Supplementary Material for pixelNeRF: Neural Radiance Fields from One or Few Images

Anonymous CVPR 2021 Submission
Paper ID 1291

In this supplementary document, we discuss implementation details and provide additional results.

A. Additional Results

In this section, we provide additional qualitative and quantitative results for several key experiments. The reader is encouraged to refer to the video and website for a richer, animated presentation of qualitative results.

A.1. Category-agnostic ShapeNet: Random Results

We show *randomly* sampled results for the category-agnostic setting (§ 5.1.2) in Fig. 1, Fig. 2, and Fig. 3. Specifically, we sample 6 uniformly random objects for each of the 13 largest ShapeNet categories and show comparisons to the baselines [5, 7, 9] as in the main paper. Two random views are selected from the 24 available views to be source and target views respectively.

A.2. Generalization to novel categories

In Table 1 we show a detailed breakdown of metrics by category on unseen categories, as promised in the main paper.

A.3. Two-object Scenes

We show samples from our rendered dataset in Fig. 4. An analysis of performance as more views become available is in Table 2, for our method when compared with SRN. We also show *randomly sampled* results of scenes when given two input views in Figure 5. We train our model using two random views, and give the model either one, two, or three fixed informative views during inference.

A.4. DTU

In Fig. 6, we show quantitative results for each scene as well as renderings of of all test scenes not shown in the main paper.

In Table 3 we provide means and standard deviations of metrics for our method and NeRF on the DTU test set, with 1, 3, 6, 9 views. The PSNR here was plotted in Fig. 10 of the main paper

B. Reproducibility

B.1. Implementation Details

Here we describe implementation details in the interest of reproducibility. A general remark is that due to the high compute cost, we did not spend significant effort to tune the architecture or training procedure, and it is possible that variations can perform better, or that smaller models may suffice.

Encoder E As briefly discussed in the main paper, we use a ResNet34 backbone and extract a feature pyramid by taking the feature maps prior to the first pooling operation and after the first ResNet 3 layers. For a $H \times W$ image, the feature maps have shapes

1. $64 \times H/2 \times W/2$
2. $64 \times H/4 \times W/4$
3. $128 \times H/8 \times W/8$
4. $256 \times H/16 \times W/16$

These are upsampled bilinearly to $H/2 \times W/2$ and concatenated into a volume of size $512 \times H/2 \times W/2$. For a 64×64 image, to avoid losing too much resolution, we skip the first pooling layer, so that the image resolutions are at $1/2, 1/2, 1/4, 1/8$ of the input rather than $1/2, 1/4, 1/8, 1/16$. We use ImageNet pretrained weights provided through PyTorch.

NeRF network f We employ a fully-connected ResNet architecture with 5 ResNet blocks and width 512, similar to that in [7]. To enable arbitrary number of views as input, we aggregate across the source-views after block 3 using an average-pooling operation. This architecture is illustrated in Fig. 8. We remark that due to computational cost, we did not tune this architecture very much in practice.

Hierarchical volume sampling To improve the sampling efficiency, in practice, we also use *coarse* and *fine* NeRF networks f_c, f_f as in the vanilla NeRF [6], both of which

