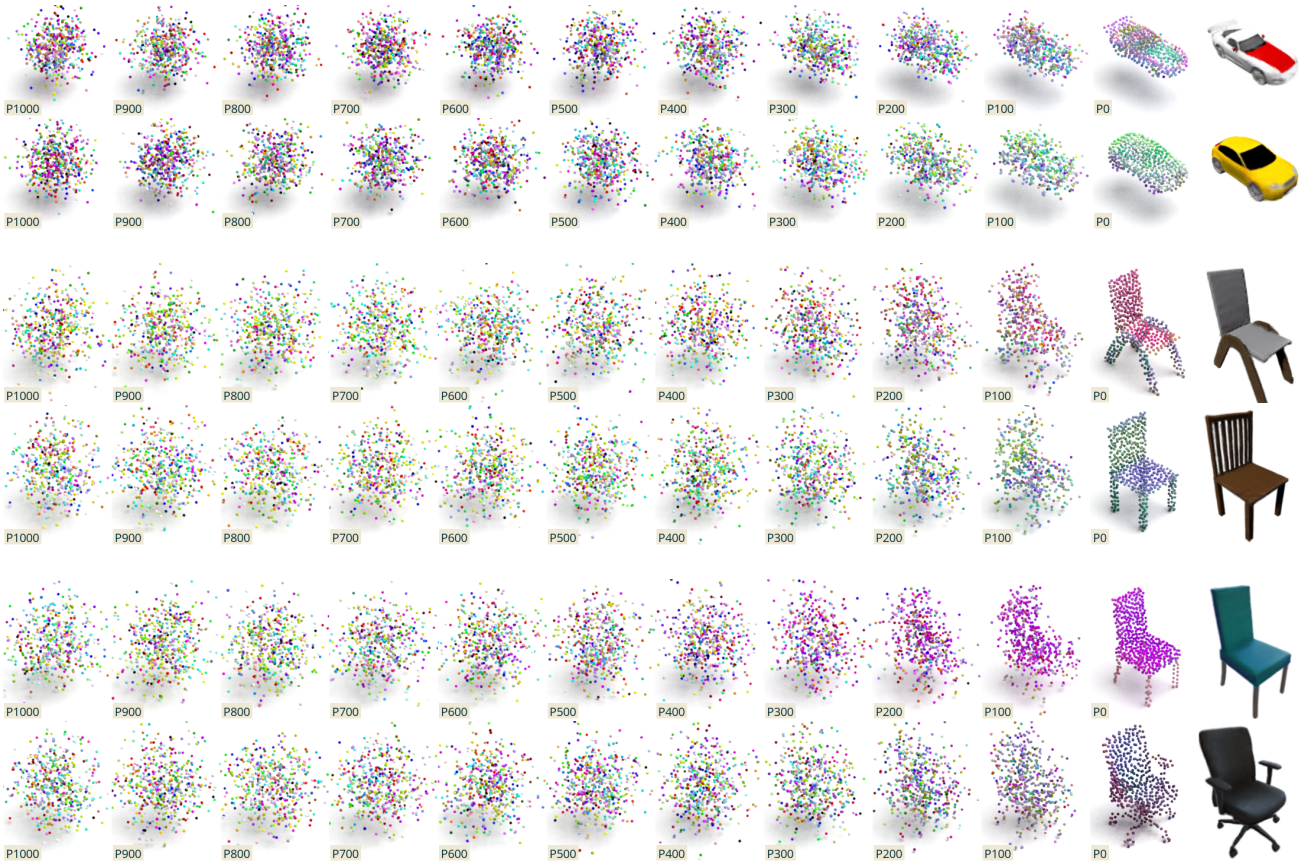

Figure A1. **Visualization of the neural point cloud diffusion process**. We generate the shape and appearance of 3D objects on ShapeNet Cars, ShapeNet Chairs, and PhotoShape Chairs with the proposed Neural Point Cloud Diffusion (NPCD) model. We visualize the neural point clouds $\mathcal{P}_t = (\mathbf{P}_t, \mathbf{F}_t)$ from intermediate timesteps t of the diffusion process. In total, the diffusion process of NPCD has 1000 timesteps and we visualize every 100th timestep. The features of the neural point clouds are visualized by taking the first three PCA components as RGB color. The last visualized neural point cloud $\mathcal{P}_0$ represents the final generated 3D object. Additionally, we visualize a Point-NeRF rendering of the final neural point cloud.

A2. Implementation details

A2.1. Category-Level Point-NeRF Autodecoder

Architecture. For the aggregation MLP F_ϕ, we use 4 linear layers with a hidden dimension of 256, each followed by a LeakyReLU, and an output projection linear layer that maps to 256d. For the color MLP G_ψ, we use 4 linear layers with a hidden dimension of 256, each followed by a LeakyReLU, and an output projection linear layer that maps to 3d. For the density MLP H_γ, we use 1 linear layer with a hidden dimension of 256, followed by a LeakyReLU, and an output projection linear layer that maps to 1d.

