Supplementary Material
DiffusioNeRF: Regularizing Neural Radiance Fields with Denoising Diffusion Models

Jamie Wynn   Daniyar Turmukhambetov
Niantic
www.github.com/nianticlabs/diffusionerf

1. Details of the Diffusion Model

The diffusion model that we use to regularise our NeRFs is based on the architecture described in [2], as implemented in the denoising-diffusion-pytorch [9] repository. This is a UNet-based architecture with added self-attention layers. We use 64, 128, 256 and 512 feature channels in the four successive downsampling blocks of the UNet, and the reverse in the upsampling blocks; our network is constructed with:

```
import denoising_diffusion_pytorch
model = denoising_diffusion_pytorch.Unet(
    dim=64,
    dim_mults=(1, 2, 4, 8),
    channels=4
)
```

When converting depths to inverse depths for input to the diffusion model – both when training the model and when subsequently using it to train NeRFs – we clip depths to a minimum of 20 cm to keep the range of possible inverse depths finite. We then linearly transform that range of possible inverse depths so that it runs from -1 to 1, along with the RGB channels which we also transform to the range [-1, 1]. The concatenation of these normalised inverse depth and RGB channels forms the $4 \times 48 \times 48$ input to the diffusion model.

2. Diffusion Gradient Normalization Scheme

In practice we find that directly using the diffusion model's predicted noise as the negative gradient of a loss tends to produce a somewhat brittle system, which overweights the diffusion model in some situations and underweights it in others. Therefore we use a normalisation scheme to make the system more robust across different scenes.

The diffusion model operates in disparity-space, and we invert the rendered depth of the RGBD patches to produce a valid input to the model. For a given pixel of the RGBD image at inverse-depth z, the model's predicted noise gives us a gradient $\partial \mathcal{L}_{\text{diff}} / \partial z$, and backpropagating this gradient results in:

$$\frac{\partial \mathcal{L}_{\text{diff}}}{\partial d} = -\frac{1}{d^2} \frac{\partial \mathcal{L}_{\text{diff}}}{\partial z} \tag{1}$$

Where d is depth and $z = 1/d$. We note that this means that the gradients from the diffusion model get amplified in regions closer to the camera due to the d^2 in the denominator. To counteract this, we scale the model's gradients by the disparity, which suppresses this effect. Empirically we find this produces better results, and that not doing so tends to lead to overly aggressive regularisation near to the camera and overly weak regularisation far from it.

We also find that we obtain best results when we normalise the gradients w.r.t. the RGB channels in the patch:

$$\frac{\partial \mathcal{L}'_{\text{diff}}}{\partial \mathbf{x}_{\text{RGB}}} = \frac{\partial \mathcal{L}_{\text{diff}}}{\partial \mathbf{x}_{\text{RGB}}} \Bigg/ \left\| \frac{\partial \mathcal{L}_{\text{diff}}}{\partial \mathbf{x}_{\text{RGB}}} \right\|_2 . \tag{2}$$

This helps us control the size of the updates coming from the diffusion model relative to those from the training images – we never want the former term to overpower the latter. Again, we find empirically that this produces better results. Not doing this results in occasional visual artifacts where the diffusion model outputs a very large gradient w.r.t. the RGB channels of the patch.

When backpropagating the gradients from the diffusion model we use different weights for the gradient of the rendered patch with respect to depth and with respect to color. For LLFF we use weights of 4×10^{-7} for depth gradients, and 3×10^{-6} for RGB gradients; for DTU we use weights of 4×10^{-6} for depth and 3×10^{-5} for RGB.

3. Camera Pose Sampling

When rendering patches to pass into the diffusion model we generate camera poses that are from roughly the same perspective as the training poses.

For LLFF we take a training pose and randomly perturb its position. In particular we perturb each element of its position with uniformly distributed noise from $U(-0.2, 0.2)$ and we perturb its orientation with a random rotation through an angle between 0 and $\pi/5$ radians.

Because the DTU dataset has images with much narrower field of view, the above scheme tends to produce images in which the object of interest is not visible, unless the noise is turned down to extremely low levels. So for experiments on DTU we use a different approach in which we randomly select a consecutive pair of training camera poses and interpolate between them. The interpolation is done by linearly interpolating the poses' positions and slerping their orientations, using a random interpolation weight so that the final pose can be anywhere in between the first and second input pose. Because of the object-centric nature of this dataset, this tends to produce a pose that is also looking at the object of interest.

With both LLFF and DTU, 25% of the time we use a training pose for patch rendering, and sample the RGB component of the RGBD patch directly from the training image rather than by rendering it. This provides information flow directly from the training images to the diffusion model, rather than indirectly via the reconstruction. We find that this tends to be helpful in the early stages of the NeRF fitting process in which rendered RGBs are not yet accurate.

4. Distance Scaling

Our diffusion model was trained with metric depths, and so when using the model to regularize NeRFs, the camera poses should be roughly metric. For LLFF scenes we judged an approximate distance by inspection of the SfM point clouds.

In the LLFF dataset we also scale up all scenes so that their bounds are at least 7.5 metres in size. This is so that the distribution of depths becomes closer to that of the room-scale Hypersim data on which the model was trained. This affects the close-up object-centric scenes in LLFF – the 'flower' and 'orchids' scenes.

5. A note on LPIPS metrics

An earlier version of this paper used the Alex version of LPIPS rather than the VGG version throughout. However, our baselines in Table 1 of the main paper use the VGG version, making the comparison unfair. After this oversight was pointed out to us, we recomputed the LPIPS column in Table 1 using the VGG version (and the 'Average' column, which changes as a result). As a result, this paper (and its supplementary) use LPIPS-Alex everywhere *except* for Table 1, where LPIPS-VGG is used instead.

