In this supplement, we provide the following:

- Additional experimental details on datasets, model training, and model editing (Section A).
- Additional evaluations of our method and additional ablations. (Section B).
- Additional methods for shape editing. (Section C).
- A description of our user interface. (Section D).
- Additional visualizations of color edits with our approach (Section E).
- Additional visualizations of shape edits with our approach (Section F).
- Additional visualizations of color and shape swapping edits with our approach (Section G).
- A visualization of reconstructions with our conditional radiance field (Section H).
- Changelog (Section **??**)

A. Additional Experimental Details

Dataset rendering details. For the PhotoShape dataset [51], we use Blender [28] to render 40 views for each instance with a clean background. To obtain the clean background, we obtain the occupancy mask and set everywhere else to white. For the Aubry chairs dataset [5], we resize the images from 600×600 resolution to 400×400 resolution and take a 256×256 center crop. For the CARLA [17] dataset, we train our radiance field on exactly the same dataset as GRAF [63].

Conditional radiance field training details. As in Mildenhall *et al.* [45], we train two networks, a coarse network $\mathcal{F}_{\text{coarse}}$ to estimate the density along the ray, and a fine network $\mathcal{F}_{\text{fine}}$ that renders the rays at test time. We use stratified sampling to sample points for the coarse network and sample a hierarchical volume using the coarse network's density outputs. The rendered outputs of these networks for an input ray $\boldsymbol{r}$ are given by $\hat{C}_{\text{coarse}}(\boldsymbol{r})$ and $\hat{C}_{\text{fine}}(\boldsymbol{r})$, respectively. The networks are jointly trained with the shape and color codes to optimize a photometric loss. During each training iteration, we first sample an object instance $k \in \{1, \dots, K\}$ and obtain the corresponding shape code $\boldsymbol{z}_k^{(s)}$ and color code $\boldsymbol{z}_k^{(c)}$. Then, we sample a batch of training rays from the set of all rays R_k belonging to the instance k, and optimize both networks using a photometric loss, which is the sum of squared-Euclidean distances between the predicted colors and ground truth colors,

$$\mathcal{L}_{\text{train}} = \sum_{k=1}^{K} \Bigg[\sum_{\boldsymbol{r} \in R_k} ||\hat{C}_{\text{coarse}}(\boldsymbol{r}, \boldsymbol{z}_k^{(s)}, \boldsymbol{z}_k^{(c)}) - C(\boldsymbol{r})||_2^2 + ||\hat{C}_{\text{fine}}(\boldsymbol{r}, \boldsymbol{z}_k^{(s)}, \boldsymbol{z}_k^{(c)}) - C(\boldsymbol{r})||_2^2 \Bigg]. \quad (7)$$

When training all radiance field models, we optimize our parameters using Adam [32] with a learning rate of 10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. We train our models until convergence, which on average is around 1M iterations.

Conditional radiance field editing details. During editing, we keep the coarse network fixed and edit the fine network $\mathcal{F}_{\text{fine}}$ only. We increase the learning rate to 10^{-2}, and we optimize network components and codes for 100 iterations, keeping all other hyperparameters the same. To obtain training rays, we first randomly select an edited view of the instance, then randomly sample batches of rays from the subsampled set of foreground and background rays.

Conditional radiance field architecture details. Like the original NeRF paper [45], we use skip connections in our architecture: the shape code and embedded input points are fed into the fusion shape network as well as the very beginning of the network.

Furthermore, in our model architecture, we introduce a bottleneck that allows feature caching to be computationally feasible. The input to the color branch is an 8-dimensional vector, which we cache during color editing.

Last, when injecting a shape or color code to a layer, we run the code through a linear layer with ReLU nonlinearity and concatenate the output with the input to the layer.

GAN editing details. In our GAN experiments, we use the default StyleGAN2 [31] configuration on the PhotoShapes dataset and train for 300,000 iterations. For model rewriting [8], we optimize the 8-th layer for 2,000 iterations with a learning rate of 10^{-2}.

View direction dependence. On the CARLA dataset [17, 63], we find that having only one training view per instance can cause color inconsistency across rendered views. To address this, we regularize the view dependence of the radiance $\boldsymbol{c}$ with an additional self-consistency loss that encourages the model to predict for a point $\boldsymbol{x}$ similar radiance across viewing directions $\boldsymbol{d}$. This loss penalizes, for point $\boldsymbol{x}$ and viewing direction $\boldsymbol{d}$ in the radiance field, the squared difference between the sampled radiance $\boldsymbol{c}(\boldsymbol{x}, \boldsymbol{d})$ and the average radiance of the point $\boldsymbol{x}$ over all viewing directions. Specifically, given radiance field inputs $\boldsymbol{x}, \boldsymbol{d}$, we minimize

$$\mathbb{E}_{\boldsymbol{x} \sim p_{\boldsymbol{x}}; \boldsymbol{d} \sim p_{\boldsymbol{d}}} \left[\left\| \boldsymbol{c}(\boldsymbol{x}, \boldsymbol{d}) - \mathbb{E}_{\boldsymbol{d}' \sim p_{\boldsymbol{d}}}[\boldsymbol{c}(\boldsymbol{x}, \boldsymbol{d}')) \right\|^2 \right]$$

where $p_{\boldsymbol{x}}$ is the probability distribution over points $\boldsymbol{x}$ and $p_{\boldsymbol{d}}$ is the probability distribution over all viewing directions $\boldsymbol{d}$.

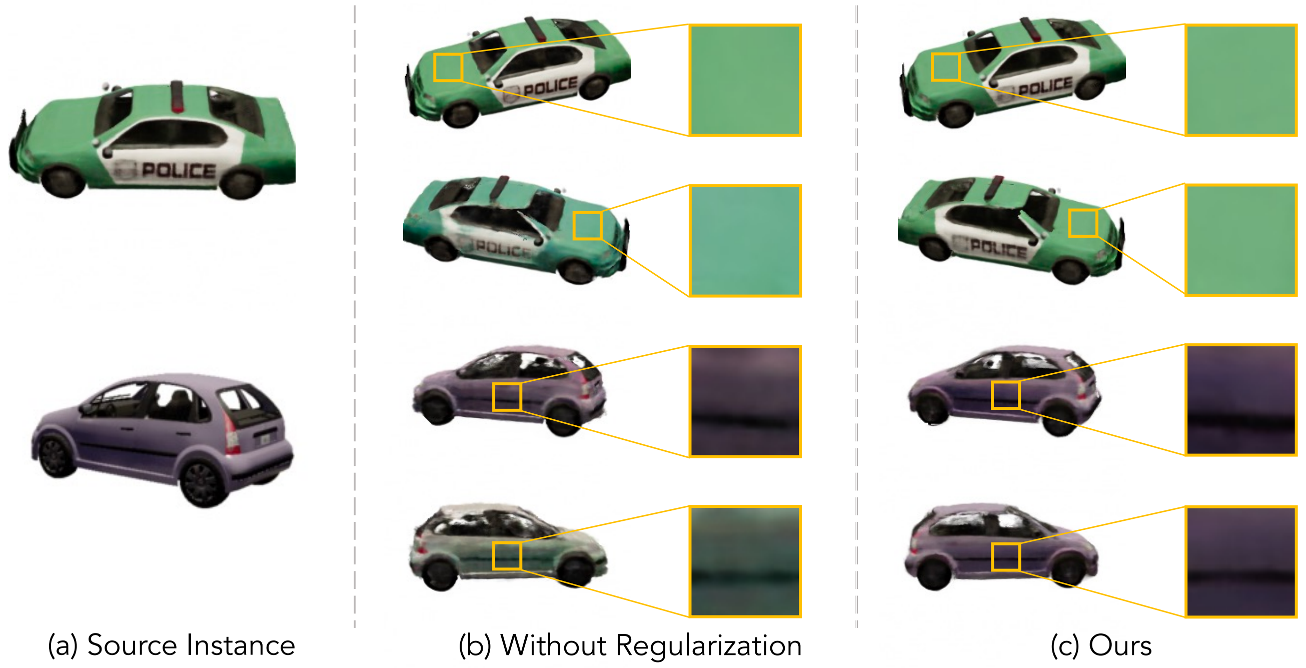

Figure 7: **CARLA [17] dataset radiance view dependence regularization.** We show synthesized views from an unregularized conditional radiance field and a regularized conditional radiance field trained on one view per instance. Notice how the regularized model is consistent in color across views while the unregularized model is not, hallucinating between green and blue (top) and purple and green (bottom).