Training parameters. We construct training samples by splitting the available views per object into groups of 50 views per sample. In each iteration, we use 8 samples and 112 pixels of each view in each sample. The image reconstruction loss is hence optimized for an effective batch size of $8 \cdot 50 \cdot 112 = 44800$ pixels. For the volumetric rendering of each ray, we sample 128 shading points. We use the Adam optimizer with a constant learning rate of $1e$–3. On SRN cars and chairs, we train Point-NeRF for ca. 7500 epochs. On PhotoShape Chairs, we train Point-NeRF for ca. 1875 epochs (due to the 4 times higher number of training views per object, each epoch contains 4 times as many samples per object as in SRN Cars/Chairs). The training time on a single RTX 4090 GPU is between 3 and 5 days, depending on the dataset.

Rendering runtime. For the Point-NeRF rendering at a resolution of 128x128px, we measure a mean runtime of 35 msec on a RTX 4090 GPU.

A2.2. Diffusion model

Architecture. For the denoiser network of the diffusion model, we use a standard transformer architecture with 24 layers, a feature dimension of 1024d and 16 heads [8, 9]. This architecture has ca. 300M parameters.

Diffusion model parameters. For the diffusion model, we use the linear noise schedule from DDPM [5] with 1,000 steps and β ranging from 0.0001 to 0.02. We normalize the neural point clouds such that the positions are unit Gaussian distributed and the features are in the range $[-1, 1]$ (and apply the inverse transform later before rendering the representation). During sampling, we clip the coordinates and features to the respective minimum and maximum values of the training dataset.

Training parameters. We train the diffusion model for ca 1.8M iterations with a batch size of 32. We train on a single RTX 4090 GPU with 16 bit and employ flash attention [4]. Training takes ca. 8 days. We use an exponential moving average over the model parameters with a decay of of 0.9999. On ShapeNet Cars, we use a constant learning rate of $7e$–5. On ShapeNet Chairs and PhotoShape Chairs, we use a lower constant learning rate of $4e$–5, as we observed instablities during training. On all datasets, we use a weight decay of 0.01.

Runtime for unconditional generation. For unconditional generation with 1000 diffusion steps, we measure a mean runtime of 8.56 sec for a single generation (batch size 1) on a RTX 4090 GPU.

A2.3. Disentangled generation

As described in the main paper, our disentangled generation is comparable to masked image inpainting, using an approach similar to RePaint [7]. Instead of masking image parts, we mask one modality of our representation (point positions or features). In Algorithm 1, we provide the algorithm that we use for disentangled generation in full detail. The algorithm is for appearance-only generation, *i.e.* where point positions $\mathbf{P}_0$ are given and the goal is to generate point features $\mathbf{F}_0$. For shape-only generation, it works vice-versa.

Initialization. As described in the main paper, we obtain the initial noisy neural point cloud $(\mathbf{P}_T, \mathbf{F}_T)$ by sampling $\mathbf{F}_T$ from a unit Gaussian distribution and computing $\mathbf{P}_T$ from $\mathbf{P}_0$ via the forward diffusion process. In Algorithm 1, this is described by line 2. Importantly, for computing point positions via the forward diffusion process, we sample noise $\epsilon^{\mathbf{P}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ once in the beginning and re-use it in subsequent diffusion steps.

Diffusion process. Throughout the diffusion process, we update the point features $\mathbf{F}_{t-1}$ from the denoiser outputs according to the reverse diffusion process (lines 4 and 5).

We update the point positions $\mathbf{P}_{t-1}$ from the given $\mathbf{P}_0$ via the forward diffusion process in instead of the denoiser outputs (line 10). For this, we re-use the noise $\epsilon^{\mathbf{P}}$ that we sampled during the initialization.

Reverse process also for the point positions. In the last N_{rev} steps of the diffusion process, we also use the outputs of the denoiser network to update the point positions via the reverse process (line 13). Depending on the chosen N_{rev} this allows for a trade-off between better coherence in the generations at the cost of deviations from the given input point positions.

Resampling. Further, as in RePaint, we apply N_{resample} resampling steps to the denoised point features in the last N_{repaint} steps of the diffusion process (lines 15 to 21). We observe that this is helpful for the coherence of the generations (in accordance with Fig. 3 of the RePaint paper). As we use the reverse process for the point positions in the last N_{rev} steps of the diffusion process, we do not use resampling in these iterations (second condition in line 15).