6. Visualizations of LLFF results

We show additional results on test views of the LLFF dataset with 3, 6 and 9 training poses in Figures 2, 3, and 4, respectively.

7. Quantitative Evaluation of LLFF Reconstruction

The LLFF dataset was compiled by taking photographs of a scene and recovering camera poses by running Colmap [6]. Pseudo ground-truth depth maps can also be computed with Colmap [7].

One could assume that this data would allow direct evaluation of reconstruction quality of our method. However, we found that evaluating NeRF depth maps using the MVS depth maps is problematic. First, Colmap estimates depth data around distinct and textured regions and provides very little geometry data around uniformly textured areas such as many walls, floors, ceilings, etc. These areas are exactly where our DDM improves the geometry estimation in NeRFs. Secondly, Colmap depth estimation produces noisy depths on reflective and transparent surfaces such as glass walls and shiny tables. Discarding depths on such surfaces by varying Colmap depth estimation parameters is not straightforward as a more conservative set of parameters also discards depth maps on uniformly textured areas.

In Figure 1 we show depths estimated by Colmap. The top row shows the 4 out of 8 problematic scenes that are unsuitable for reconstruction evaluation. These 4 scenes are rooms with challenging geometries that are also within Hypersim dataset distrubution. The remaining 4 scenes are object-centric.

8. Visualizations of DTU results

Figure 5 shows some qualitative results on the DTU dataset in addition to those given in the main paper.

9. Hypersim Patches

Figure 6 shows more examples of patches extracted from the Hypersim dataset used for our DDM training. Figure 7 shows more examples of patches synthesized using our DDM.

10. MipNeRF-360 Dataset Experiments

We also tested our model on the MipNeRF-360 [1] dataset. This is a challenging dataset in the few-view regime, because many areas of the scene are not visible in many training views. This is in contrast to forward-facing datasets such as LLFF in which most of the scene is visible from every training view. We find that for this reason, NeRF reconstructions tend to fail catastrophically in the 3-

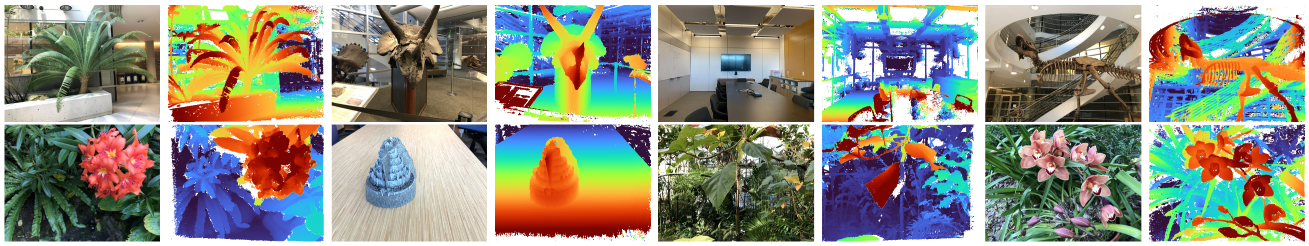

Figure 1. LLFF dataset images and depths estimated using Colmap. Top-row shows room-scale scenes and bottom-row shows object-centered scenes. Notice noisy depths in ceilings, glass surfaces, reflective surfaces and walls.

Method	PSNR ↑			SSIM ↑			LPIPS-Alex ↓			Average ↓		
	9-view	18-view	27-view	9-view	18-view	27-view	9-view	18-view	27-view	9-view	18-view	27-view
Instant-NGP [4]	**15.69**	17.86	19.54	0.241	0.365	0.457	0.654	0.483	0.375	0.234	0.181	0.158
Geometric Baseline	15.19	<u>19.94</u>	<u>22.06</u>	<u>0.283</u>	<u>0.484</u>	<u>0.583</u>	<u>0.536</u>	<u>0.343</u>	<u>0.273</u>	<u>0.220</u>	<u>0.146</u>	<u>0.123</u>
DiffusioNeRF (Ours)	<u>15.44</u>	**20.25**	**22.41**	**0.307**	**0.515**	**0.600**	**0.503**	**0.316**	**0.247**	**0.207**	**0.143**	**0.121**

Table 1. Results for novel view synthesis task on Mipnerf-360 datasets with few input views [5, 10]. We report scores on PSNR, SSIM, LPIPS and Average metrics averaged over all 8 scenes when NeRFs are fitted with 9, 18 and 27 training views. For each view/metric combination the **first** and <u>second</u> scores are highlighted.

and 6-view cases, even with our geometric and diffusion-based regularisers. Therefore we make the task easier by tripling the number of training views from 3, 6 and 9 to 9, 18 and 27. We find that our geometric baseline is an improvement over instant-NGP, and our diffusion-based regularizer offers a further improvement over the geometric baseline. Results are shown in table 1, and qualitative comparisons against Instant NGP and Geometric Baseline are shown in Figures 8, 9 and 10. As shown on the qualitative figures, catastrophic failures still occur sometimes even with this higher number of training views.

11. Synthetic Dataset Experiments

We also tested our model on the synthetic dataset introduced in [3]. These scenes include a solid white background, so unlike real scenes it is now reasonable for a ray to pass through the scene without being fully absorbed. For this reason we set λ_{fg} – the weight for the loss term that encourages the weights along each ray to sum to unity – to zero. We also set the background color of our NeRF models to white, matching the background on the training views.

The results are shown in table 2. We find our full model to outperform both Instant-NGP and our geometric baseline on all metrics with 3, 6, and 9 training views. However, the improvements over Instant-NGP are relatively modest. This may suggest that part of the benefit of our model is that it creates robustness against inaccuracies in the intrinsics and extrinsics of the training data, a source of error which is absent in synthetic datasets such as this one.

Qualitative comparisons against Instant NGP and the Geometric Baseline are shown in Figures 11, 12 and 13.