During training, we approximate this loss by first sampling a training point $\boldsymbol{x}$ and viewing direction $\boldsymbol{d}$, then approximating the inner expectation $\mathbb{E}_{\boldsymbol{d}' \sim p_{\boldsymbol{d}}}[\boldsymbol{c}(\boldsymbol{x}, \boldsymbol{d}')]$ by sampling K viewing directions $\boldsymbol{d}_i \sim p_{\boldsymbol{d}}$, and taking

$$\mathbb{E}_{\boldsymbol{d}' \sim p_{\boldsymbol{d}}}[\boldsymbol{c}(\boldsymbol{x}, \boldsymbol{d}')] \approx \frac{1}{K}\sum_{i=1}^{K} \boldsymbol{c}(\boldsymbol{x}, \boldsymbol{d}_i).$$

In our experiments, we use $K = 64$. We visualize results with and without this regularization in Figure 7.

B. Additional Evaluations

SSIM metric. We report additional evaluation on model ablation, color editing, and shape editing results using SSIM [71]. Quantitative results can be found in this supplement's Table 4 (model ablation), Table 5 (color edits), and Table 6 (shape edits).

Subsampling user constraints. During editing, we do not train on the whole foreground and background regions provided by the user, which can potentially decrease the quality of our edits. This is because training on fewer rays can cause the edit to propagate onto unwanted areas. For example, regions which the user specify as background, but are not in the set of sampled rays, can potentially be changed. However, we find that upon adding this optimization, the average PSNR over the three color edits decreases to 34.49.

Additional GAN editing baselines. We compare our editing method against a naive generator fine-tuning method [8]. The method is identical to the model rewriting method, except instead of conducting a low-rank update of the weights of a particular layer, we freely optimize all the weights of the generator. This optimization is done over 10,000 steps with a learning rate of 10^{-3}. We report our results in Table 5.

C. Additional Shape Editing Methods

Shape removal. For shape removal method described in the main paper, we assume that there is nothing behind an object part that a user scribbles over, allowing us to replace the object part with a white background. However, in practice, a user may wish to remove an object part that is in front of another one. To handle such occlusion, we propose a separate procedure: for each ray in the foreground mask, we zero out the first mode of density along the ray. We define the first mode of density to start at the first point with nonzero-density up to the first subsequent point with zero density. We find that this procedure is effective but can be slow and may leave artifacts of incomplete removal.

Shape addition. For shape addition, our method for recon-

	PhotoShapes [51]			Aubry *et al.* [5]		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
1) Single NeRF [45]	17.81	0.836	0.435	14.26	0.814	0.390
2) + Learned Latent Codes	36.50	0.979	0.029	20.93	0.892	0.164
3) + Sep. Shape/Color Codes	36.88	0.980	0.028	21.54	0.898	0.144
4) + Shar./Inst. Net (Ours)	**37.67**	**0.982**	**0.022**	**21.78**	**0.900**	**0.141**
5) NeRF Separate Instances	37.31	0.972	0.035	24.15	0.963	0.041

Table 4: **Conditional radiance field ablation study.** We evaluate our model and several ablations on novel view synthesis. Notice how separating the shape and color codes and using the shared/instance network improves the view synthesis quality.

	PSNR ↑	SSIM ↑	LPIPS ↓
GAN-Finetuning	19.64	0.704	0.255
Model Rewriting [8]	18.42	0.622	0.325
Finetuning Single-Instance NeRF	29.53	0.955	0.068
Only Finetune Color Code	26.29	0.968	0.090
Finetuning All Weights	31.00	0.957	0.050
Our Method	**35.25**	**0.977**	**0.027**

Table 5: **Color editing quantitative results.** We evaluate color editing of a source object instance to match a target instance. Our method outperforms the baselines on all criteria.

	PSNR ↑	SSIM ↑	LPIPS ↓	Time (s) ↓
Only Finetune Shape Code	22.08	0.931	0.119	36.9
Only Finetune $\mathcal{F}_{\text{dens}}$	21.84	0.921	0.118	**27.2**
Finetuning All Weights	20.31	0.910	0.117	66.4
Our Method	**24.57**	**0.944**	**0.081**	37.4

Table 6: **Shape editing quantitative results.** Notice how our hybrid network update approach achieves high visual edit quality while balancing computational cost.

structing a composite image leads to effective but slow edits. We propose an additional density-based loss which is faster but less effective in executing the edit. The method for obtaining the editing example is the same, but we now optimize a loss that encourages the density values in the modified regions of the composite view to match with the density values of the regions copied from.

Specifically, let $y_f = \{(\boldsymbol{r}, \sigma_f)\}$ be the set of rays and densities in the foreground mask and $y_b = \{(\boldsymbol{r}, \sigma_b)\}$ be the set of rays and densities in the background mask. Here, σ_f are density values of the rays copied from the new object instance, and σ_b represent density values of the rays from the original instance. Furthermore, let $\sigma_{\boldsymbol{r}}$ be the densities predicted by our model for ray $\boldsymbol{r}$. Again, densities are normalized to sum to one.

We optimize a cross-entropy loss:

$$\mathcal{L}_{\text{dens}} = \sum_{(\boldsymbol{r}, \sigma_f) \in y_f} -\sigma_f^T \log \sigma_{\boldsymbol{r}} + \sum_{(\boldsymbol{r}, \sigma_b) \in y_b} -\sigma_b^T \log \sigma_{\boldsymbol{r}}, \quad (8)$$

which encourages the predicted densities to match the target densities for the edited regions and be unchanged for the unedited regions.

D. User Interface

For our user interface, the user first picks an object instance they would like to edit. Our UI then displays several rendered views of that instance, and the user picks one view to edit. The user can then edit the selected view on an editing panel.

We provide four types of user edits: color edits, shape removal, shape addition, and color/shape transfer. Next, we describe the user interactions for each type of edit.

Color edits. The user chooses the target color from a color palette. Then, the user specifies a foreground mask over the view by clicking the `edit color` button, selecting a brush color, and scribbling over parts of the object. Last, the user specifies the background mask by clicking the `BG` brush and scribbling over where they would like to keep the part unchanged.

Shape removal. The user clicks the `remove shape` button and scribbles over parts of the image they would like to remove.

Shape addition. The user clicks the `add shape` button and several instances to copy shape from will pop up. The user specifies a target instance they would like to copy from, and a view of that instance is shown. Then, the user scribbles over the object part they would like to copy, and clicks on the location of the source instance where they would like to paste.

Shape/Color transfer. The user clicks either the `transfer color` button or the `transfer shape` button and several instances to transfer color/shape from will pop up. Then, the user clicks a desired target instance to transfer color/shape information.

Once the user editing is done, the user will click the `execute` button to execute the desired edit. Our algorithm will then finetune the latent variables and network weights and update the renderings of the edited object. Please see our video demo for more details.