Algorithm 1 Appearance-only generation

1: **Input:** Point Positions $\mathbf{P}_0$, N_{rev}, N_{repaint}, N_{resample}

2: **Initialization:** $\epsilon^{\mathbf{P}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ $\quad \mathbf{P}_T = \sqrt{\bar{\alpha}_T}\mathbf{P}_0 + \sqrt{1-\bar{\alpha}_T}\epsilon^{\mathbf{P}}$ $\quad \mathbf{F}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

3: **for** $t = T, \dots, 1$ **do** ▷ we use $T = 1000$

4: $\quad (\epsilon_\theta^{\mathbf{P}}, \epsilon_\theta^{\mathbf{F}}) = T_\theta((\mathbf{P}_t, \mathbf{F}_t), t)$ ▷ estimate noise with denoiser T_θ

5: $\quad \mathbf{F}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{F}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta^{\mathbf{F}}\right) + \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t\epsilon \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

6: $\quad$ **if** $t > N_{\text{rev}}$ **then** ▷ compute $\mathbf{P}_{t-1}$ via forward process

7: $\quad\quad$ **if** $t == 1$ **then**

8: $\quad\quad\quad \mathbf{P}_{t-1} = \mathbf{P}_0$

9: $\quad\quad$ **else**

10: $\quad\quad\quad \mathbf{P}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\mathbf{P}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\epsilon^{\mathbf{P}}$

11: $\quad\quad$ **end if**

12: $\quad$ **else** ▷ compute $\mathbf{P}_{t-1}$ via reverse process for last N_{rev} steps

13: $\quad\quad \mathbf{P}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{P}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta^{\mathbf{P}}\right) + \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t\epsilon \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

14: $\quad$ **end if**

15: $\quad$ **if** $t \leq N_{\text{repaint}}$ and $t \geq N_{\text{rev}}$ **then** ▷ use RePaint-resampling in last N_{repaint} steps in case $\mathbf{P}_t$ comes from forward process

16: $\quad\quad$ **for** $1, \dots, N_{\text{resample}}$ **do** ▷ resample N_{resample} times

17: $\quad\quad\quad \mathbf{F}_t \sim \mathcal{N}(\mathbf{F}_t; \sqrt{1-\beta_t}\mathbf{F}_{t-1}; \beta_t\mathbf{I})$ ▷ forward process

18: $\quad\quad\quad (\epsilon_\theta^{\mathbf{P}}, \epsilon_\theta^{\mathbf{F}}) = T_\theta((\mathbf{P}_t, \mathbf{F}_t), t)$ ▷ denoise again

19: $\quad\quad\quad \mathbf{F}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{F}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta^{\mathbf{F}}\right) + \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t\epsilon \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

20: $\quad\quad$ **end for**

21: $\quad$ **end if**

22: **end for**

For all disentanglement figures in the main paper, we use the following parameters:

- Appearance-only SRN chairs / PhotoShape Chairs: $N_{\text{rev}} = 15$, $N_{\text{repaint}} = 50$, $N_{\text{resample}} = 10$; Mean Runtime for a single generation (batch size 1) on a RTX 4090 GPU: 11.61 sec
- Appearance-only SRN cars: $N_{\text{rev}} = 15$, $N_{\text{repaint}} = 80$, $N_{\text{resample}} = 40$; Mean Runtime for a single generation (batch size 1) on a RTX 4090 GPU: 34.75 sec
- Shape-only SRN chairs / PhotoShape Chairs: $N_{\text{rev}} = 50$, $N_{\text{repaint}} = 100$, $N_{\text{resample}} = 2$; Mean Runtime for a single generation (batch size 1) on a RTX 4090 GPU: 9.15 sec
- Shape-only SRN cars: $N_{\text{rev}} = 50$, $N_{\text{repaint}} = 0$, $N_{\text{resample}} = 0$; Mean Runtime for a single generation (batch size 1) on a RTX 4090 GPU: 8.85 sec

A3. Analysis

As described in the main paper, we conduct ablations studies regarding the effects of different initialization strategies, feature dimensionality and regularization methods in the category-level Point-NeRF autodecoder and diffusion model. These parameters affect both, the quality of the reconstructions and the structure of the neural point cloud feature space. Both properties in turn affect the diffusion model that is trained on the resulting neural point clouds. As the reconstructed objects serve as training data for the diffusion model, the reconstruction quality likely is the upper bound of the generation quality. On the other hand, the structure of the feature space affects how well the data distribution can be learned by the diffusion model.

A3.1. Setup

We conduct the analyses with similar settings as described in the main paper. The main difference is that, for computational reasons, we conduct the analyses with a smaller transformer denoiser network with 40M parameters. Thus, the numbers of this configuration might differ from the best configuration in the main paper. At the end of the analysis, we compare this 40M parameter model to the 300M parameter model from the main paper for the best initialization strategy, dimensionality, and regularization parameters. We conduct the analyses on ShapeNet Cars and Chairs. In case the results on Cars are very clear, we omit the corresponding experiments on Chairs.

A3.2. Neural point cloud initialization

Regarding the initialization of the neural point cloud features, we analyze initialization with features sampled from a Gaussian distribution against a zero initialization. Interestingly, we find that these different initializations strongly affect the feature space of the trained models. To illustrate this, we visualize neural point features of reconstructed objects in Fig. A2.