12. DiffusionSDF

Since our diffusion model-based regularizer can be used with any model which can differentiably render RGBD patches, we also tried using it with SDF (signed distance function)-based models. Models based on fitting an SDF generally reconstruct more accurate geometry than do models that fit a density field. We integrated our regularizer with the MonoSDF [11] code. To better isolate the effect of our regularizer, we switched off the smoothing, depth and normals regularization terms in MonoSDF.

When training on few views, we found that results were often quite poor, with the geometry often looking less smooth rather than more smooth (in contrast to the behaviour observed when using our regularizer with Instant-NGP). It is possible that the problem occurs because SDF-based methods must generally be fit with a lower learning rate than density-based methods to prevent instabilities during training. Spreading out the fitting process over many steps helps the SDF to remain valid during the optimization process (i.e. to satisfy to a good approximation the Eikonal condition $\|\nabla f(x)\| = 1$, where $f(x)$ is the signed distance function). Our patch-based regularizer generates large, highly localized gradients in the part of the scene that the current patch is looking at. The injection of these gradients makes it difficult for the SDF to remain valid. It is likely that this could be addressed by lowering the learning rate still further, but fitting MonoSDF to a single scene with our patch-based regularizer already takes 1 day, so the running time would become prohibitive.

Therefore we conclude that our regularizer is not well-suited for combination with SDF-based models. Nevertheless, we show qualitative results on a single scene from the

Method	PSNR ↑			SSIM ↑			LPIPS-Alex ↓			Average ↓		
	3-view	6-view	9-view	3-view	6-view	9-view	3-view	6-view	9-view	3-view	6-view	9-view
Instant-NGP [4]	16.31	19.71	22.57	0.653	0.767	0.819	0.388	0.211	0.136	0.136	0.060	0.053
Geometric Baseline	<u>16.37</u>	<u>19.84</u>	<u>22.73</u>	<u>0.670</u>	<u>0.774</u>	<u>0.826</u>	<u>0.358</u>	<u>0.201</u>	<u>0.129</u>	<u>0.133</u>	<u>0.057</u>	<u>0.050</u>
DiffusioNeRF (Ours)	**16.45**	**20.42**	**23.45**	**0.681**	**0.787**	**0.838**	**0.323**	**0.177**	**0.115**	**0.120**	**0.053**	**0.047**

Table 2. Results for novel view synthesis task on the blender synthetic dataset with few input views. We report scores on PSNR, SSIM, LPIPS and Average metrics averaged over all 8 scenes when NeRFs are fitted with 3, 6 and 9 training views. For each view/metric combination the **first** and <u>second</u> scores are highlighted.

DTU dataset, with and without our regularizer in Figure 14.

13. 3D Voxel Grid Regularization

As discussed in the conclusion of the main paper, we have also begun to experiment with using diffusion models to regularise the model directly in 3D, rather than by regularising 2D rendered patches. This is an appealing approach because each voxel requires just one evaluation of the colour and density, whereas each pixel in a patch requires hundreds of such evaluations to render it using ray-marching. Additionally it allows the regulariser direct access to the 3D geometry of the scene, which may be more informative than only being able to see 2D projections of it. In this section we describe some early results of these experiments. We use the KLEVR toy dataset introduced in Neural Semantic Fields (NeSF) [8]. KLEVR consists of synthetic renders of simple geometric shapes on a gray plane against a white background; we show three example scenes from the dataset in Figure 15.

We wish to train a DDM model to learn the 3D geometry of KLEVR directly, so we require 3D training data. To generate the training data for a given scene from KLEVR, we backproject the ground-truth depth maps provided as part of the dataset and accumulate them across all views to form a point cloud. We then voxelise the point cloud into a voxel grid of size $128 \times 128 \times 128$. Each voxel stores four channels. The first three are RGB colour channels as in our 2D experiments, and the fourth is occupancy. We set the occupancy of each voxel based on the distance to the nearest point in the point cloud; it is normalised to between zero and one.

To train the model we pick random $48 \times 48 \times 48$ 3D chunks of the voxel grid – analogously to the way we pick random 2D patches when training our 2D model – and train the model as usual, by applying noise and tasking the model to predict the noise. The architecture of the diffusion model is unchanged, except for the extra dimensionality of its inputs and outputs.

While fitting NeRFs, we then use the diffusion model to regularise the NeRF by sampling the model on a random $48 \times 48 \times 48$ grid, feeding the sampled grid into the diffusion model, and backpropagating the negative of the predicted noise through the NeRF. This is just the same approach as we use in our 2D experiments, except that we backpropagate into the sampled 3D grid, rather than into a rendered patch. To convert the density (which ranges from zero to infinity) to an occupancy (which ranges from zero to one) for input to the diffusion model, we normalise the density using:

$$\zeta(\mathbf{r}) = 1 - \exp(-\gamma \sigma(\mathbf{r})) \tag{3}$$

where ζ is the occupancy of the voxel whose centre is at position $\mathbf{r}$, $\sigma(\mathbf{r})$ is the density evaluated at that position, and γ is a hyperparameter controlling how high the density must be for us to consider a voxel to be occupied; we set it to 0.1.

Although we have used a very simple dataset, early results are promising and merit further investigation on more realistic data. We show an example result in Figure 17, in which the model successfully improves the geometry of the fitted NeRF. Because the three training views are from very similar positions, the geometry in Figure 17(b) is poorly constrained and is smeared out parallel to the look vectors of the training views. However, when our 3D regularizer is used, the results are substantially improved (Figure 17(c)).

