3D Neural Field Generation using Triplane Diffusion

We provide implementation details and additional experiments in this supplement. Please see the supplementary website (https://jryanshue.com/nfd/) for further visual results, code, and pre-trained models.

A1. Implementation Details

A1.1. Learning a Dataset of Triplane Features

Data. We train our model on 3 separate categories from the ShapeNet V1 dataset: *Cars*, which contains 7496 objects, *Chairs*, which contains 4971 objects, and *Planes*, which contains 4045 objects. We train a separate model for each class of objects.

Watertighting. As a preprocessing step, we convert meshes from the ShapeNet dataset into watertight meshes. We perform watertighting with the implementation and settings from Mescheder et al. [7]. We render depth images from 20 views from a dodecahedron, which gives equally spaced views, and use the marching cubes algorithm [5] to extract a watertight mesh.

Computing ground truth occupancy. We follow the implementation of [7] for computing occupancy values for arbitrary 3D coordinates. For any point in 3D space, we compute the occupancy value of the point by casting a ray along the z-axis and counting the number of intersections with the watertight mesh—an odd number of intersections means the point is inside the watertight shape. When computing our dataset, we draw half of our query points uniformly at random from the volume, while the rest are importance sampled near the surface of the watertight mesh.

Triplane Features We used triplane features of dimension $128 \times 128 \times 32 \times 3$. While higher triplane resolutions guarantee lower degradation of decoded ground truth meshes, the increased dimensionality also places a burden on time and memory constraints. We initialize the triplane features to values drawn from a normal distribution with standard deviation 0.1.

Shared MLP. Our MLP is designed to be lightweight to enable quick training and inference. Our MLP is composed of a Fourier feature mapping layer [9] with a scale factor of 1, followed by 3 fully connected layers of dimension 128, each with ReLU activation functions.

Training. As discussed in the main manuscript, we train our triplanes and MLP in two stages: first jointly on a subset of data, then independently on each object in the dataset, with a frozen MLP. During the first stage, we train on 500 randomly selected shapes with a batch size of 1 object per iteration and 500k occupancy values points per object. We train this first stage for 200 epochs with a learning rate of 1e-3. Training was conducted on a single RTX 2080ti, and took approximately 1 day to complete. The shared MLP is then frozen and used to train triplane features for every object object in the dataset. During this second stage, we train triplane features for each object individually. We use a batch size of 200k occupancy values per object and train for 30 epochs with a learning rate of 1e-3. This stage takes 10 minutes to train on an RTX 2080ti per shape, but can be parallelized across an arbitrary number of GPUs. The resulting triplane features are used as pseudo-ground truth images for training the diffusion model.

A1.2. Training a Diffusion Model for Triplane Features

We base our implementation on the official code-base of [2], available at https://github.com/openai/guided-diffusion. Unless otherwise stated, DDPM hyperparameters are identical to the class-specific LSUN model in [2].

Diffusion Model Training. We train all models with a batch size of 128 across 8 A6000 GPUs. For cars, we used a learning rate of 1e-4 while for chairs and planes, a lower learning rate of 3e-5 helped prevent instability during training. We trained cars, chairs, and planes for 400k, 200k, and 200k steps respectively. Cars took around 6 days to train while chairs and planes each took approximately 3 days. The cars model was pretrained on a subset of the cars data for 160k steps before training on the full dataset for the remaining 240k iterations.

Normalization. The learned triplane feature images, with which we train our diffusion model, are regularized (see Sec. A2) but still theoretically unbounded, and we find outliers to skew the distribution. We apply normalizization to ensure the values of the triplane feature images to be within a fixed range. We normalize the feature channels to zero-mean and clip each channel to be within $S = 16$ standard deviations of the mean. We then scale each channel to be within the range $[-1, 1]$.