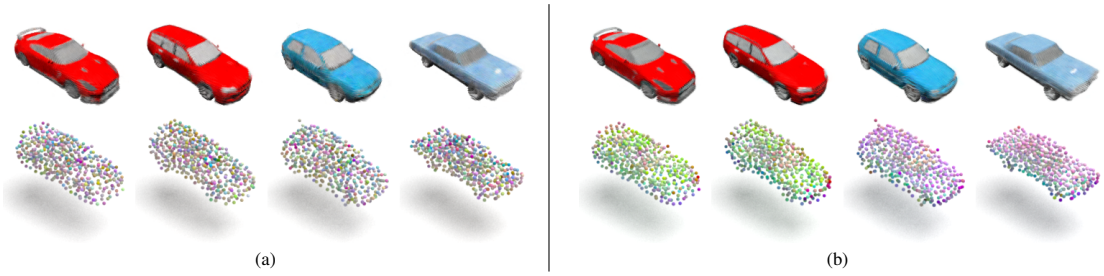

Figure A2. The first row shows the Point-NeRF autodecoder renderings of four *reconstructed* training objects. The second row shows a visualization of the features from the point clouds by taking the first three PCA components as RGB colors. (a) **Random initialization**: The features learned starting from a random initialization are distributed randomly across an object and differ across objects with the same appearance. (b) **Zero initialization**: Features learned starting with a zero initialization are coherent within an object and across objects with the same appearance.

This effect is measured quantitatively via the cosine similarities in Tab. 4 of the main paper. As shown in Fig. A3 and Tab. A1a, the more coherent features from the zero initialization, are vital to enable the successful training of a diffusion model.

Figure A3. Generated samples from diffusion models trained on neural point clouds from category-level Point-NeRF autodecoders that were optimized with different initialization strategies: (a) **Random initialization** leads to many artifacts in the appearance of generated samples. (b) **Zero initialization** leads to more diversity and fewer artifacts.

Setting	Dim.	Init.	Reg.	λ	ShapeNet SRN Cars					ShapeNet SRN Chairs				
					PSNR↑	FID$_{rec}$↓	KID$_{rec}$↓	FID↓	KID↓	PSNR↑	FID$_{rec}$↓	KID$_{rec}$↓	FID↓	KID↓
a) Initialization														
Random initialization	32	Rand.	✗	-	29.24	37.24	25.37	125.51	97.82	-	-	-	-	-
Zero initialization	32	Zero	✗	-	31.32	18.96	11.17	53.55	35.37	34.91	10.37	4.85	39.19	23.64
b) Dimensionality														
16D features	16	Zero	✗	-	30.60	22.56	13.86	56.02	39.30	-	-	-	-	-
32D features	32	Zero	✗	-	31.32	18.96	11.17	53.55	35.37	34.91	10.37	4.85	39.19	23.64
128D features	128	Zero	✗	-	32.65	19.33	11.60	73.93	52.09	-	-	-	-	-
c) Regularization														
No regularization	32	Zero	✗	-	31.32	18.96	11.17	53.55	35.37	34.91	10.37	4.85	39.19	23.64
TV regularization	32	Zero	TV	3.5e-6	29.72	22.42	13.71	45.90	28.70	32.38	14.10	6.70	32.87	17.49
KL regularization	32	Zero	KL	1e-6	30.02	24.93	15.60	55.01	35.86	34.20	8.37	3.17	18.13	8.17
TV+KL regularization	32	Zero	TV,KL	3e-7,1e-7	29.70	26.12	16.44	43.92	26.53	33.62	8.58	3.34	17.17	7.44
d) Model size														
40M parameters	32	Zero	TV,KL	3e-7,1e-7	29.70	26.12	16.44	43.92	26.53	33.62	8.58	3.34	17.17	7.44
300M parameters	32	Zero	TV,KL	3e-7,1e-7	29.70	26.12	16.44	28.38	17.62	33.62	8.58	3.34	9.87	3.62

Table A1. **Analysis of the Point-NeRF autodecoder reconstructions and the diffusion model generations regarding**: **a)** feature initialization, **b)** feature dimensionality, **c)** feature regularization (λ is the weight of the regularization loss) and **d)** model size. The PSNR, FID$_{rec}$ and KID$_{rec}$ metrics in the gray columns measure the quality of the reconstructions in the Point-NeRF autodecoder optimization stage. The FID and KID metrics measure the quality of the diffusion model generations. The reported KID is multiplied with 10^3. Zero initialization clearly outperforms random initialization. 32D features outperform 16D and 128D features. Combined TV+KL regularization outperforms having no regularization or TV or KL regularization alone.

A3.3. Neural point cloud feature dimension

We analyze 16D, 32D and 128D features for the neural point clouds in Tab. A1b and Fig. A4.