References

[1] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In *CVPR*, 2022. 2

[2] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *NeurIPS*, 2020. 1

[3] 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 *ECCV*, 2020. 3

[4] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM TOG*, 2022. 3, 4

[5] Michael Niemeyer, Jonathan T. Barron, Ben Mildenhall, Mehdi S. M. Sajjadi, Andreas Geiger, and Noha Radwan. RegNeRF: Regularizing Neural Radiance Fields for View Synthesis from Sparse Inputs. In *CVPR*, 2022. 3

[6] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-Motion Revisited. In *CVPR*, 2016. 2

[7] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016. 2

[8] Suhani Vora, Noha Radwan, Klaus Greff, Henning Meyer, Kyle Genova, Mehdi S. M. Sajjadi, Etienne Pot, Andrea Tagliasacchi, and Daniel Duckworth. Nesf: Neural semantic fields for generalizable semantic segmentation of 3d scenes, 2021. 4

[9] Phil Wang. Denoising Diffusion Probabilistic Model in Pytorch, 2022. github.com/lucidrains/denoising-diffusion-pytorch. 1

[10] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021. 3

[11] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. MonoSDF: Exploring Monocular Geometric Cues for Neural Implicit Surface Reconstruction. In *NeurIPS*, 2022. 3

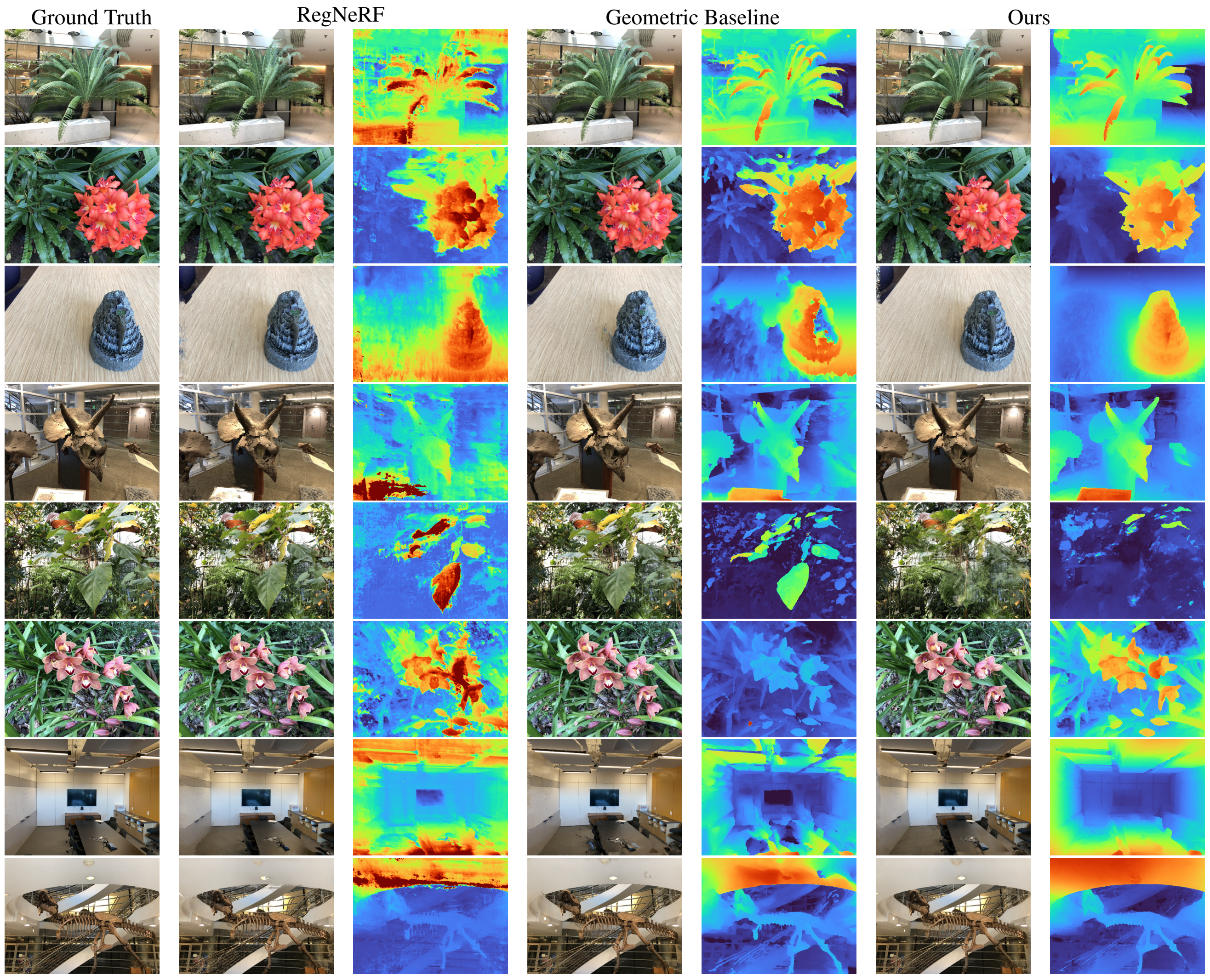

Figure 2. Additional qualitative results for the task of novel view synthesis on LLFF dataset. NeRF models are trained with 3 views and rendered from one of the test views. Our DDM model encourages more realistic geometry as seen in the depth maps.