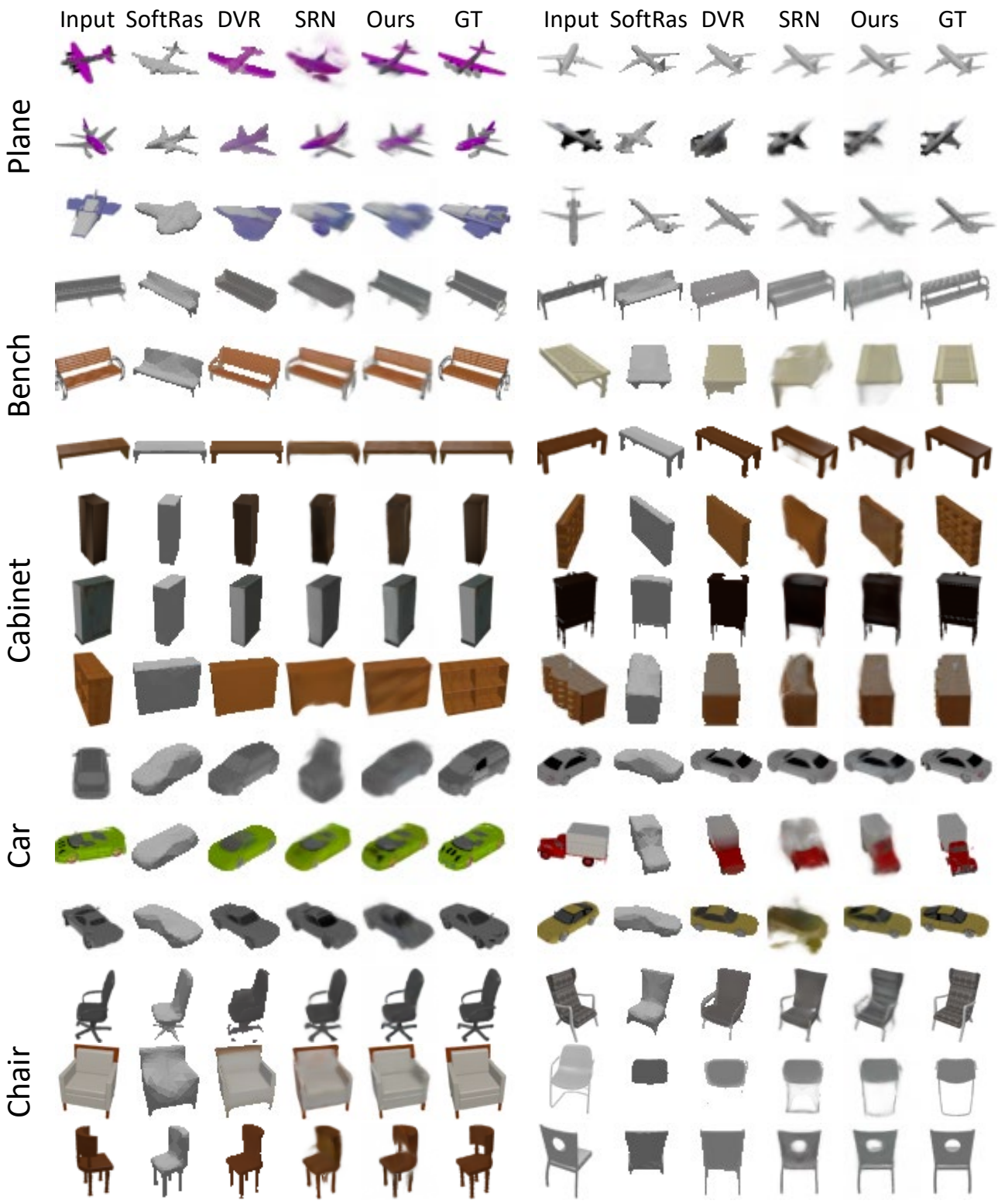


Figure 1: **Randomly sampled results.** part 1

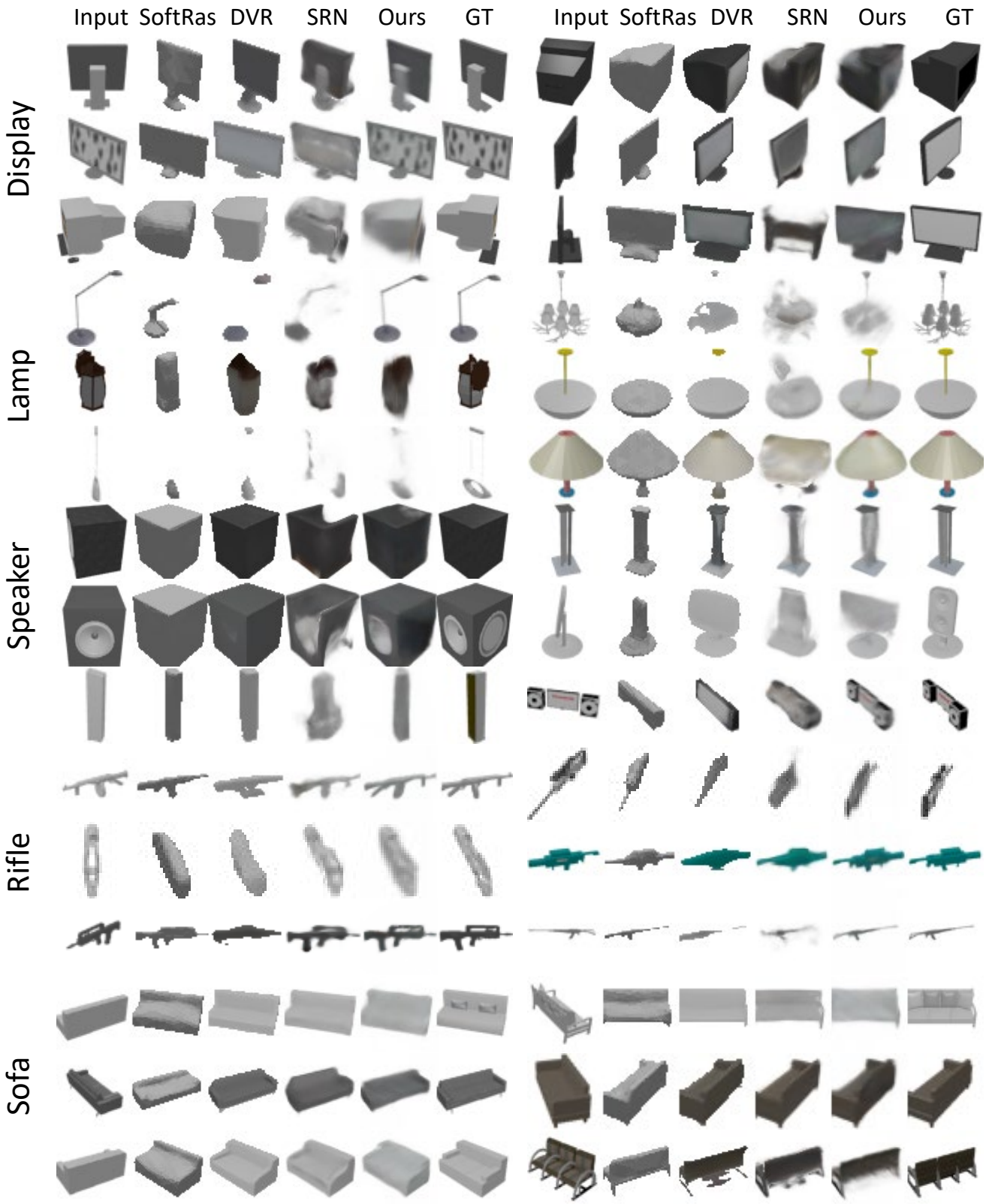


Figure 2: **Randomly sampled results.** part 2

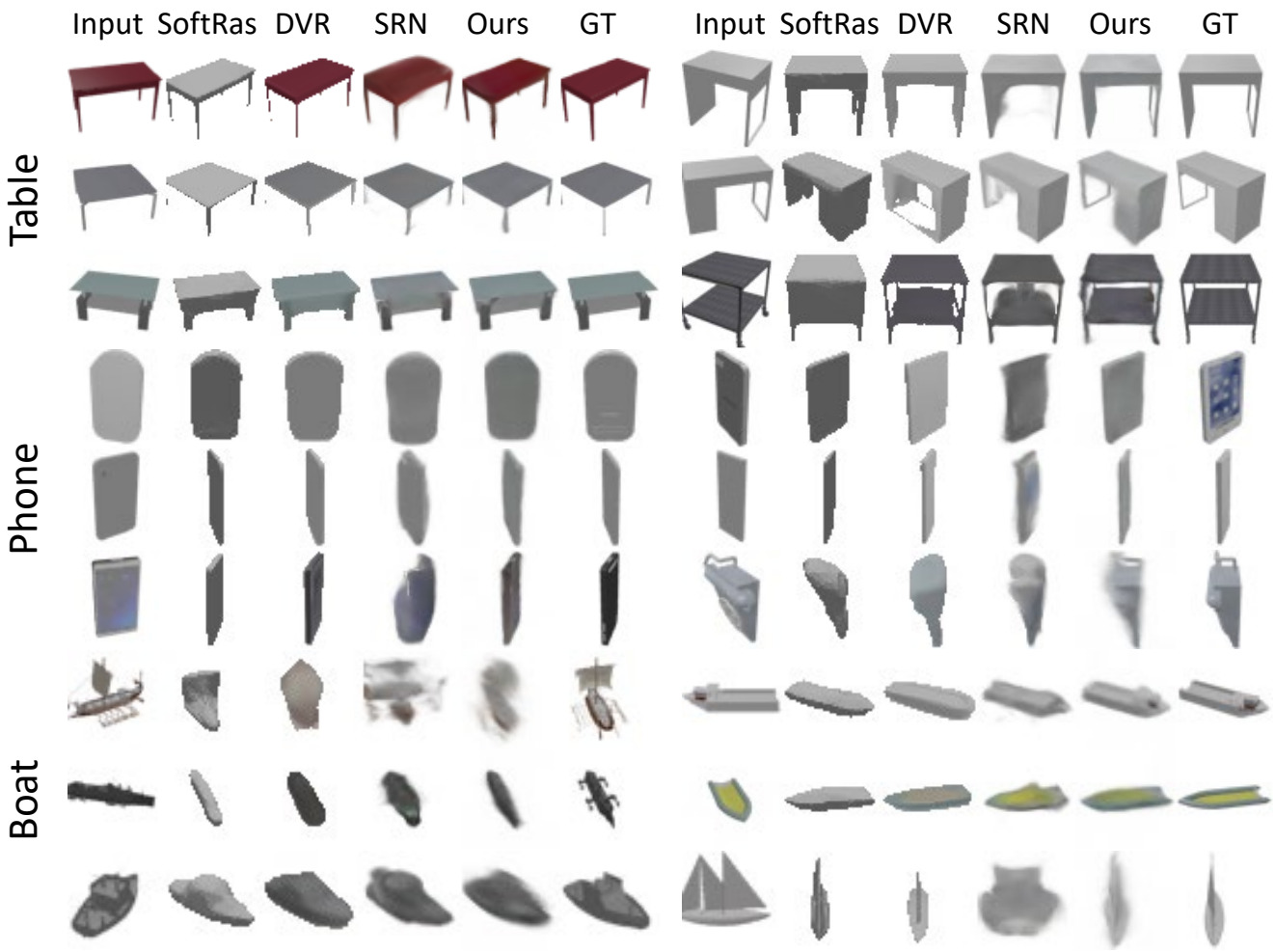


Figure 3: **Randomly sampled results.** part 3

share an identical architecture described above. Note that the encoder E is not duplicated.

More precisely, we use 64 stratified uniform and 16 importance samples, and additionally take 16 fine samples with a normal distribution (SD 0.01) around the expected ray termination (i.e. depth) from the coarse model, to further promote denser sampling near the surface.

NeRF rendering hyperparameters We use positional encoding γ from NeRF for the spatial coordinates, with exponentially increasing frequencies:

$$\gamma(\mathbf{x}) = \begin{pmatrix} \sin(2^0\omega\mathbf{x}) \\ \cos(2^0\omega\mathbf{x}) \\ \sin(2^1\omega\mathbf{x}) \\ \cos(2^1\omega\mathbf{x}) \\ \vdots \\ \sin(2^{L-1}\omega\mathbf{x}) \\ \cos(2^{L-1}\omega\mathbf{x}) \end{pmatrix} \tag{1}$$