Sampling at inference. When generating shapes, we default to using a DDPM with 1000 iterations. Generating a set of triplane features for a single example takes roughly 20 seconds on a single A6000 GPU, but the number of iterations can be decreased to 250 for faster generation and a small (judged visually) reduction in fidelity. Decoding the resulting occupancy field and extracting a mesh at a resolution of 128^3 takes about 5 seconds per mesh, including both MLP evaluation and marching cubes.

Interpolation. We used DDIM [8] to sample shapes for interpolation. We noticed visually worse-quality meshes in the DDIM setting compared to the DDPM setting. Cars, chairs, and planes were sampled with 25, 250, and 25 steps respectively, though we noticed only small differences when the number of steps was changed.

A2. Triplane Regularization

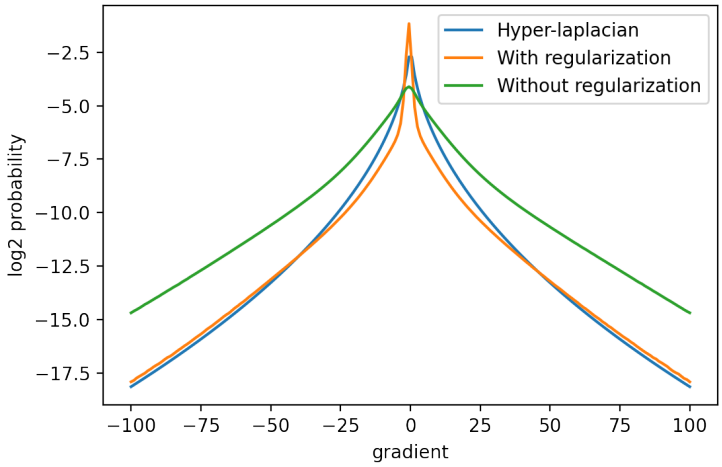

Figure A1. Distribution of image gradients for natural images and triplane features with and without regularization. After regularization, image gradients of ground truth triplane features closely resembles gradients found in natural images. Natural image gradients modelled by a hyper-Laplacian with $\alpha = 0.5$ per Krishnan et al. [4].

State-of-the-art diffusion models have empirically performed well when trained on natural images. However, without proper regularization, ground truth triplanes trained using an autodecoder result in high frequency artifacts as shown in Figure 7. We apply TV regularization as illustrated in Equation 4, resulting in smoother triplane features that are more similar to the manifold of natural images.

Krishnan et al. [4] found that gradients of natural images are closely modelled by a hyper-Laplacian with $0.5 \leq \alpha \leq 0.8$. Supplementary Figure A1 shows the distribution of gradients of natural images modelled by a hyper-Laplacian with $\alpha = 0.5$ and gradients of trained triplane features with and without TV regularization. Gradients of triplanes trained with regularization closely resemble gradients found in natural images.

A3. Ablation on Triplane Resolution, Channel Depth, and Complexity

Fig. A2 provides an ablation over triplane resolution and channel depth for fitting triplanes, while Fig. A3 examines computational complexity for the same task. We observe that while increased triplane resolution and channel depth can help capture high-frequency details in the fitted shape, computational complexity is linear in the number of parameters of the triplane, i.e., quadratic in triplane resolution. Operating with limited computational resources, we selected $128^2 \times 32$ triplanes as a balance between speed and quality.

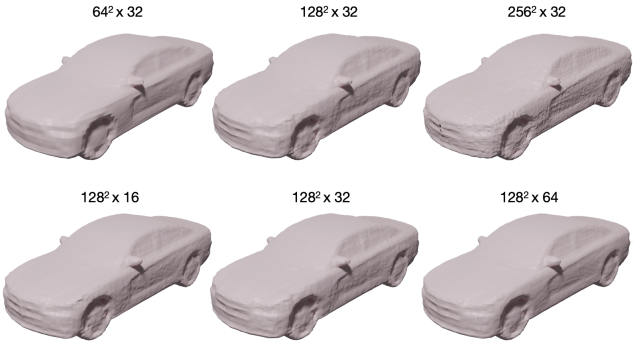

Figure A2. Ablation over triplane resolution and channel depth.