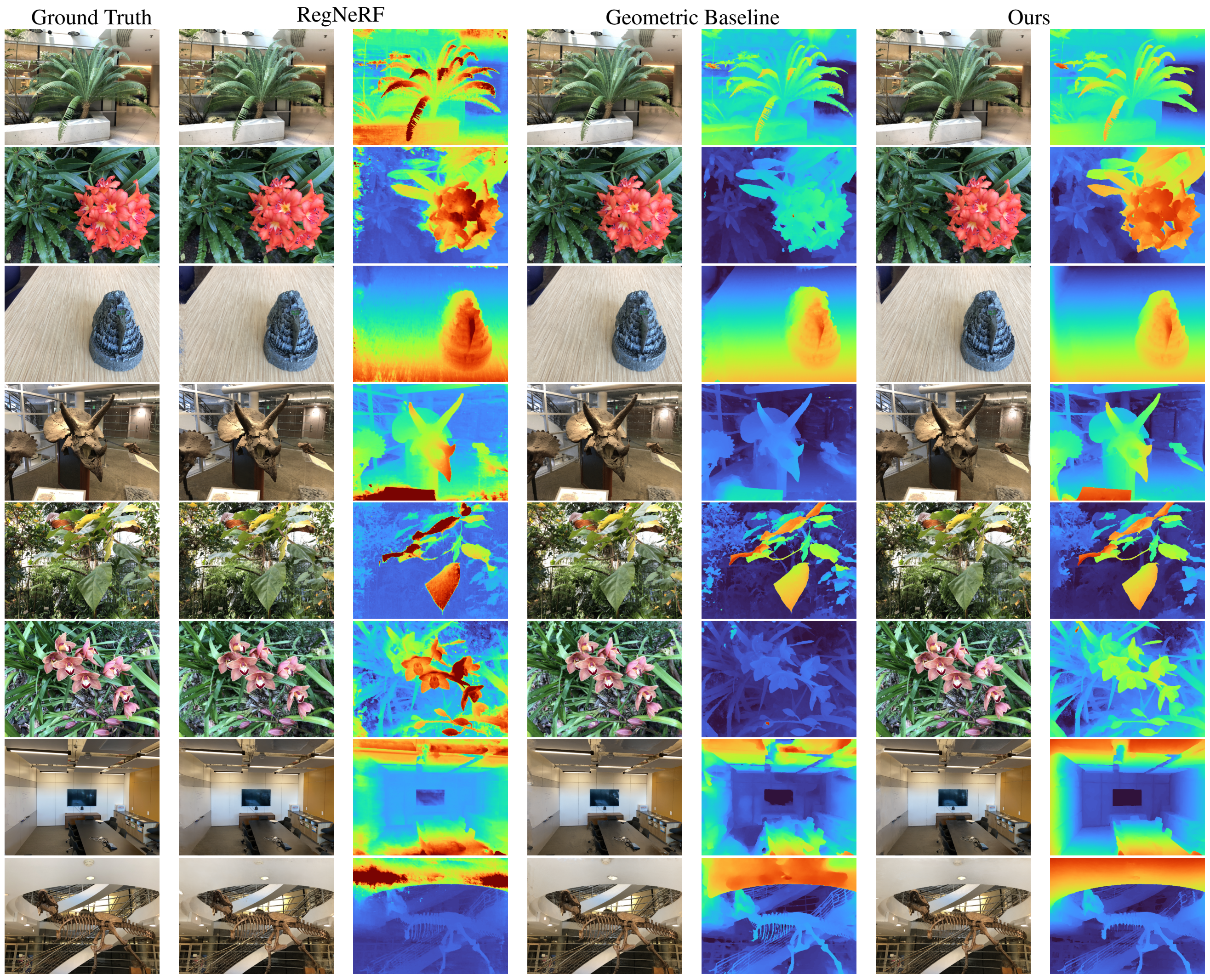

Figure 3. Additional qualitative results for the task of novel view synthesis on LLFF dataset. NeRF models are trained with 6 views and rendered from one of the test views.