Note that we do not apply the encoding to the view directions. In all experiments, we set $L = 6$. We also concatenate the input coordinates along the encoding as in the NeRF implementation. ω is a scaling factor, set (rather arbitrarily) to 1.5 for the single-category, category-agnostic ShapeNet experiments as well as the DTU experiment, and to 2.0 for the multi-object experiment. While the exponent base can be tuned, in practice we left it at 2 as in NeRF.

		bench	cbnt.	disp.	lamp	spkr.	rifle	sofa	table	phone	boat	mean
↑ PSNR	DVR	18.37	17.19	14.33	18.48	16.09	20.28	18.62	16.20	16.84	22.43	17.72
	SRN	18.71	17.04	15.06	19.26	17.06	23.12	18.76	17.35	15.66	24.97	18.71
	Ours	**23.79**	**22.85**	**18.09**	**22.76**	**21.22**	**23.68**	**24.62**	**21.65**	**21.05**	**26.55**	**22.71**
↑ SSIM	DVR	0.754	0.686	0.601	0.749	0.657	0.858	0.755	0.644	0.731	0.857	0.716
	SRN	0.702	0.626	0.577	0.685	0.633	0.875	0.702	0.617	0.635	0.875	0.684
	Ours	**0.863**	**0.814**	**0.687**	**0.818**	**0.778**	**0.899**	**0.866**	**0.798**	**0.801**	**0.896**	**0.825**
↓ LPIPS	DVR	0.219	0.257	0.306	0.259	0.266	0.158	0.196	0.280	0.245	0.152	0.240