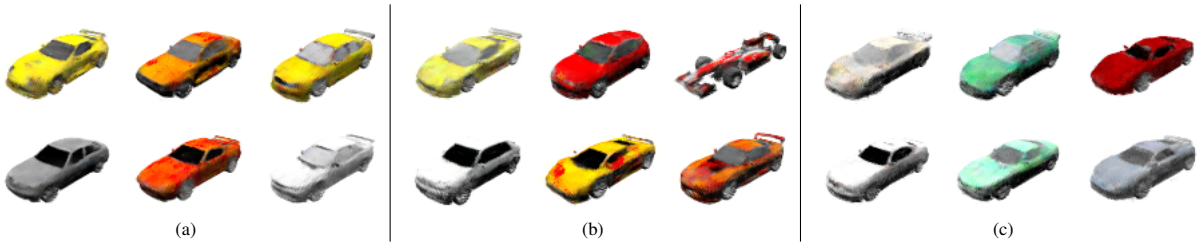

Figure A4. Generated samples from diffusion models trained on neural point clouds with different numbers of feature dimensions: (a) 16D features, (b) 32D features, (c) 128D features. We observe a clear difference between 16 and 32D features, while the difference to 128D is small.

As indicated by the PSNR metrics, higher feature dimensions allow better training data reconstructions in the category-level Point-NeRF autodecoder training. However, the FID and KID metrics for the generation performance of the diffusion model decrease for 128D. As the visual quality of 32D and 128D in Fig. A4 is very similar and the FID and KID metrics for 32D are better, we choose to continue with the 32D features.

A3.4. Neural point cloud regularization

As stated in the main paper, we analyze the effects of applying KL and TV regularizations to the neural point features during the category-level Point-NeRF autodecoder training. The cosine similarities in Tab. 4 of the main paper shows that both regularizations further decrease the ambiguity of the latent space over the zero initialization. However, we observe different behaviors w.r.t. quantitative and qualitative results. On the one hand, among TV and KL regularization, TV leads to the better FID and KID scores in Tab. A1c. The qualitative comparison in Fig. A5 on the other hand shows that KL regularization results

in cleaner samples with less artifacts compared to TV regularization. As a consequence, we try a combination of TV and KL regularization, which leads to an equally high visual quality and the best FID and KID scores.

Figure A5. Generated samples from diffusion models trained on neural point clouds from Point-NeRFs that were optimized with different regularization strategies: (a) No regularization (FID 53.55), (b) TV regularization (FID 45.90), (c) KL regularization (FID 55.01), (d) TV+KL regularization (FID 43.92). TV regularization increases performance regarding the FID and KID metrics. KL regularization leads to cleaner qualitative results. The proposed method NPCD uses a combination of both regularizations, which improves quantitative and qualitative results.

A3.5. Model size

Lastly, we compare the performance of the transformer model with 40M parameters that was used in the analyses, with the performance of the transformer model with 300M parameters, that was used in the main paper. These models differ as follows:

- 40M parameter model: 12 layers, hidden dimension 512, 8 heads; trained with batch size 64, learning rate $1e$–4, not using EMA on the model weights.
- 300M parameter model: 24 layers, hidden dimension 1024, 16 heads; trained with batch size 32, a learning rate $7e$–5 on Cars and $4e$–5 on Chairs, using EMA on the model weights.

Quantitative Results. The quantitative comparison in Tab. A1d shows that the quality of the generated samples scales with the model size.

Overfitting. We observed that 3D diffusion models on ShapeNet that achieve good FID scores, often produce samples that are very similar to specific samples in the training dataset. This is illustrated in Fig. A6, where we show generated samples and their nearest neighbour training sample for SSDNeRF [3], our 40M parameter model, and our 300M parameter model.

Based on results from diffusion models for other representations (*e.g.* Point-E for RGB point clouds, StableDiffusion for images), we expect that given larger datasets and appropriate conditioning mechanisms, the models scale well and are capable to generate diverse samples that are different from the training data. However, on small scale datasets like ShapeNet, our conclusion is that FID alone should be taken with a grain of salt.

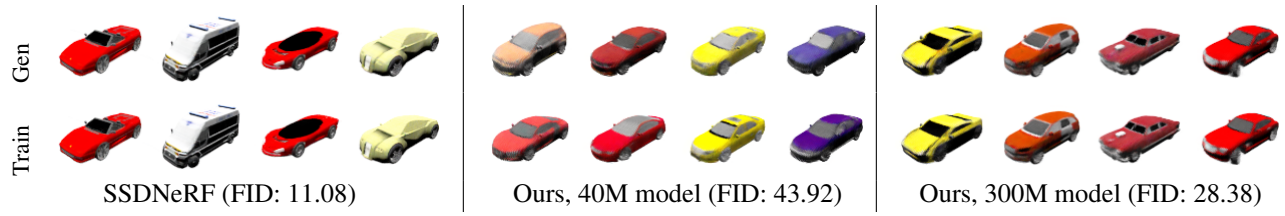

Figure A6. Analysis regarding similarity between generated objects and objects in the training set for SSDNeRF, our 40M parameter model, and our 300M parameter model. The top row shows generated samples and the bottom row shows the nearest neighbour sample in the training dataset. We retrieve the nearest neighbor training object based on the L2 distance between the Inception features of the generated and training images rendered from the same pose. One can observe that SSDNeRF and our 300M parameter model, which achieve good FID scores, generate samples that are very similar to specific samples in the training dataset.