E. Additional Color Edits

Quantitative color editing evaluation. In this section, we provide the visualizations of all three color edits used for evaluation in the main paper. Visually, we again see that editing a single-instance NeRF leads to visual artifacts and visual inconsistencies across views. Similarly, GAN-based methods are unable to learn an edit that generalizes across

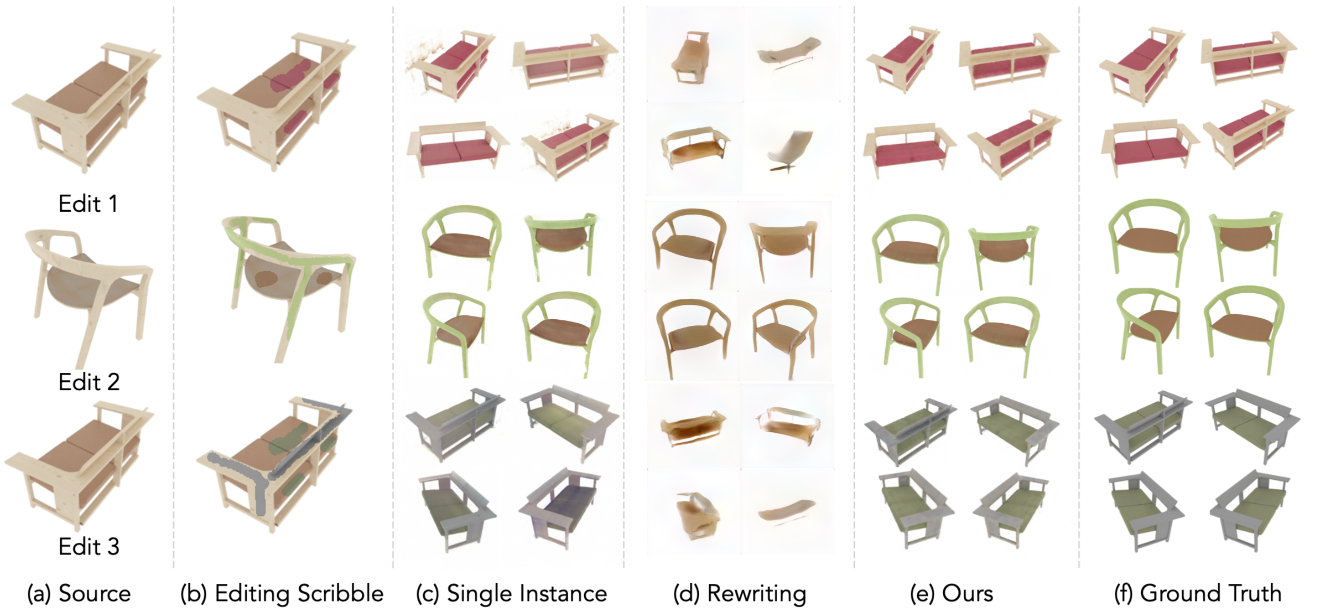

Figure 8: **Color editing qualitative results.** We visualize color editing results where the goal is to match a source instance's colors to a target. Our method accurately captures the colors of the target instance given scribbles over one view. Notice how (d) Rewriting a GAN [8] fails to propagate the edit to unseen views and results in unrealistic generated outputs. Moreover, editing a single-instance NeRF causes visual floating artifacts (Edit 1) and non-transferring colors (Edit 3).

	Edit 1			Edit 2			Edit 3		
	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
GAN-Finetuning	21.42	0.744	0.218	20.45	0.730	0.264	18.91	0.686	0.251
Model Rewriting	20.44	0.663	0.301	19.11	0.659	0.322	18.03	0.612	0.315
Finetuning Single-Instance NeRF	28.22	0.933	0.125	28.32	0.956	0.057	29.88	0.964	0.051
Only Finetune Color Code	28.15	0.975	0.054	27.77	0.977	0.097	22.95	0.953	0.120
Finetuning All Weights	33.36	0.970	0.029	32.47	0.967	0.036	27.17	0.936	0.086
Our Method	**35.32**	**0.978**	**0.024**	**35.72**	**0.979**	**0.027**	**34.72**	**0.975**	**0.031**

Table 7: **Color editing quantitative results.** We evaluate color editing of a source object instance to match a target instance. Please refer to Figure 8 (this supplemental) for a visualization of each of the edits. Notice that our method outperforms the baselines for all color edits on all criteria.

views, likely due to their lack of a 3D representation. We visualize the results of the three edits in Figure 8 and quantify them in Table 7.

In Figure 8, the first two rows visualize Edits 1 and 2 discussed in the main paper, and are identical to the visualizations in the main paper's Figure 3. The last row of Figure 8 visualizes Edit 3, which changes the seat of a chair from brown to green, then the chair back from beige to grey.

In Table 7, we quantify the quality of each of the four edits. The first two main columns correspond to the Edits 1 and 2 discussed in the main paper, while the last two main columns correspond to Edits 3 discussed in Section E.

Editing an unseen instance. The source instances we edit in the paper are in the training set. Here, we additionally find that our method succeeds at editing an unseen source instance in the *test set*, which we show in Figure 9. We first finetune our conditional radiance field on a single view of a test set instance using the optimization method for finetuning on real images, and use our editing method to change the color of the instance.

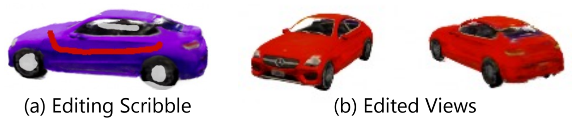

Figure 9: **Unseen instance editing.** Our method finetunes and edits an unseen *test set* instance.

Color editing additional ablations. In this section, we evaluate our color editing method and several ablations on 10 edits. We report mean and standard errors in Table 8. Notice how our method reliably outperforms all ablations.

Single-instance NeRF editing. We also provide an additional comparison of our method against editing a single-instance NeRF. Here, we change the color of a seat from brown to bright red. Again, we observe that the single-instance NeRF does not learn an edit that generalizes; the model frequently creates red artifacts in chair's background. In contrast, our model can still learn an edit that successfully propagates to the seat but not to other regions of the scene. We visualize these results in Figure 10.

Additional color editing results. We visualize additional color edits on the Aubry chairs [5] and the CARLA cars [17, 63] datasets in Figure 11.

F. Additional Shape Edits

Quantitative shape editing evaluation. In this section, we visualize all three shape edits used for evaluation in the main paper. We visualize the results of the three edits in Figure 12 and quantify them in Table 9. Visually, we see that consistent with the main paper, both finetuning the shape code and the shape branch are not enough to change the instance, but finetuning the whole network causes unwanted changes in the instance.

Shape editing additional ablations. In this section, we evaluate our shape editing method and several ablations on 10 edits. We report mean and standard errors in Table 10. Notice how our method reliably outperforms all ablations.

Single-instance NeRF editing. We compare our method against editing a single-instance NeRF [45]. We find that similar to the case with color edits, single-instance NeRFs are unable to learn an edit that generalizes to unseen views, likely due to a lack of a category-level prior. We visualize these results on the PhotoShapes dataset [51] in Figure 13.

Additional shape editing results. We visualize additional shape edits on the PhotoShapes [51] and the CARLA cars datasets [17, 63] in Figure 14.

G. Additional Color/Shape Swapping Edits

We visualize additional shape and color swapping results on the PhotoShapes dataset [51] in Figure 15 (this supplement). Notice again how changing the color code keeps the shape of the instance unchanged, and how changing the shape code keeps the color of the instance unchanged.

H. View Reconstruction

View consistency results. For each of our three datasets, we visualize synthesized views for a fixed instance and observe that the rendered views are all consistent in shape and color. We visualize these results in Figure 16. Notice how in the CARLA dataset [17, 63], despite training on only one image per car instance, the model is able to infer the occluded regions of the car.

Additional reconstruction results. We visualize reconstructed views and depth maps across several instances of the PhotoShapes dataset [51] using our conditional radiance field. For each instance, we render four unseen viewpoints from our model and visually compare them against the ground truth views. We find that our method is able to almost perfectly reconstruct each instance, as well as learn convincing depth estimates of each instance. We visualize reconstructions and depth maps for unseen views in Figures 17-21.