	SRN	0.282	0.314	0.333	0.321	0.289	0.175	0.248	0.315	0.324	0.163	0.280
	Ours	**0.164**	**0.186**	**0.271**	**0.208**	**0.203**	**0.141**	**0.157**	**0.188**	**0.207**	**0.148**	**0.182**

Table 1: **Generalization to novel categories**. Expanding on Table 5 in the main paper, we show quantitative results with a breakdown by category.

	1-view			2-view			3-view		
	↑ PSNR	↑ SSIM	↓ LPIPS	↑ PSNR	↑ SSIM	↓ LPIPS	↑ PSNR	↑ SSIM	↓ LPIPS
SRN	13.76	0.658	0.422	14.28	0.660	0.432	14.67	0.664	0.431
Ours	**20.15**	**0.767**	**0.274**	**23.40**	**0.832**	**0.207**	**23.68**	**0.800**	**0.206**

Table 2: **Performance on synthetic two-object dataset with increasing number of views at test time.** Image quality metrics for SRN and our method, when increasing the number of views given at test time.

The sampling bounds were set manually for each dataset. They were $[1.25, 2.75]$ for ShapeNet chairs, $[0.8, 1.8]$ for ShapeNet cars, $[1.2, 4.0]$ for Kato et al. [2] renderings (category agnostic, novel category), $[4.0, 9.0]$ for our rendered 2-object dataset, and $[0.1, 5.0]$ for input.