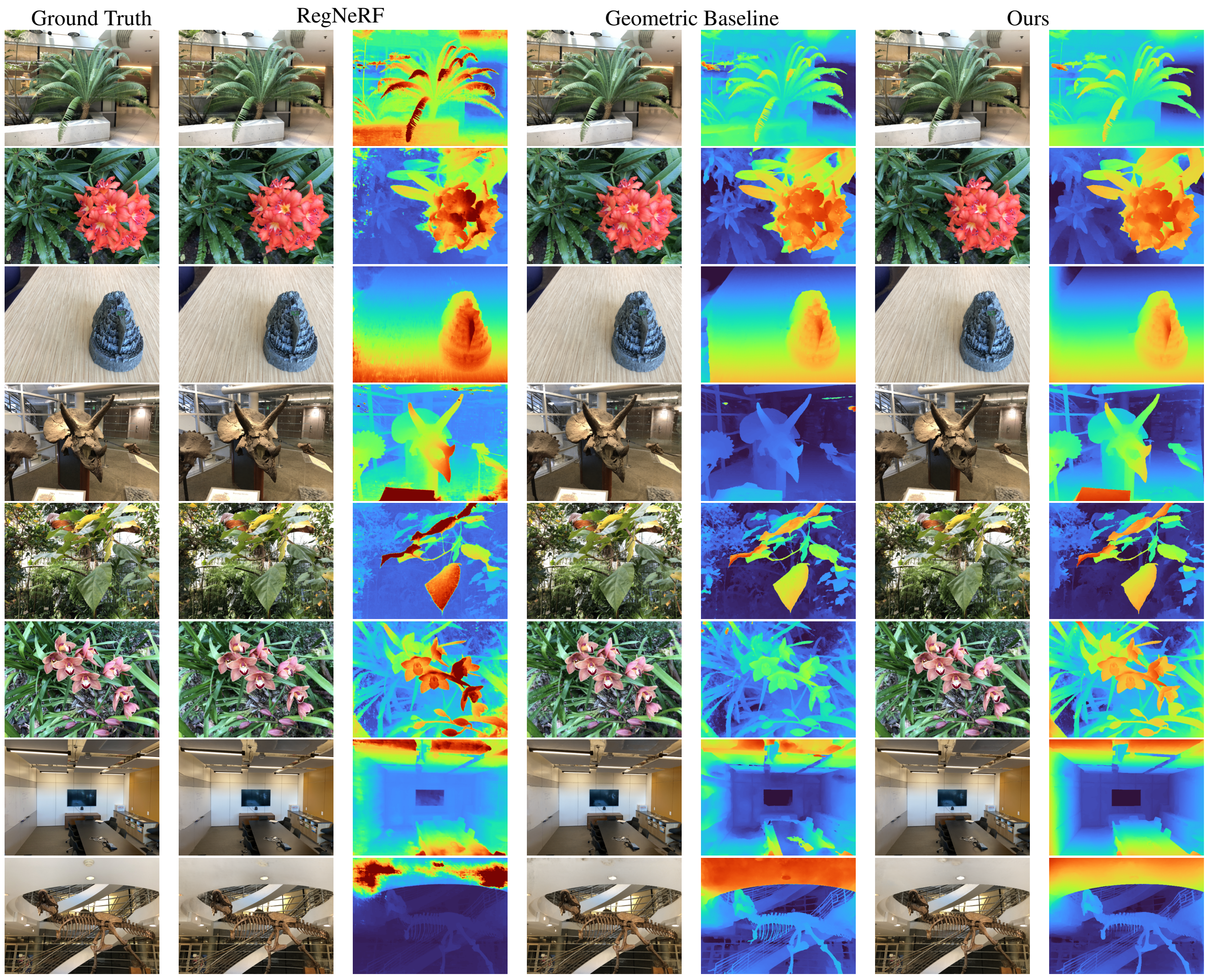

Figure 4. Additional qualitative results for the task of novel view synthesis on LLFF dataset. NeRF models are trained with 9 views and rendered from one of test views.

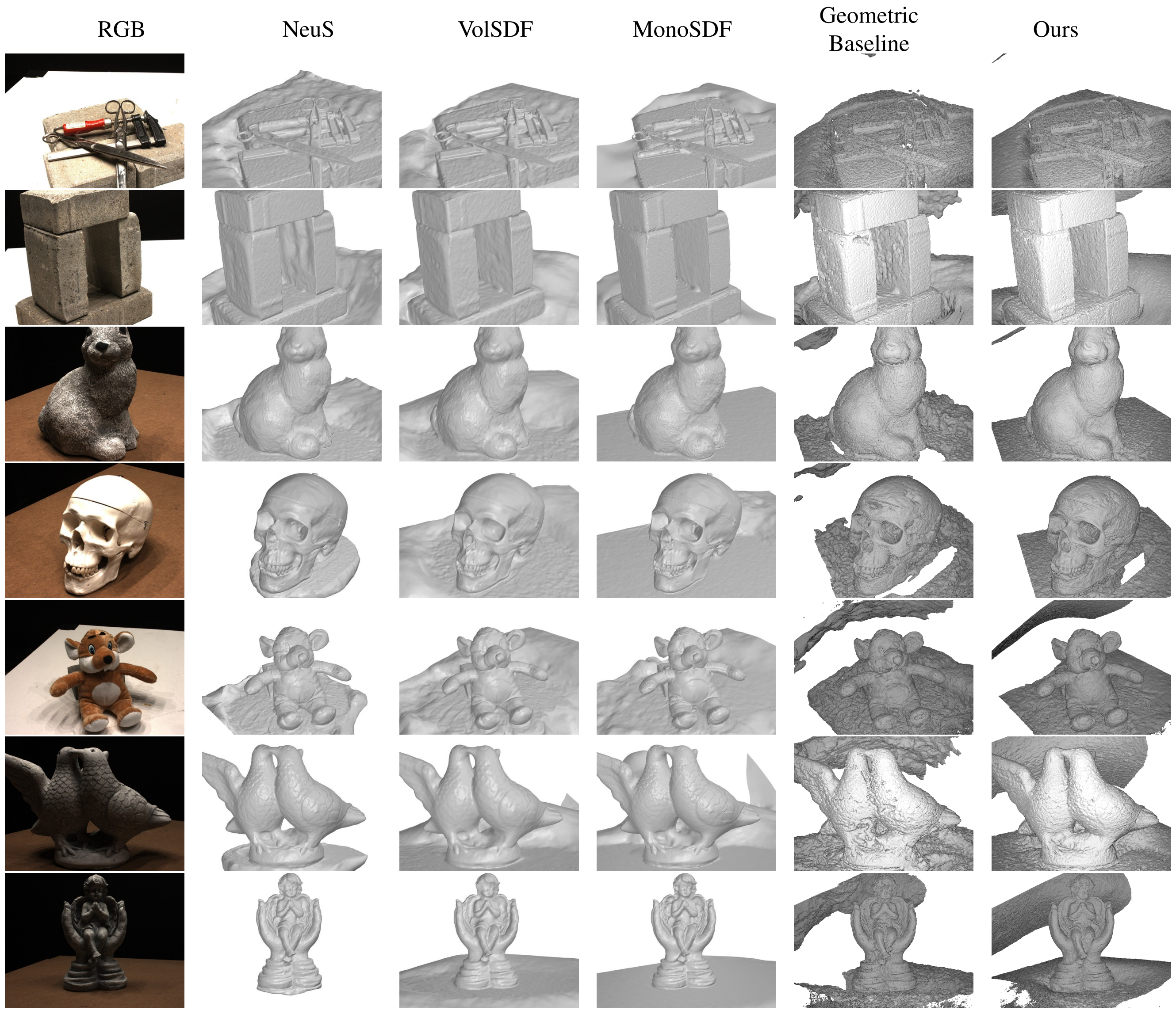

Figure 5. Additional qualitative comparison of our method against SOTA on geometry reconstruction evaluated on DTU dataset.