	PSNR ↑	SSIM ↑	LPIPS ↓
Finetuning Single-Instance NeRF	29.84 ± 2.15	0.959 ± 0.008	0.095 ± 0.008
Only Finetune Color Code	31.19 ± 1.55	0.968 ± 0.004	0.090 ± 0.005
Finetuning All Weights	26.29 ± 0.82	0.965 ± 0.005	0.042 ± 0.006
Our Method	**35.15** ± 0.95	**0.976** ± 0.002	**0.030** ± 0.002

Table 8: **Color editing quantitative results.** We evaluate color editing of a source object instance to match a target instance. We report means and standard errors over 10 edits. Our method consistently outperforms the baselines on all criteria.

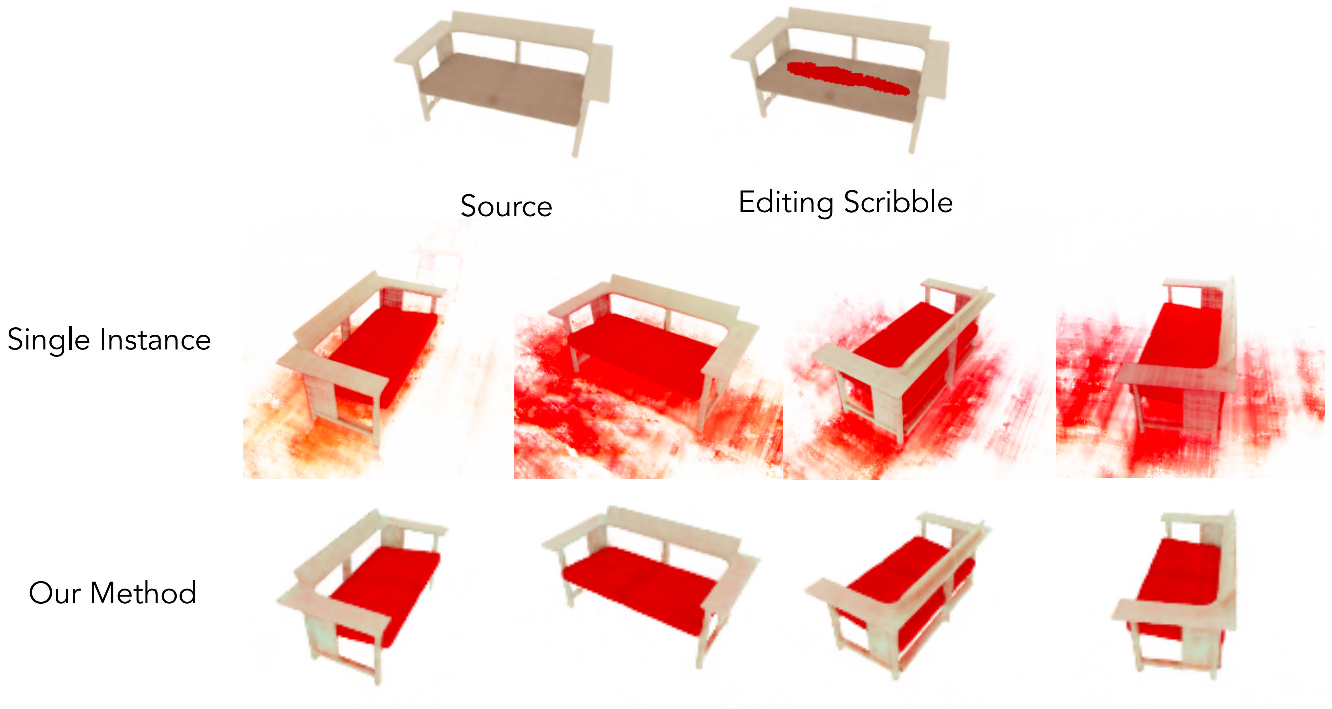

Figure 10: **Single-instance vs. conditional radiance field editing.** We visualize the edit of changing the color of a seat from brown to red. We find that edits on single-instance NeRFs propagate to outside the chair and cause artifacts in the background, whereas our model successfully propagates the edit across only the seat of the chair.

	Edit 1			Edit 2			Edit 3		
	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
Only Finetune Shape Code	23.52	0.947	0.100	20.18	0.919	0.138	22.52	0.927	0.118
Only Shape Branch	24.59	0.947	0.090	17.96	0.887	0.160	22.94	0.929	0.104
Finetuning All Weights	21.31	0.923	0.100	19.77	0.903	0.128	19.84	0.903	0.123
Our Method	**25.68**	**0.958**	**0.069**	**22.97**	**0.933**	**0.091**	**25.04**	**0.943**	**0.083**

Table 9: **Shape editing quantitative results.** We evaluate shape editing of a source object instance to match a target instance. Please refer to Figure 12 for a visualization of each of the edits. Notice that our method outperforms the baselines for all color edits on all criteria.

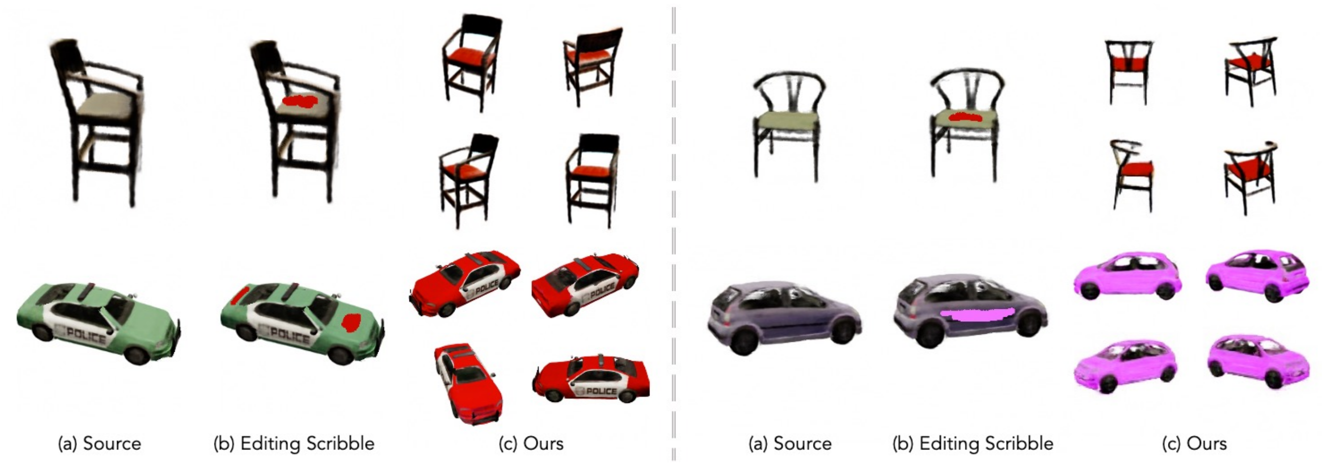

Figure 11: **Additional color editing qualitative results.** Our method successfully colors the seats of two Aubry *et al.* chairs [5] to red, and changes the car body colors to red and pink.