We use a white background color in NeRF to match the ShapeNet renderings, except in the DTU setup where a black background is used.

Model implementation We implement all models using the PyTorch [8] framework.

B.2. Experimental Details

We first provide general details about the metrics and training procedure common to all experiments, then present more specific details for each experimental setting in subsections.

Metric details We use PSNR and SSIM from the scikit-image [10] package as in SRN [9], whereas LPIPS is computed with the code provided by the LPIPS authors [12] after normalizing the pixel values to the $[-1, 1]$ range. We use the VGG network version of LPIPS following NeRF [6].

Training For all experiments, we take the learning rate to be 10^{-4}. We use a batch size of 4 instances and 128 rays per instances.

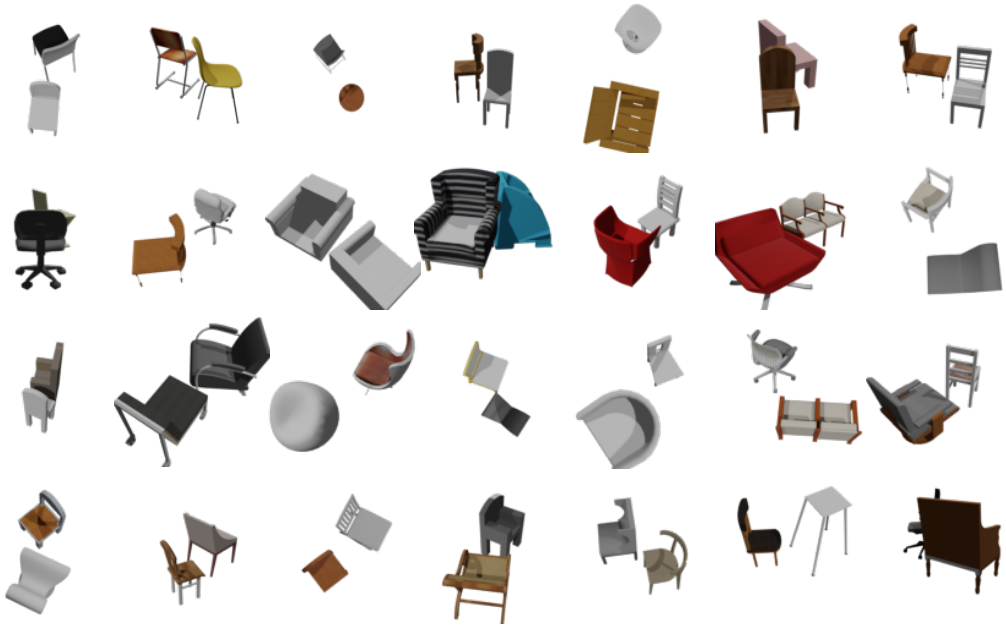
B.2.1 Single-category ShapeNet

We train for 400000 iterations, which took roughly 6 days on a single Titan RTX. For efficiency, we sample rays from within a tight bounding box around the object for the first 300000 iterations, after which we remove the bounding box to avoid background artifacts. Further, we use 2 input views for the first 300000 iterations and after that, we randomly choose to take either 1 or 2 views as input to encourage the model to work with either 1 or 2 views.

SRN's evaluation protocol is followed: in the 1-view case, we use view 64 as input, and in the 2-view case, we use views 64 and 128.

Baselines For SRN [9], we use the pretrained chair model from the public GitHub repository. Note that SRN requires a test-time training step (latent inversion) to generate result images; we apply latent inversion for 170000 iterations for both the 1-view and 2-view cases for chairs.

Recall that, due to a camera sampling bug, we use an updated car dataset provided by the SRN author. Thus, we



(a) Train set

(b) Test set

Figure 4: Randomly sampled images from the synthetic two-object scene dataset

follow instructions in the Github to train a model on the new dataset; we train for 400000 iterations and apply latent inversion for 100000 iterations for each of the 1-view and 2-view cases. Note the quantitative results we report are slightly lower than that in [1] in the single-view case, but substantially higher than in the original SRN paper, which used the bugged renderings. For the remaining baselines, we only report numbers from the relevant papers on the same task.

Figure 5: Randomly sampled results for two object scenes, when given two input views.