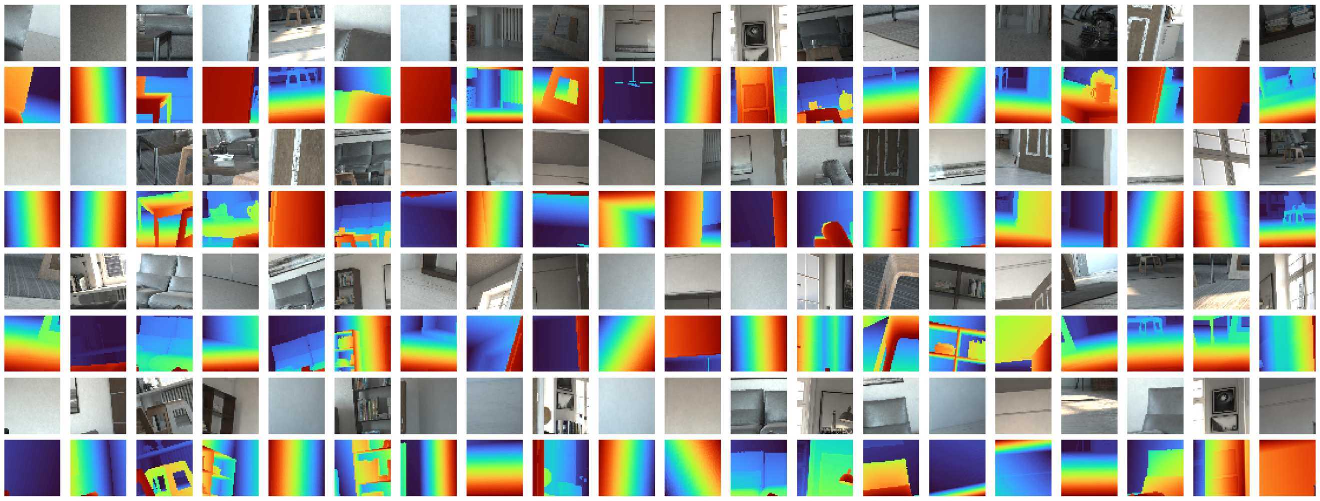

Figure 6. Example RGBD patches in the training set of the DDM model extracted from Hypersim dataset. Depths are shown as normalized inverse depths for visualization purposes.

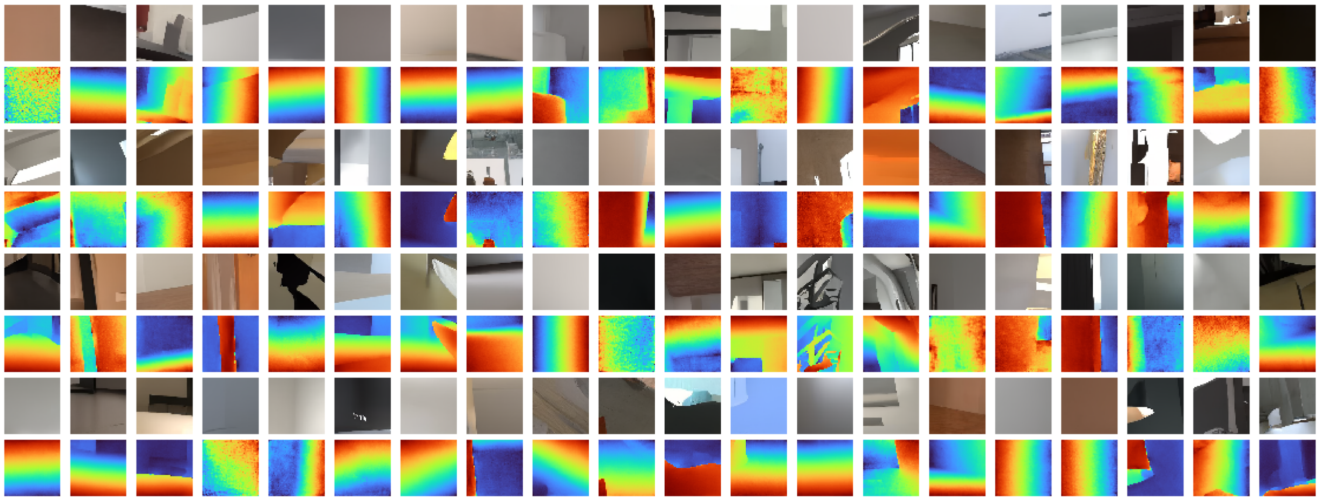

Figure 7. Example RGBD patches generated with our DDM model trained on Hypersim dataset. Depths are shown as normalized inverse depths for visualization purposes.