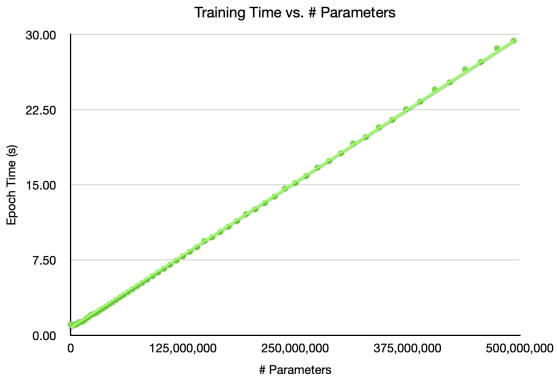

Figure A3. Computational complexity for training triplanes.

A4. Controlled generation with SDEdit

While we aimed to focus on unconditional generation to provide a foundation for future work, we include an example of guided generation in Fig. A4, following SDEdit [6].

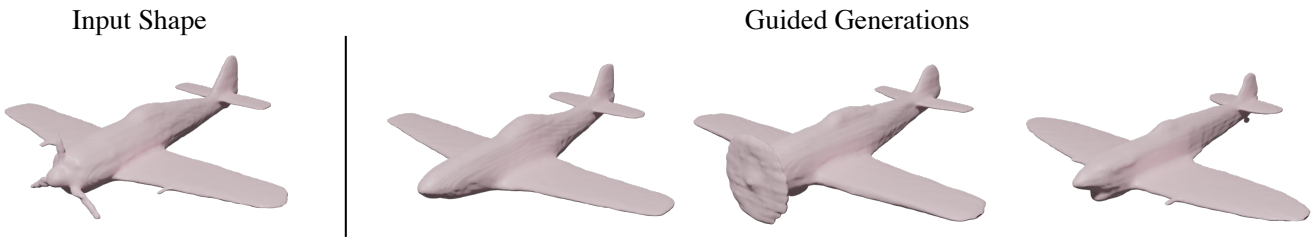

Figure A4. An example of guided generation using our model.

A5. Exploring Radiance Field Generation

We provide preliminary results using our approach for radiance field generation in Fig. A5, showing its potential for other types of neural fields. For this proof of concept, we train our diffusion model on triplanes synthesized by EG3D [1].

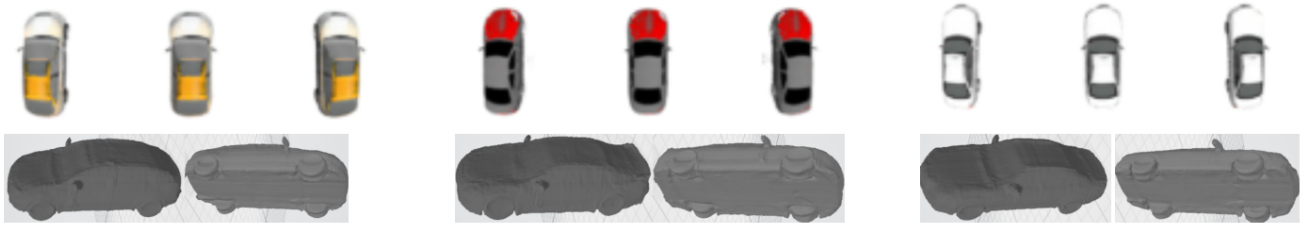

Figure A5. Preliminary results depicting renderings and shapes produced by a variant of our method trained to synthesize radiance fields.