		1 View			3 View			6 View			9 View		
		PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
Ours	Mean	**15.55**	**0.537**	**0.535**	**19.33**	**0.695**	**0.387**	**20.43**	**0.732**	0.361	21.10	0.758	0.337
	SD	1.87	0.127	0.081	2.59	0.131	0.105	2.66	0.115	0.102	2.71	0.102	0.094
NeRF	Mean	8.00	0.286	0.703	9.85	0.374	0.622	18.59	0.719	**0.347**	**22.14**	**0.820**	**0.262**
	SD	3.20	0.093	0.055	4.69	0.173	0.137	4.72	0.177	0.133	4.33	0.131	0.109

Table 3: **DTU aggregate metrics vs. NeRF**. Expanding on Fig. 10 in the main paper, we compare our method to NeRF on DTU test scenes quantitatively. Recall higher is better for PSNR and SSIM, while lower is better for LPIPS. Note that PixelNeRF is a feed-forward method, while a NeRF was optimized for 14 hours for each scene and set of input views.

Full Name	cabinet	display	speaker
Abbreviation	cbnt.	disp.	spkr.

Table 4: ShapeNet category name abbreviations.

B.2.2 Category-agnostic ShapeNet

We train our model for 800000 iterations on the entire training set, where rays are sampled from within a tight bounding box for the first 400000 iterations. This took about 6 days on an RTX 2080Ti.

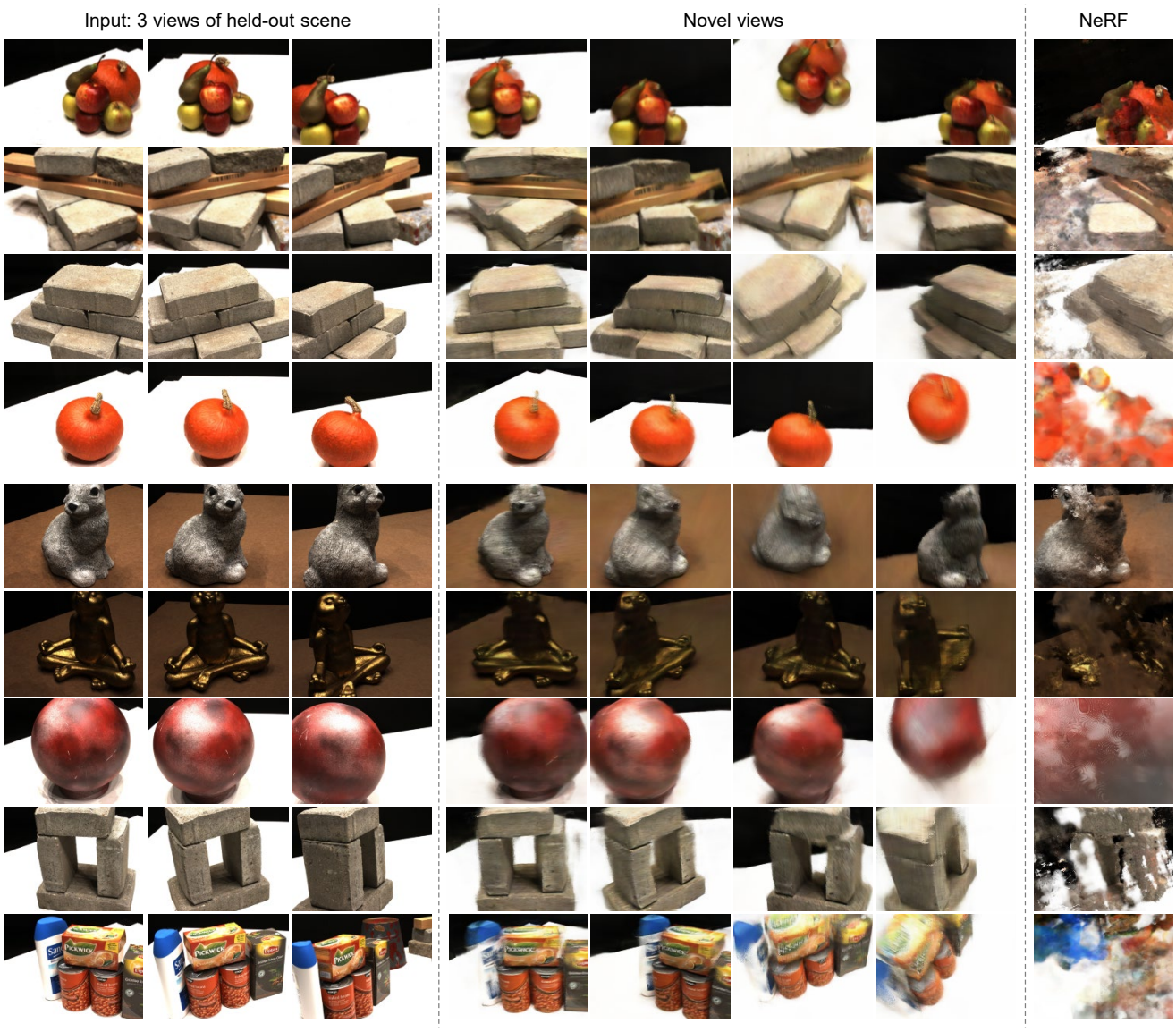


Figure 6: **Additional DTU results.** Views from the remaining 9 scenes are shown.