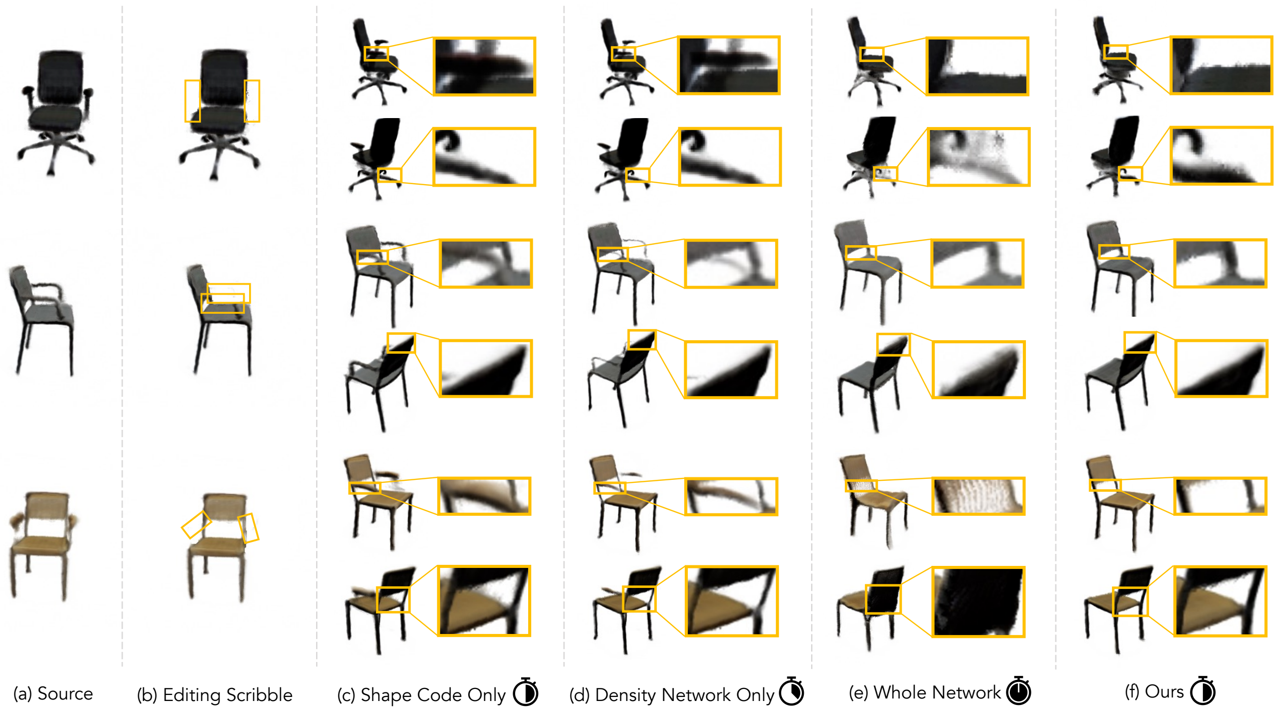

Figure 12: **Shape editing quantitative results.** Notice how only optimizing the shape code or branch are unable to fit both edits. Optimizing the whole network is slow and causes unwanted changes in the instance.

	PSNR ↑	SSIM ↑	LPIPS ↓
Only Finetune Shape Code	21.22 ± 0.62	0.922 ± 0.007	0.125 ± 0.006
Only Finetune $\mathcal{F}_{\text{dens}}$	21.84 ± 0.83	0.921 ± 0.007	0.118 ± 0.009
Finetuning All Weights	19.53 ± 0.32	0.903 ± 0.003	0.129 ± 0.005
Our Method	$\mathbf{23.69} \pm 0.76$	$\mathbf{0.934} \pm 0.007$	$\mathbf{0.084} \pm 0.007$

Table 10: **Shape editing quantitative results.** We evaluate shape editing of a source object instance to match a target instance. We report means and standard errors over 10 edits. Our method consistently outperforms the baselines on all criteria.

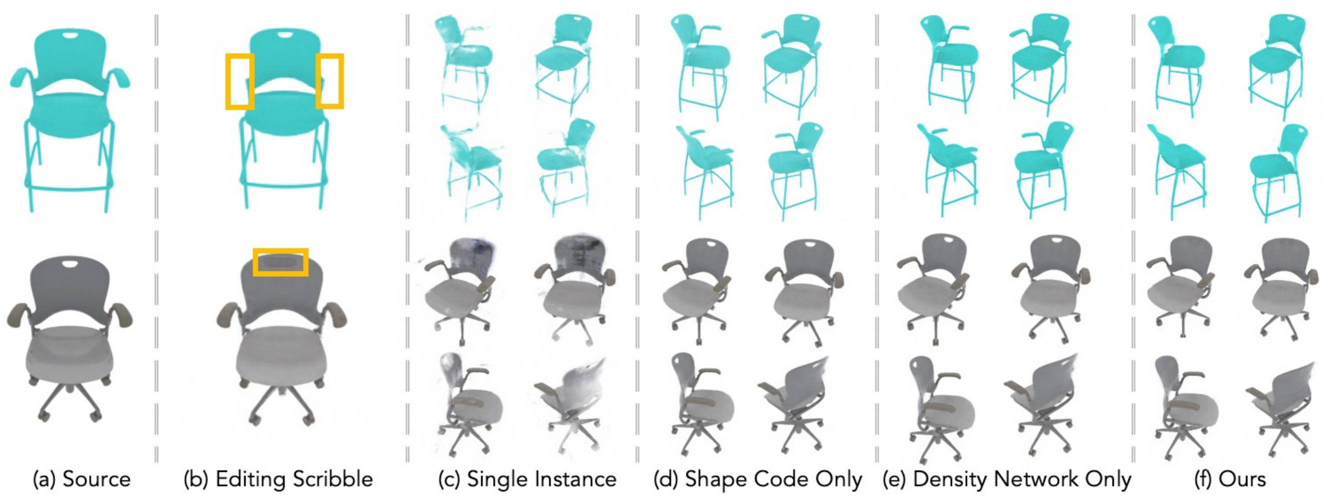

Figure 13: **Shape editing qualitative results.** Our method successfully removes the arms and fills in the hole of a chair. Notice how only optimizing the shape code or branch are unable to fit both edits. Furthermore, editing a single instance NeRF [45] causes unwanted artifacts.

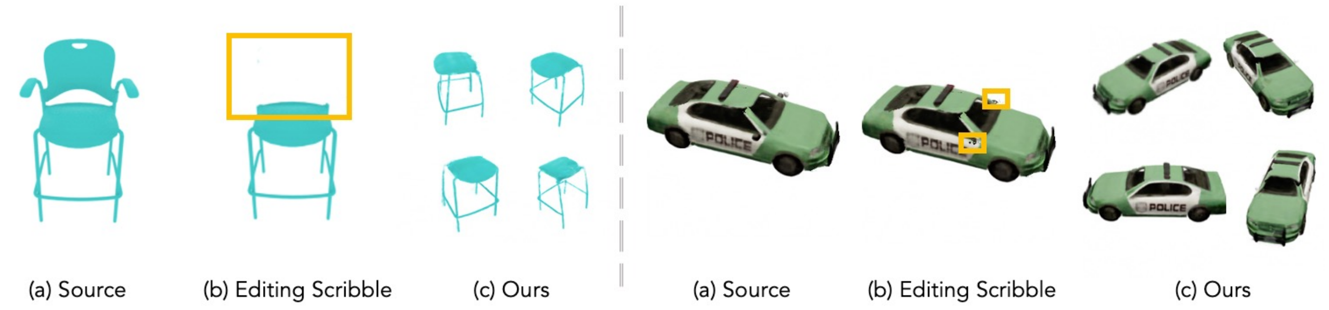

Figure 14: **Shape editing qualitative results.** Our method successfully removes the back and arms of a chair and removes the car mirrors.