A6. Generated Samples

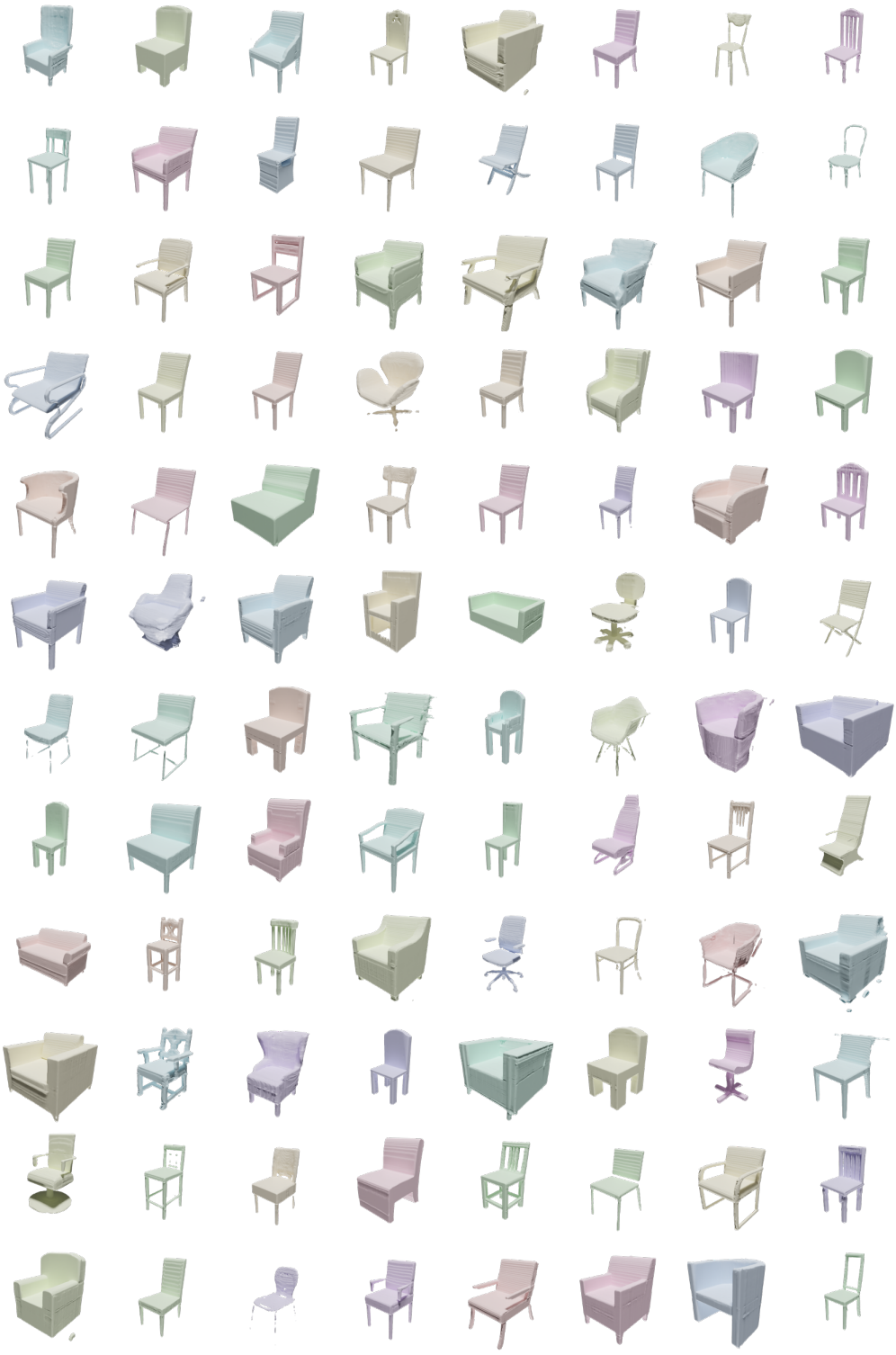

Figure A6. Set of 96 uncurated samples generated from our model trained on the *chairs* category of ShapeNet.