Overfitting and disentangled generation. Likewise, in disentangled generation, we observe that models that achieve good FID scores tend to be more "locked-in". For example in our appearance-only generation, the denoiser outputs are ignored for the point positions and instead their trajectory is forced via the forward diffusion process. We observe that this leads to stronger artifacts for models that achieve better FID scores. To resolve this, for these models, we use a higher number of resampling steps, which reduces the artifacts, but comes at the cost of longer runtimes. Further, we use a higher N_{rev}, *i.e.* use more steps where the denoiser outputs are used to update the point positions via the reverse process. This reduces the artifacts, but comes at the cost of stronger deviations from the given input shape.

We demonstrate this in Fig. A7, where we show non-cherrypicked appearance-only generations on SRN cars for our 40M parameter model and for our 300M parameter model with different disentangled generation parameters. We also observed that for SRN chairs and PhotoShape chairs, the effect is less pronounced.

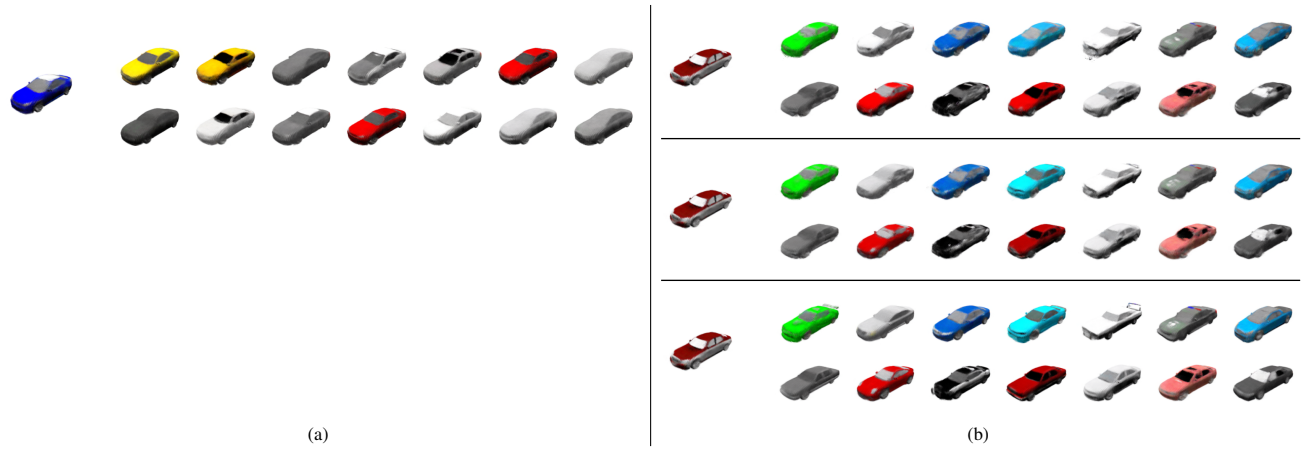

Figure A7. (a) Appearance-only generations on SRN cars for our 40M parameter model with parameters $N_{\text{rev}} = 2$, $N_{\text{repaint}} = 50$, $N_{\text{resample}} = 10$. (b) Appearance-only generations for our 300M parameter model. The top block uses the same parameters. One can observe that for this "more locked-in model", this results in more artifacts. The middle block increases the number of resampling steps: $N_{\text{rev}} = 2$, $N_{\text{repaint}} = 80$, $N_{\text{resample}} = 40$. This reduces artifacts, but comes with longer runtimes. The bottom block increases the number of final steps N_{rev} where the point position trajectory is not enforced via the forward process: $N_{\text{rev}} = 15$, $N_{\text{repaint}} = 80$, $N_{\text{resample}} = 40$. This basically resolves the artifacts, but the shape of the generated samples deviates more from the input shape.

A4. Qualitative results for unconditional generation on ShapeNet Cars

Figure A8 shows unconditional generations of the proposed NPCD model trained on ShapeNet Cars.