Evaluation protocol As discussed in the main paper, we evaluate on the test split from [2] as provided by DVR [7]. To ensure fairness, we sampled a random input view to encode for each object and use this view for all baselines as well.

Baselines For DVR [7], we use the pretrained 2D multiview-supervised model from the public GitHub and the provided rendering code (in `render.py`). For SoftRas [5], we similarly use the pretrained ShapeNet model from the public GitHub repo and obtain images using their renderer library.

Since SRN [9] did not originally evaluate in this setting, we train a model for this category-agnostic setting using the public code. We train for 1 million iterations and perform latent inversion for 260000 iterations, taking about 14 days on a Titan RTX in total.

B.2.3 Generalization to Novel Categories

We train our model for 680000 iterations across all instances of 3 categories: airplane, car, and chair. Rays are sampled from within a tight bounding box for the first 400000 iterations. This took about 5 days on a GTX 1080Ti.

Evaluation protocol Since there are more than 25000 objects in the 10 remaining categories, it would be computationally prohibitive to evaluate on all of them. For our pur-

Figure 7: **DTU split overlap.** The first and third scans (115, 119) are from the standard DTU training set from MVSNet, while the second and fourth (114, 118) are from the test set. In our split, highly similar scenes are either all placed in the same set or discarded.

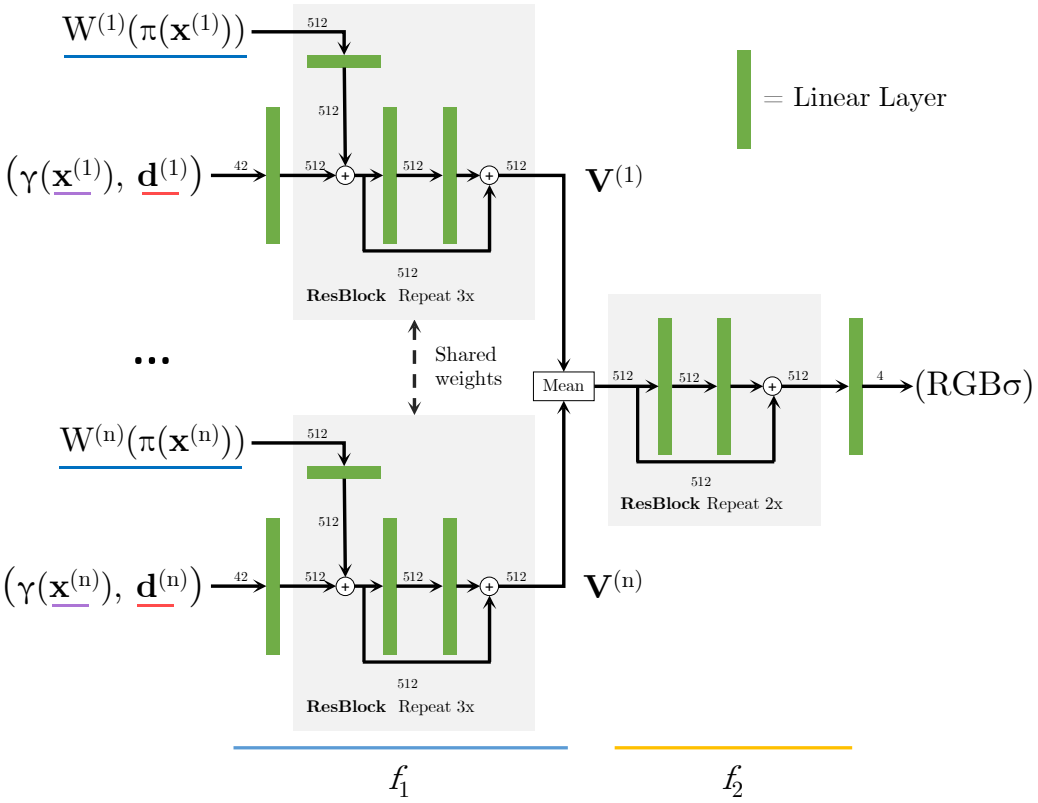


Figure 8: **Multi-view NeRF Network Architecture.** We use notation established in § 5.1.2 of the main paper, where γ denotes a positional encoding with 6 exponentially increasing frequencies. Each linear layer is followed by a ReLU activation. Note that in the single-view case, f_1 and f_2 can be considered a single ResNet $f = f_2 \circ f_1$.

poses, we sample 25% of the objects from each category for testing, using the remaining for validation. The protocol otherwise remains the same as in the category-agnostic model (§ B.2.2).