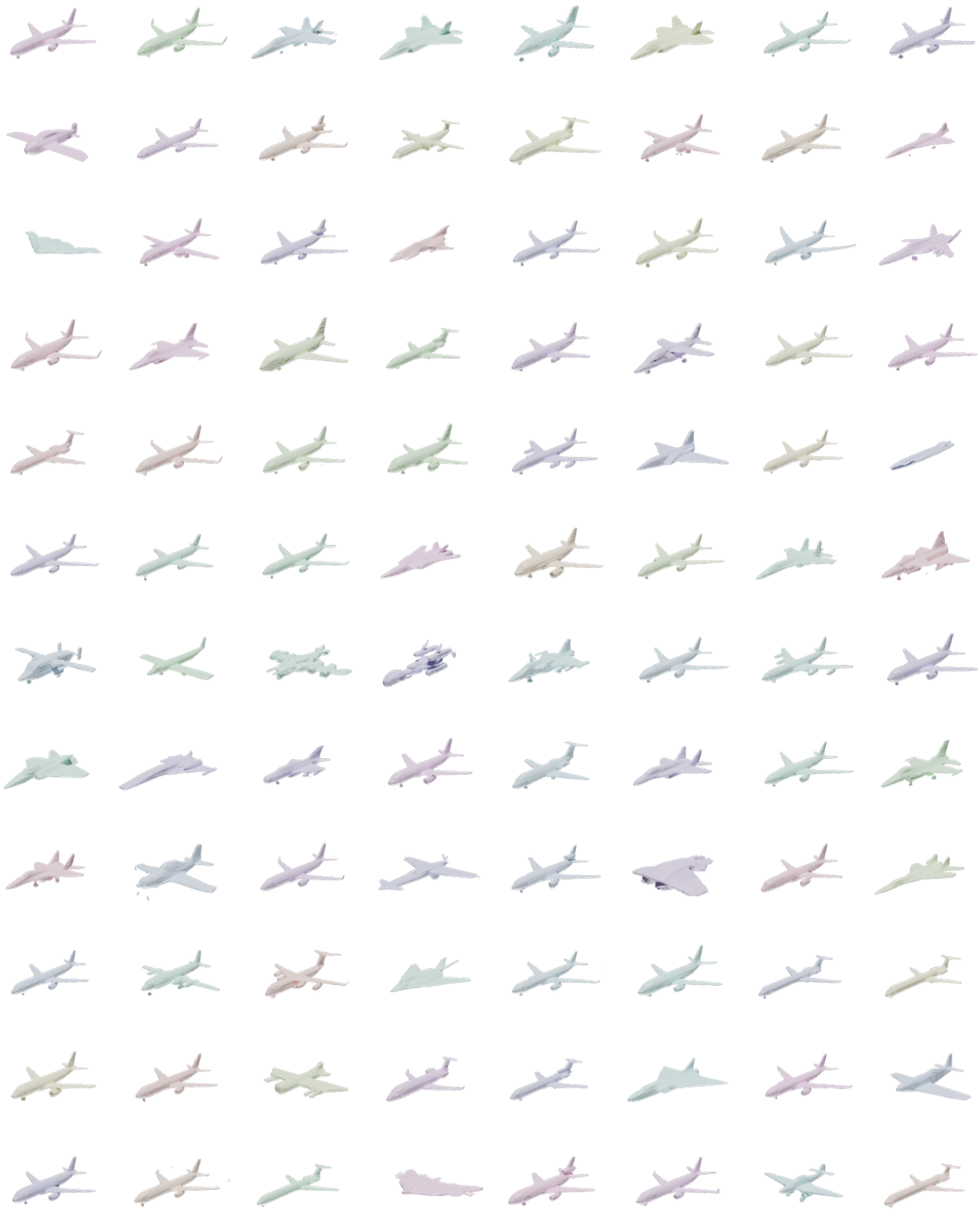

Figure A7. Set of 96 uncurated samples generated from our model trained on the *planes* category of ShapeNet.