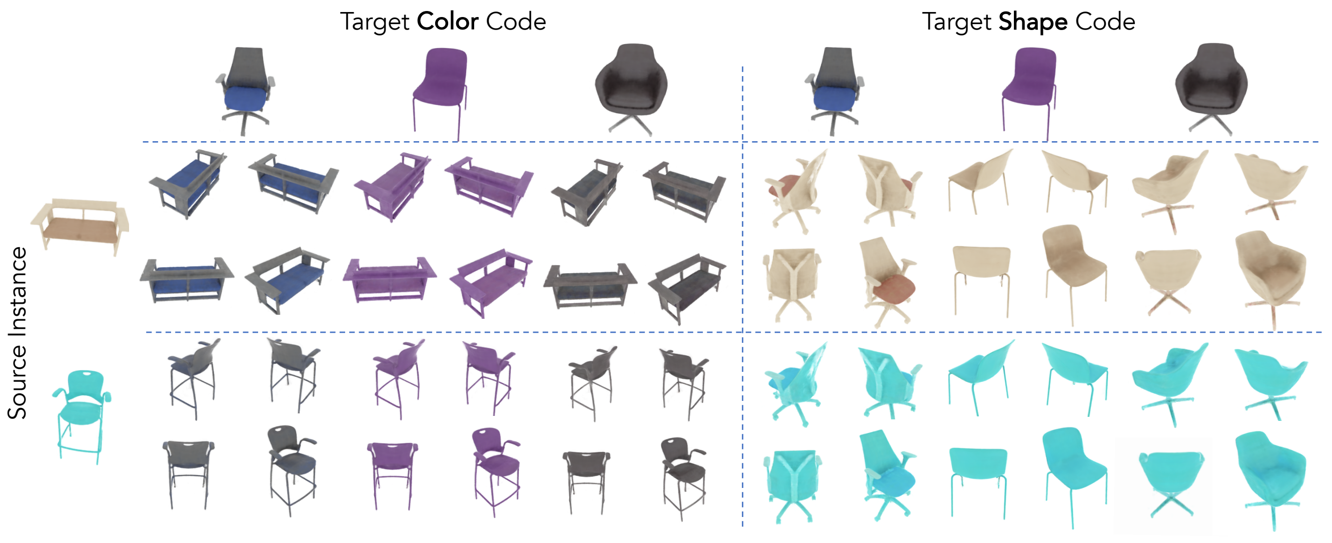

Figure 15: **Shape and color transfer results.** Our model transfers the shape and color from target instances to a given source instance. Notice that when a source's color code is swapped with a target's, the shape remains unchanged, and vice versa.

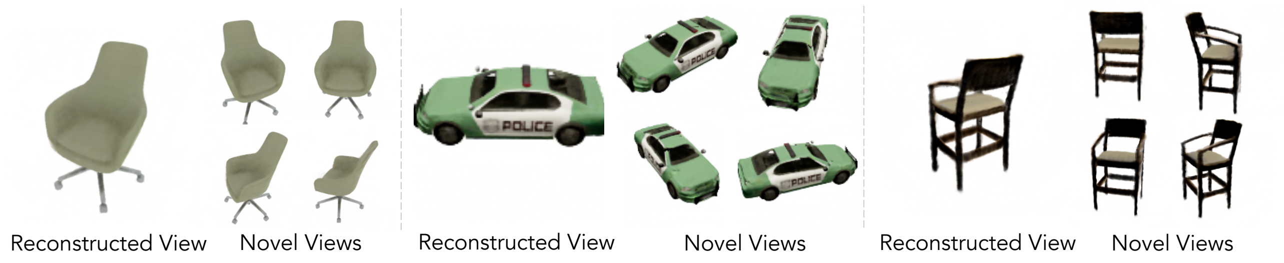

Figure 16: **View reconstruction results.** Our method renders realistic and consistent views across several instances using a single model.

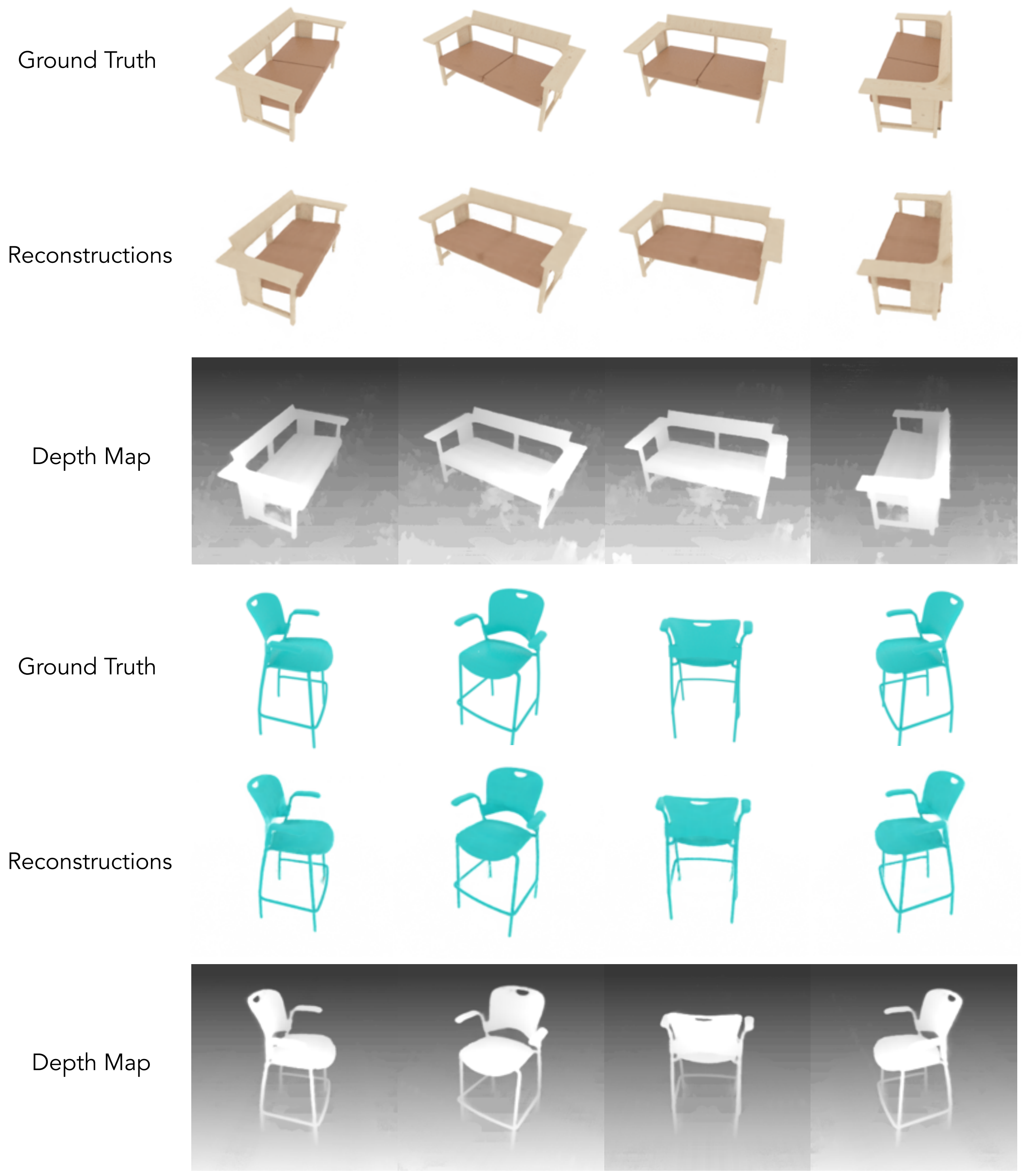

Figure 17: **View reconstruction and depth prediction.** We visualize the rendered views and predicted depth maps of our model on four unseen viewpoints. Notice our model is able to almost perfectly reconstruct the ground truth views.