Baselines We train SRN and DVR as in § B.2.2. For DVR [7] we turned off the use of visual hull depth for sampling, since this information was not provided for all instances of the dataset shipped with DVR.

B.2.4 Two-object Scenes

We generate more complex synthetic scenes consisting of two ShapeNet chairs. We subdivide ShapeNet chairs into 2715 training instances and 1101 test instances. We gener-

ate scenes by randomly placing instances within each split around the origin, rotated randomly about each object's z-axis, and render 128×128 resolution images. Per instance, we render 20 training views sampled binned uniform on the hemisphere, and 50 testing views sampled on an Archimedean spiral, similar to the SRN protocol.

We compare with SRN [9] as our baseline on this task, using the publicly available code. We note that SRN performs prediction in a canonical object-centric coordinate system, and used this version for this task. We train a model for evaluation on this two-object dataset using one, two, and three input views. We first train the model for 1 million iterations. Then for each number of input views, we fix the set of input views per instance and perform latent inversion for 150,000 iterations.

B.2.5 Sim2Real on Real Car Images

We use car images from the Stanford Cars dataset [4]. PointRend [3] is applied to the images to obtain foreground masks and bounding boxes. After removing the background with this mask, the image is then translated and rescaled so that the center of the bounding box is at the center of the image and the shorter side of the bounding box is 1/4 of the image side length, 128. This normalization heuristic is motivated by the observation that the shorter side roughly corresponds to the height or width of the car, which is a more constant quantity than the length.

For evaluation, we set the camera pose to identity and use the same sampling strategy and bounds as at train time for the single-category cars model.

B.2.6 Real Images on DTU

A single model is trained on the 88 training scenes. We use exposure level 3 only. Note that while there are several views per scene with incorrect exposure throughout the DTU dataset, we did not remove them for training purposes. At each training step, a random color jitter augmentation is applied equally to all views of each object.

Dataset details While we solve a very different task from MVSNet [11] which predicts depth maps from short-baseline views and is 2.5D supervised, we considered using the MVSNet [11] DTU split to conform to standards for training on DTU. However, we found that the the split contained effective overlap across the train/val/test sets, making it a poor benchmark of cross-scene generalization, as shown in Fig. 7. For our purposes, we created a different split to avoid this issue: we use scans 8, 21, 30, 31, 34, 38, 40, 41, 45, 55, 63, 82, 103, 110, 114 for testing and all other scans except 1, 2, 7, 25, 26, 27, 29, 39, 51, 54, 56, 57, 58, 73, 83, 111, 112, 113, 115, 116, 117 for training.

We downsampled all DTU images to 400×300 and adjusted the world scale of all scans by a factor or $1/300$.

Evaluation protocol We separately evaluate using 1, 3, 6, 9 informative input views and calculate image metrics with the remaining views. Specifically, we selected views 25, 22, 28, 40, 44,48, 0, 8, 13 for input, taking a prefix of this list when less than 9 views are used. We exclude the views with bad exposure (they are 3, 4, 5, 6, 7, 16, 17, 18, 19, 20, 21, 36, 37, 38, 39) for testing.

Baselines We train a total of 60 NeRFs for comparison, one for each scene and number of input views, using the original NeRF TensorFlow code. Each NeRF is trained for 400000 iterations with ray batch size 128, which takes about 14 hours on an RTX Titan, to ensure convergence.

We found that NeRF did not converge in 5 cases when given 6 or 9 views, including in the case of smurf (scan 82) shown in the video. This is possibly due to exposure variation in the DTU dataset. For these scenes, we initialized the model to the trained weights for the same scene with 3 less views and train for about 200000 additional iterations to get reasonable results.

References

[1] Emilien Dupont, Miguel Angel Bautista, Alex Colburn, Aditya Sankar, Carlos Guestrin, Joshua Susskind, and Qi Shan. Equivariant neural rendering. In *Int. Conf. Mach. Learn.*, 2020. 6

[2] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 5, 8

[3] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. PointRend: Image segmentation as rendering. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020. 10

[4] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013. 10

[5] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Int. Conf. Comput. Vis.*, 2019. 1, 8

[6] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Eur. Conf. Comput. Vis.*, 2020. 1, 5

[7] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020. 1, 8, 9

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner,

Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 5

[9] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Adv. Neural Inform. Process. Syst.*, 2019. 1, 5, 8, 10

[10] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014. 5

[11] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. Mvsnet: Depth inference for unstructured multi-view stereo. In *Eur. Conf. Comput. Vis.*, 2018. 10

[12] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 5