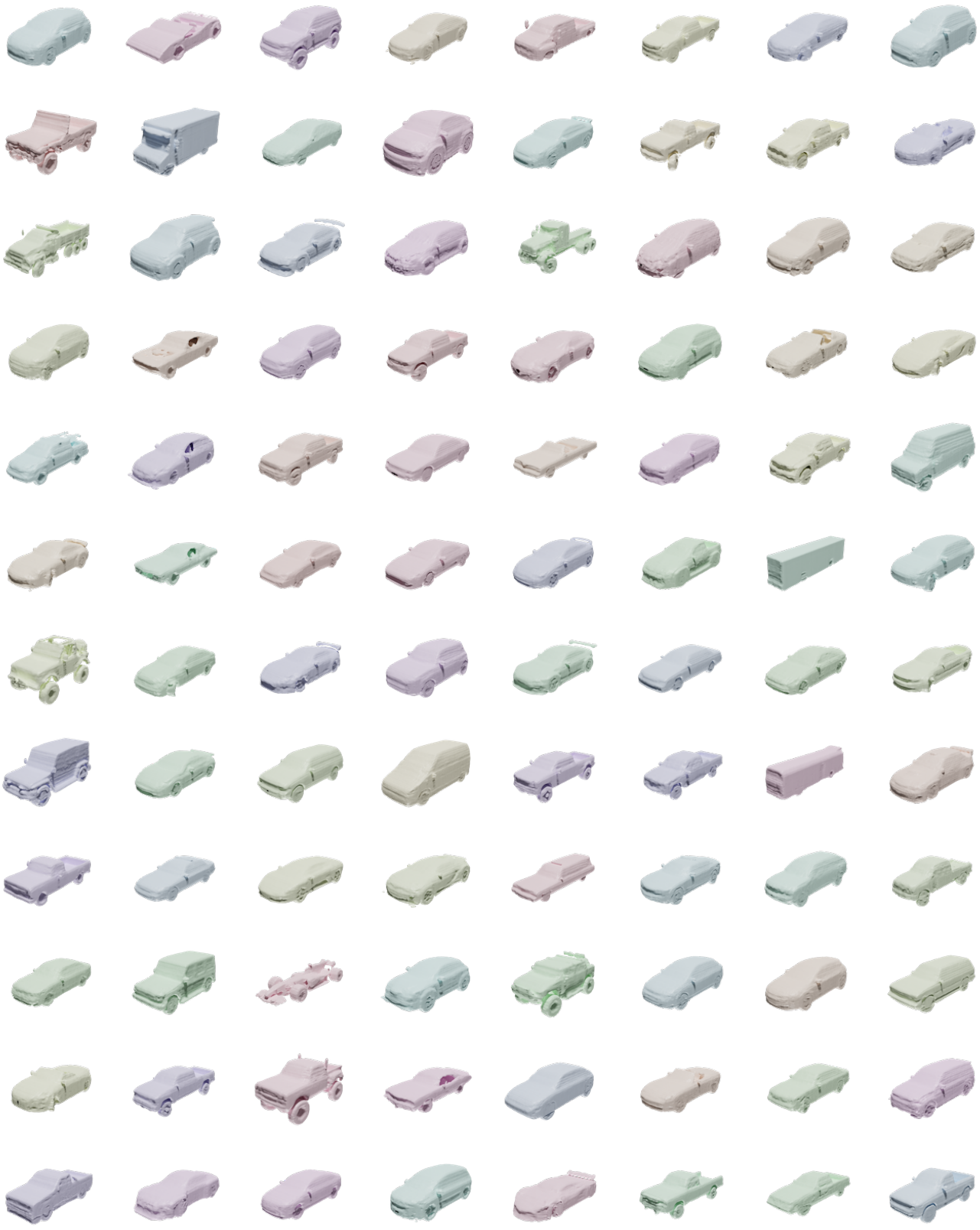

Figure A8. Set of 96 uncurated samples generated from our model trained on the *cars* category of ShapeNet.