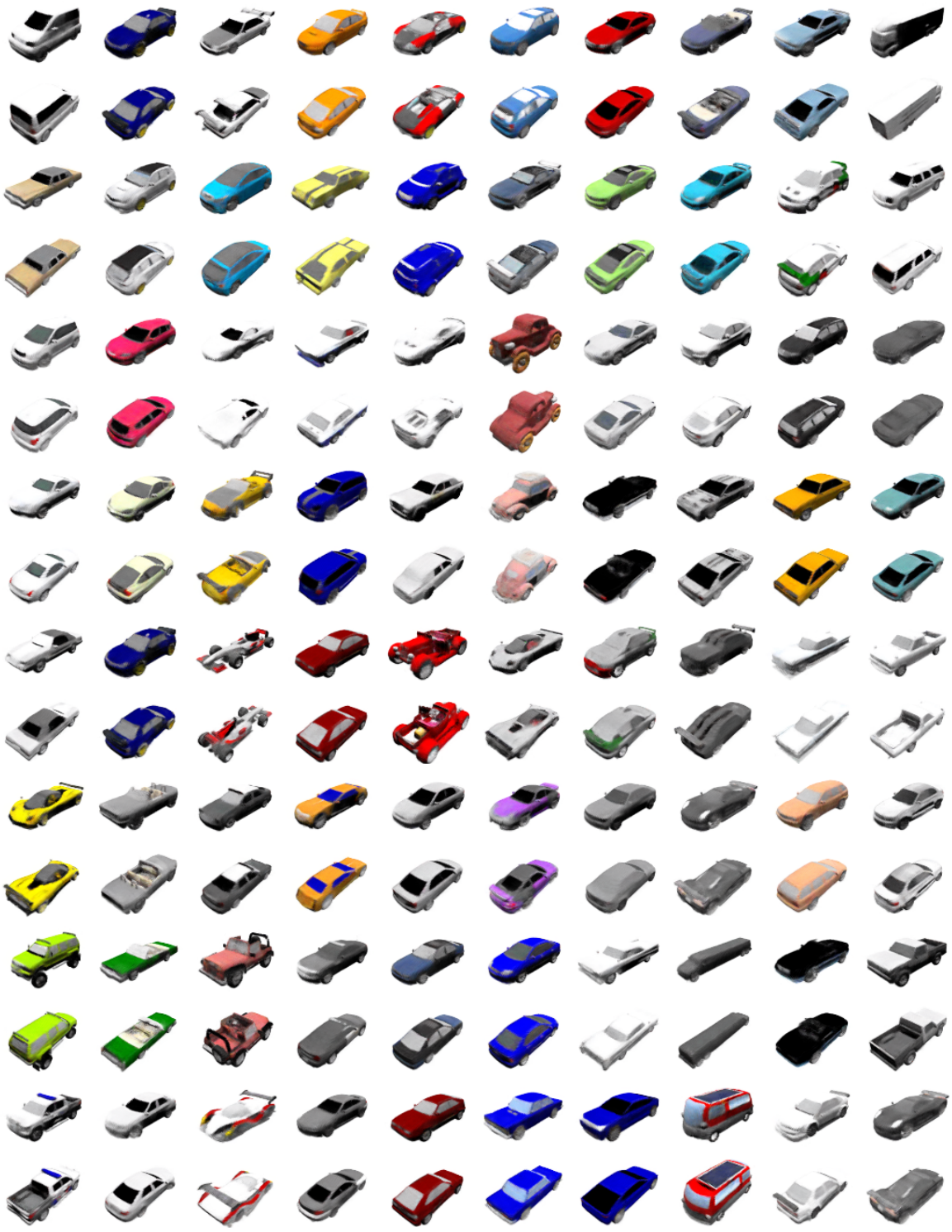

Figure A8. Unconditional generations from the proposed NPCD model trained on ShapeNet Cars. Each generated object is visualized from two different viewpoints. Note that the shown objects are not cherry-picked.

A5. Qualitative results for unconditional generation on ShapeNet Chairs

Figure A9 shows unconditional generations from the proposed NPCD model trained on ShapeNet Chairs.

Figure A9. Unconditional generations of the proposed NPCD model trained on ShapeNet Chairs. Each generated object is visualized from two different viewpoints. Note that the shown objects are not cherry-picked.

A6. Baseline details

For our comparisons on ShapeNet Cars, ShapeNet Chairs, and PhotoShape Chairs, we retrain GRAF and Disentangled3D on these datasets. For training, both approaches require a dataset of images, the distribution of camera poses p_ξ that correspond to the images, and a known camera matrix $\mathbf{K}$. Further, the volumetric rendering in both approaches requires given near and far clipping planes. Training is conducted in a GAN framework. The generator renders images for randomly sampled shape and appearance latent codes and randomly sampled camera poses. The discriminator compares generated and real images.

ShapeNet Cars. For training on ShapeNet Cars, according to the the SRN rendering parameters, we set the radius to 1.3, the field of view to 52 (chosen such that it results in the correct focal length), and sample camera poses from the full hemisphere. To define the near and far planes, we compute the cube that bounds all pointclouds. For the Cars training split, this gives:

- x-coordinate range: -0.30903995 to 0.30898672
- y-coordinate range: -0.48859358 to 0.49043328
- z-coordinate range: -0.27073205 to 0.2709335

The nearest possible point hence has the following distance to the camera:
$1.3 - \sqrt{0.30903995^2 + 0.49043328^2 + 0.2709335^2} = 1.3 - 0.6398714 = 0.6601285815238953$.
The furthest point hence has the following distance to the camera: $1.3 + 0.6398714 = 1.9398714184761048$.
Based on this, we set the near and far planes to 0.5 and 2.1.