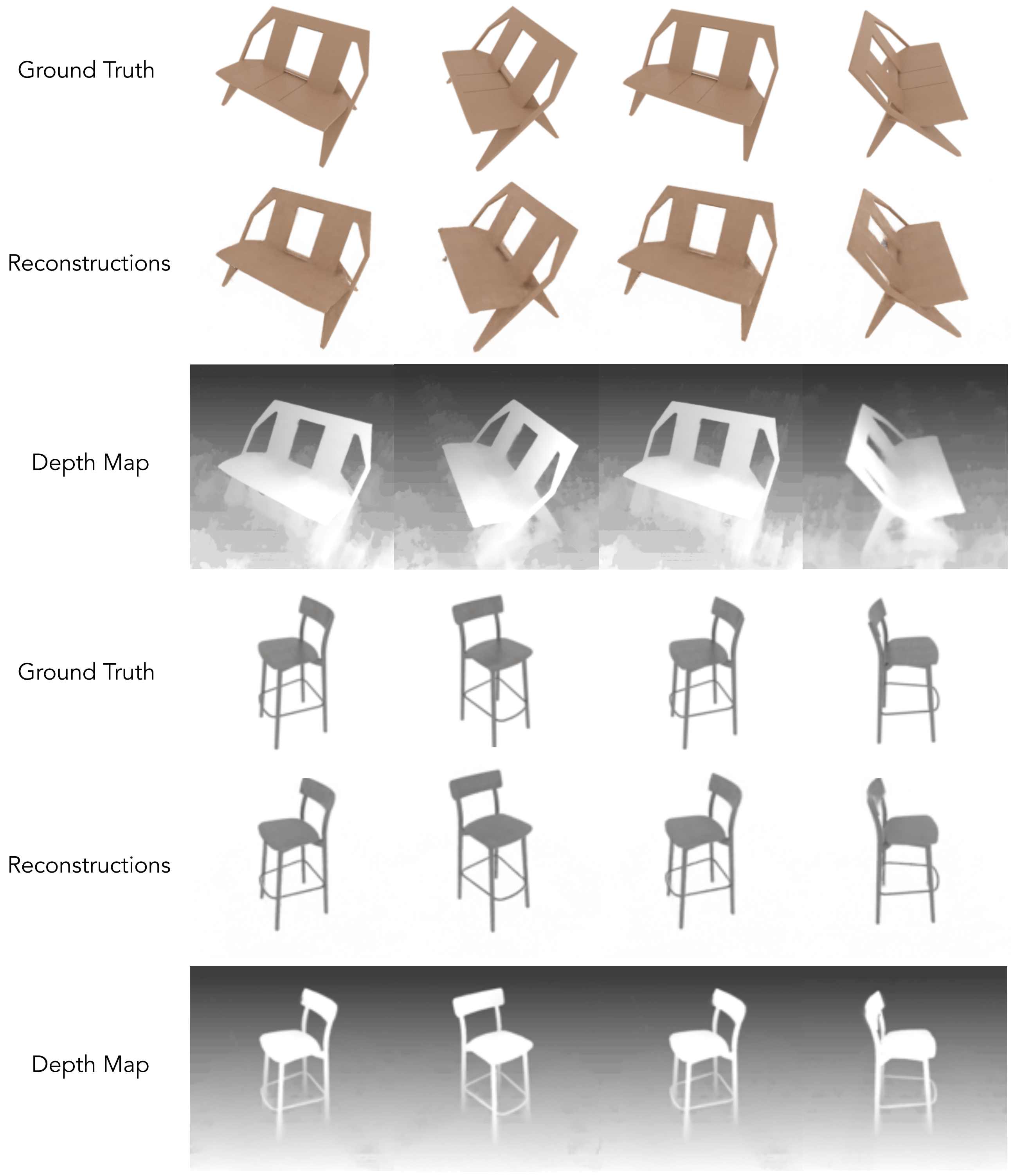

Figure 18: **View reconstruction and depth prediction.** We visualize the rendered views and predicted depth maps of our model on four unseen viewpoints. Notice our model is able to almost perfectly reconstruct the ground truth views.

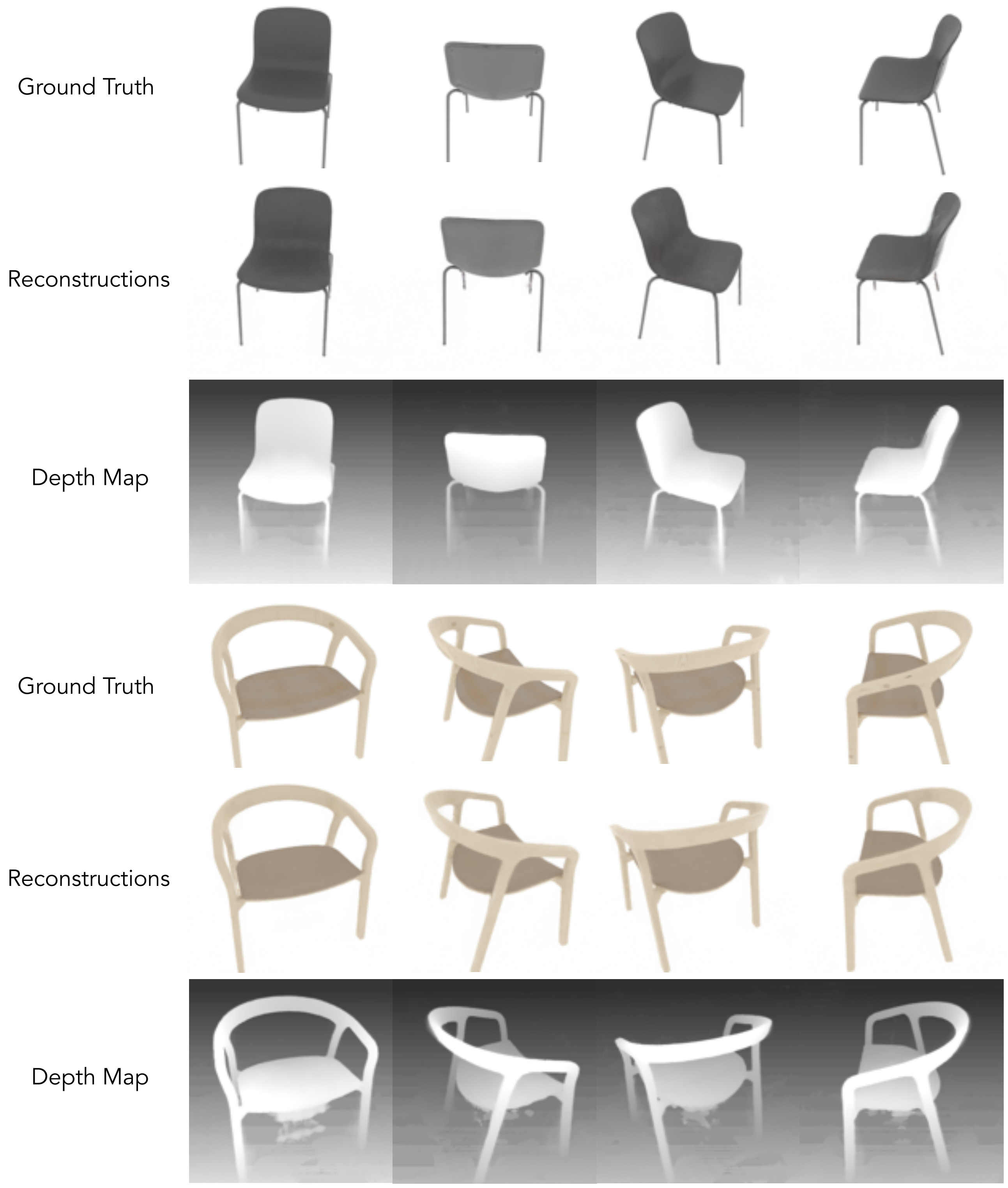

Figure 19: **View reconstruction and depth prediction.** We visualize the rendered views and predicted depth maps of our model on four unseen viewpoints. Notice our model is able to almost perfectly reconstruct the ground truth views.