A7. Additional Baselines

Fig. A9 depicts samples produced by Implicit-Grid [3]. Tab. A1 provides a comparision against Implicit-Grid [3] for ShapeNet Cars, Chairs, and Planes, as well as against EG3D [1] for ShapeNet Cars.

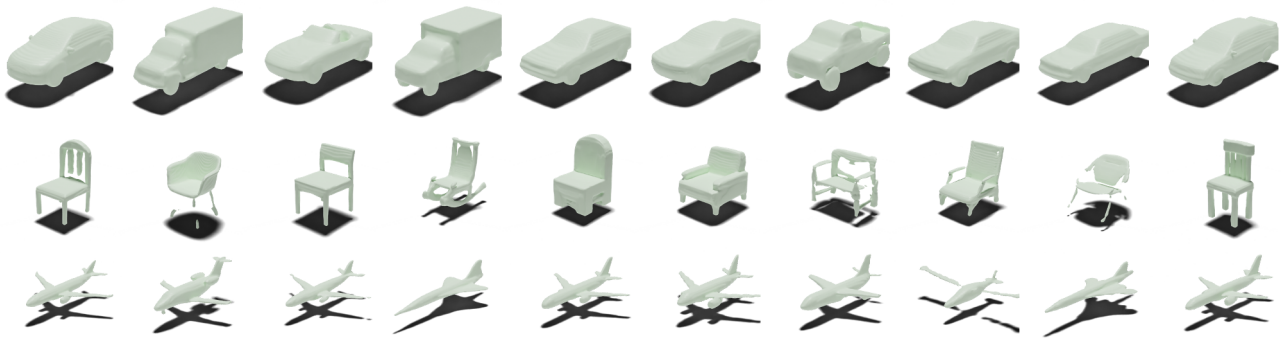

Figure A9. Generated shapes using Implicit-Grid baseline method [3].

Data	Method	FID ↓	Precision ↑	Recall ↑
Cars	PVD*	335.8	0.1	0.2
	Implicit-Grid	209.3	25.9	21.5
	SDF-StyleGAN	98.0	35.9	36.2
	EG3D	122.5	22.9	16.1
	NFD (Ours)	**83.6**	**49.5**	**50.5**
Chairs	PVD*	305.8	0.2	1.7
	Implicit-Grid	119.5	74.8	77.2
	SDF-StyleGAN	36.5	90.9	87.4
	NFD (Ours)	**26.4**	**92.4**	**94.8**
Planes	PVD*	244.4	2.7	3.8
	Implicit-Grid	145.4	67.1	66.2
	SDF-StyleGAN	65.8	64.5	72.8
	NFD (Ours)	**32.4**	**70.5**	**81.1**

Table A1. Comparison of evaluation metrics with baseline methods. Our method outperforms all baselines in FID, precision and recall, illustrating that our method generates high quality and diverse 3D shapes.

References

[1] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J. Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3d generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16123–16133, June 2022. 4, 8

[2] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 8780–8794. Curran Associates, Inc., 2021. 1

[3] Moritz Ibing, Isaak Lim, and Leif P. Kobbelt. 3d shape generation with grid-based implicit functions. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13554–13563, 2021. 8

[4] Dilip Krishnan and Rob Fergus. Fast image deconvolution using hyper-laplacian priors. In *NIPS*, 2009. 3

[5] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM TOG*, 1987. 1

[6] Chenlin Meng, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. Sdedit: Image synthesis and editing with stochastic differential equations. *arXiv preprint arXiv:2108.01073*, 2021. 4

[7] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4455–4465, 2019. 1

[8] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021. 2

[9] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *ArXiv*, abs/2006.10739, 2020. 1