ShapeNet Chairs. For training on ShapeNet Cars, according to the the SRN rendering parameters, we set the radius to 2.0, the field of view to 52 (chosen such that it results in the correct focal length), and sample camera poses from the full hemisphere. To define the near and far planes, we compute the cube that bounds all pointclouds. For the Chairs training split, this gives:

- x-coordinate range: -0.5 to 0.5
- x-coordinate range: -0.5 to 0.5
- x-coordinate range: -0.5 to 0.5

The nearest possible point hence has the following distance to the camera: $2.0 - \sqrt{0.5^2 + 0.5^2 + 0.5^2} = 2.0 - 0.87 = 1.13$.
The furthest point hence has the following distance to the camera: $2. + 0.87 = 2.87$.
Based on this, we set the near and far planes to 1.0 and 3.0.

PhotoShape Chairs. For training on PhotoShape Chairs, according to the PhotoShape Chairs rendering parameters, we set the radius to 2.5 and the field of view to 39.6 (chosen such that it results in the correct focal length). As PhotoShape Chairs views were always rendered from the same set of camera poses, we randomly sample poses from this discrete set during training. As the 3D object shapes are the same as on ShapeNet Chairs, we compute the near and far clipping planes comparably to ShapeNet Chairs. Given the different radius, this results in near and far planes of 1.25 and 3.75.

A7. Comparison to a Combination of Shape and Texture Generation

Recently, multiple works applied diffusion models for 3D shape generation [6, 10, 11, 13, 14] and for the generation of textures for a given 3D mesh [1, 2, 12]. Combining the two approaches, allows generating a shape, followed by generation of different appearances. For completeness, in the following, we conduct a comparison to such a combination. However, please note that separate 3D shape and texture generation differs strongly in concept from our work and has different assumptions and results: **(1)** with separate generation it is only possible to re-generate the appearance for a given shape, but not vice-versa. **(2)** SOTA texture generation approaches use test-time optimization with score distillation from a 2D diffusion model, which results in generation times in the order of minutes (we measure 9 minutes for Text2Tex), where our method takes only a few seconds to generate a sample and is thus orders of magnitude faster. **(3)** Both approaches differ strongly in terms of training data: our model uses a rather small dataset of 3D objects (a few thousand), whereas Text2Tex builds on a diffusion model trained on large scale image data (on the order of billions).

Despite these fundamental differences, we provide a comparison between our method and the mentioned "Locally Attentional SDF Diffusion" plus "Text2Tex" models (running their code as it is on Github) below. The following shows four resulting appearance-only generations for text prompts "`a <black/red/green/blue> chair`" in comparison to a sample from our method with resampled appearance:

Figure A10. Comparison of our model and a combination of Locally Attentional SDF Diffusion [14] plus Text2Tex [2].

It can be seen that our method finds a good trade-off between quality and generation time. In general, we think that both, generative modeling on 3D representations and generative modeling on 2D images plus score distillation, are interesting research directions, but have different advantages and disadvantages.

References

[1] Tianshi Cao, Karsten Kreis, Sanja Fidler, Nicholas Sharp, and KangXue Yin. Texfusion: Synthesizing 3d textures with text-guided image diffusion models. In *ICCV*, 2023. 11

[2] Dave Zhenyu Chen, Yawar Siddiqui, Hsin-Ying Lee, Sergey Tulyakov, and Matthias Nießner. Text2tex: Text-driven texture synthesis via diffusion models. In *ICCV*, 2023. 11

[3] Hansheng Chen, Jiatao Gu, Anpei Chen, Wei Tian, Zhuowen Tu, Lingjie Liu, and Hao Su. Single-stage diffusion nerf: A unified approach to 3d generation and reconstruction. In *ICCV*, 2023. 6

[4] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *NeurIPS*, 2022. 2

[5] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020. 2

[6] Muheng Li, Yueqi Duan, Jie Zhou, and Jiwen Lu. Diffusion-sdf: Text-to-shape via voxelized diffusion. In *CVPR*, 2023. 11

[7] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. RePaint: Inpainting using denoising diffusion probabilistic models. In *CVPR*, 2022. 2

[8] Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-E: A system for generating 3d point clouds from complex prompts. *arXiv:2212.08751*, 2022. 2

[9] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023. 2

[10] Jaehyeok Shim, Changwoo Kang, and Kyungdon Joo. Diffusion-based signed distance fields for 3d shape generation. In *CVPR*, 2023. 11

[11] J. Ryan Shue, Eric Ryan Chan, Ryan Po, Zachary Ankner, Jiajun Wu, and Gordon Wetzstein. 3d neural field generation using triplane diffusion. In *CVPR*, 2023. 11

[12] Kim Youwang, Tae-Hyun Oh, and Gerard Pons-Moll. Paint-it: Text-to-texture synthesis via deep convolutional texture map optimization and physically-based rendering. In *CVPR*, 2024. 11

[13] Biao Zhang, Jiapeng Tang, Matthias Niessner, and Peter Wonka. 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. *ACM TOG*, 2023. 11

[14] Xin-Yang Zheng, Hao Pan, Peng-Shuai Wang, Xin Tong, Yang Liu, and Heung-Yeung Shum. Locally attentional sdf diffusion for controllable 3d shape generation. *ACM TOG*, 2023. 11