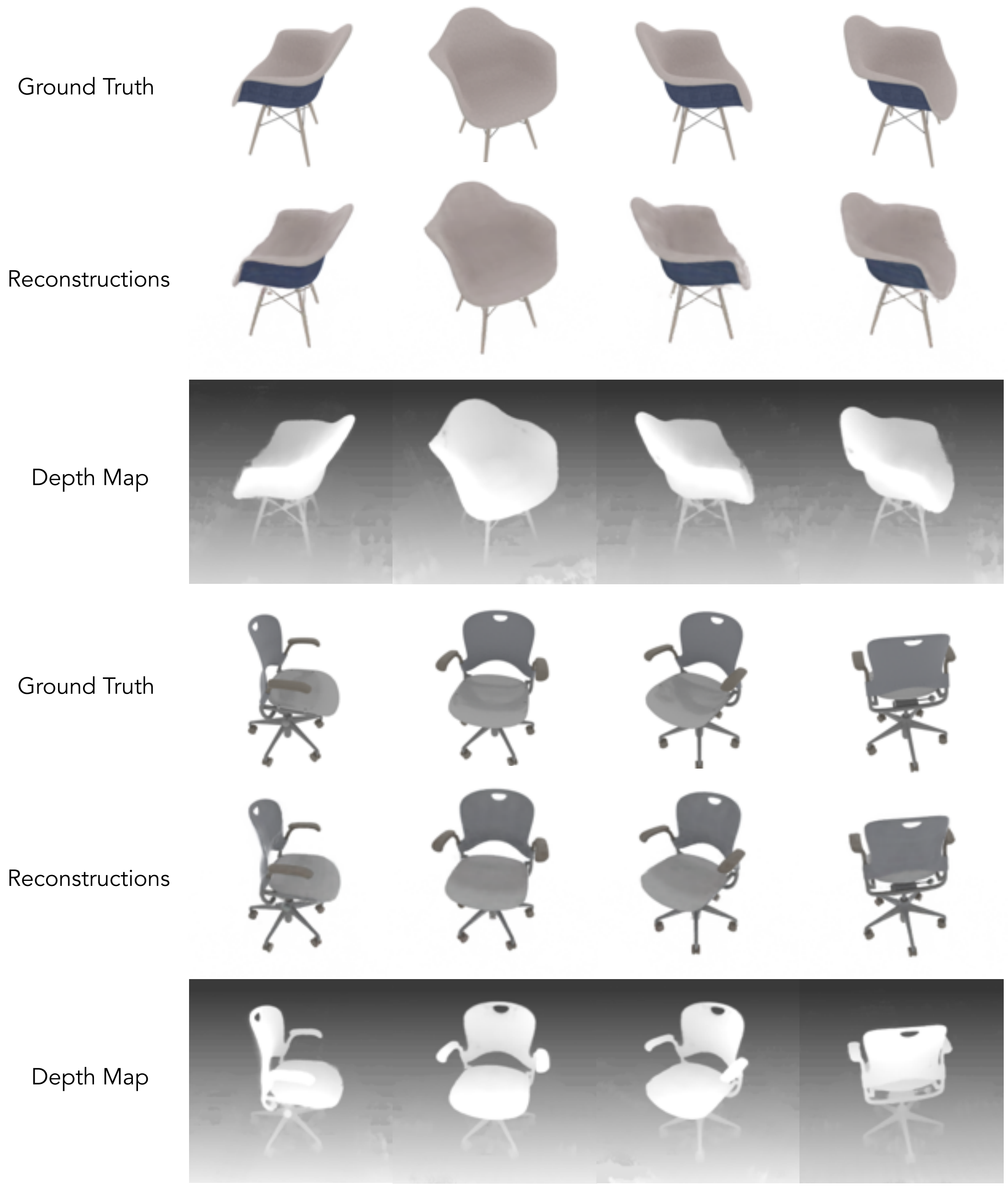

Figure 20: **View reconstruction and depth prediction.** We visualize the rendered views and predicted depth maps of our model on four unseen viewpoints. Notice our model is able to almost perfectly reconstruct the ground truth views.

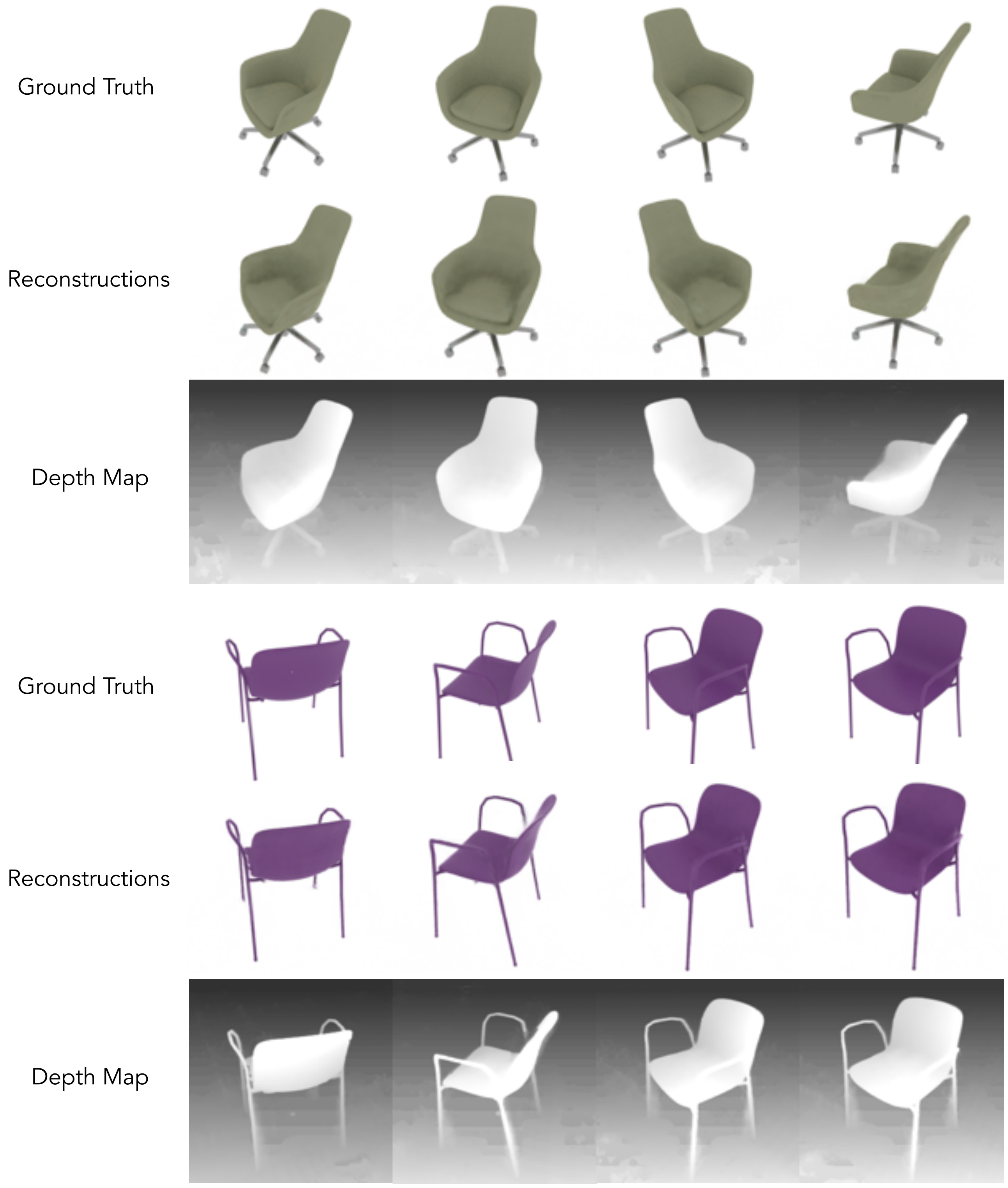

Figure 21: **View reconstruction and depth prediction.** We visualize the rendered views and predicted depth maps of our model on four unseen viewpoints. Notice our model is able to almost perfectly reconstruct the ground truth views.