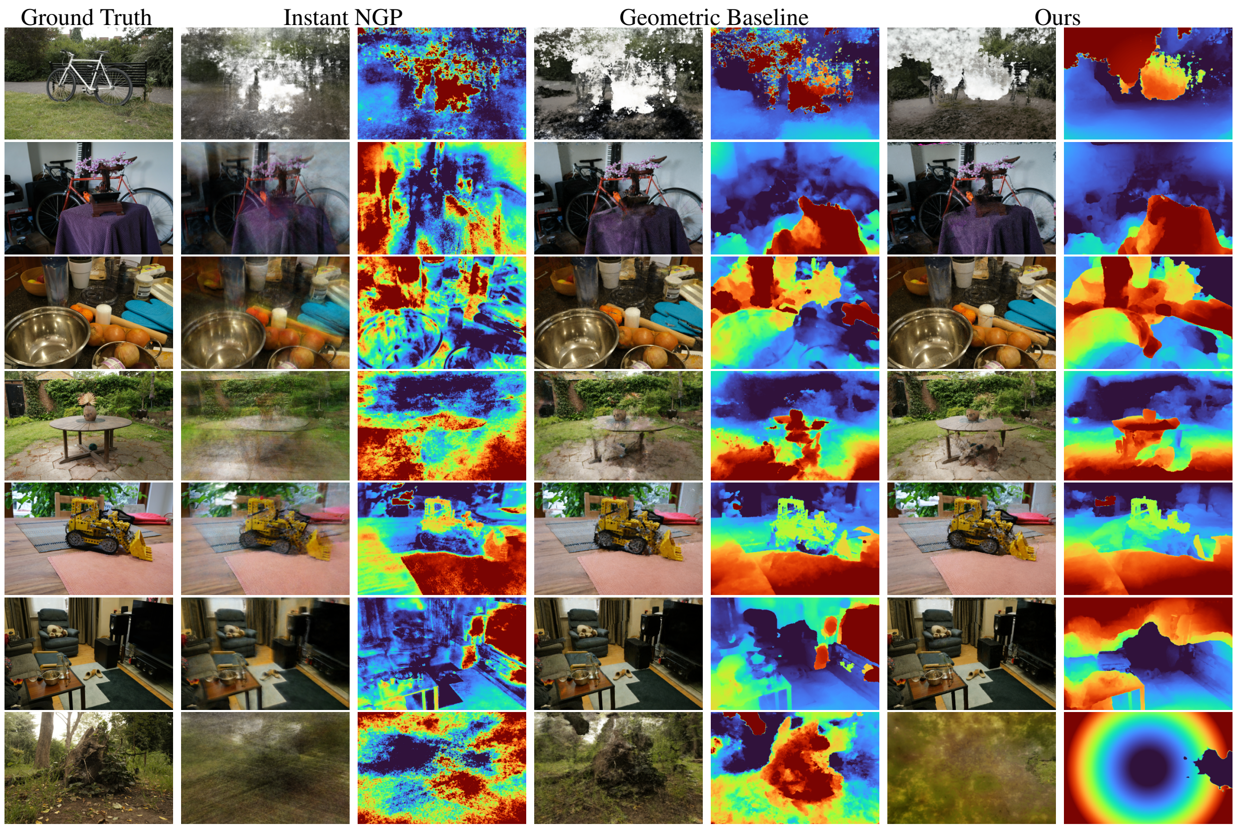

Figure 8. Additional qualitative results for the task of novel view synthesis on MipNeRF-360 dataset. NeRF models are trained with 9 views and rendered from one of test views.

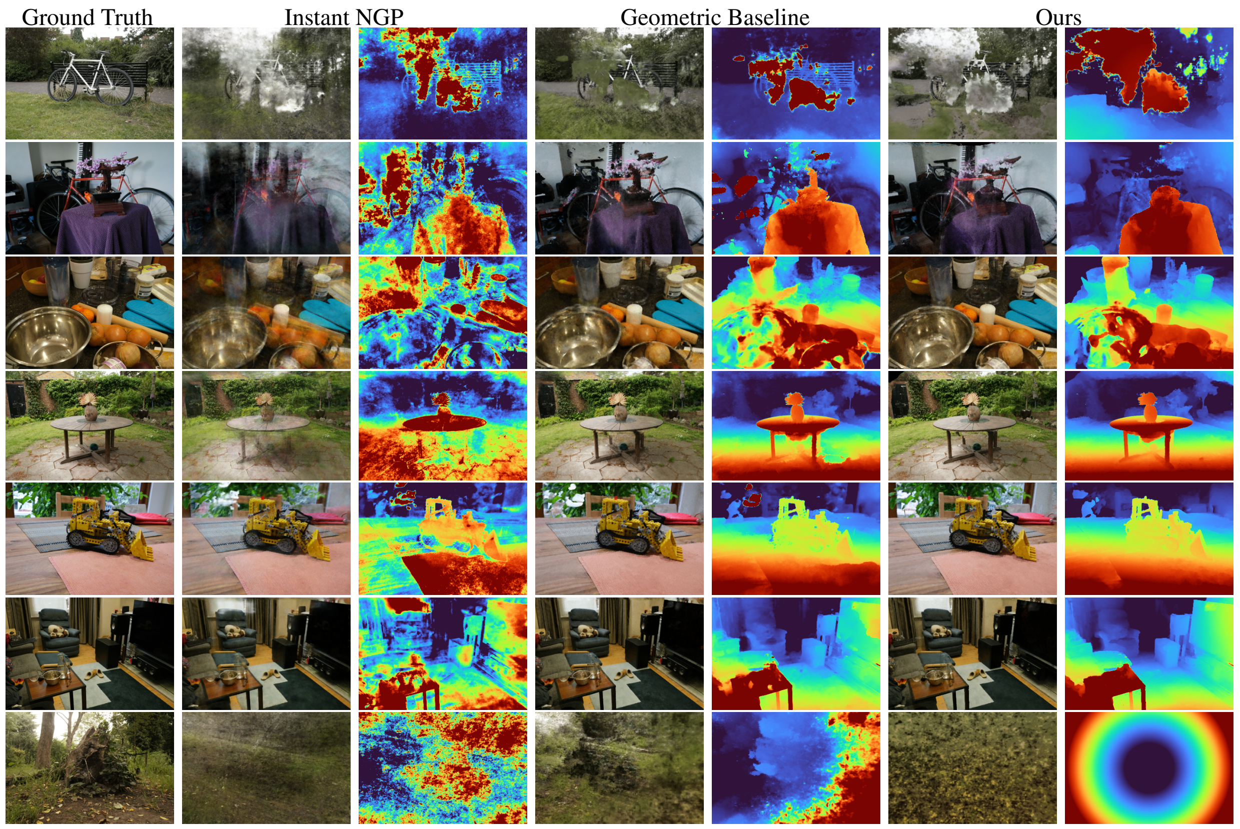

Figure 9. Additional qualitative results for the task of novel view synthesis on MipNeRF-360 dataset. NeRF models are trained with 18 views and rendered from one of test views.

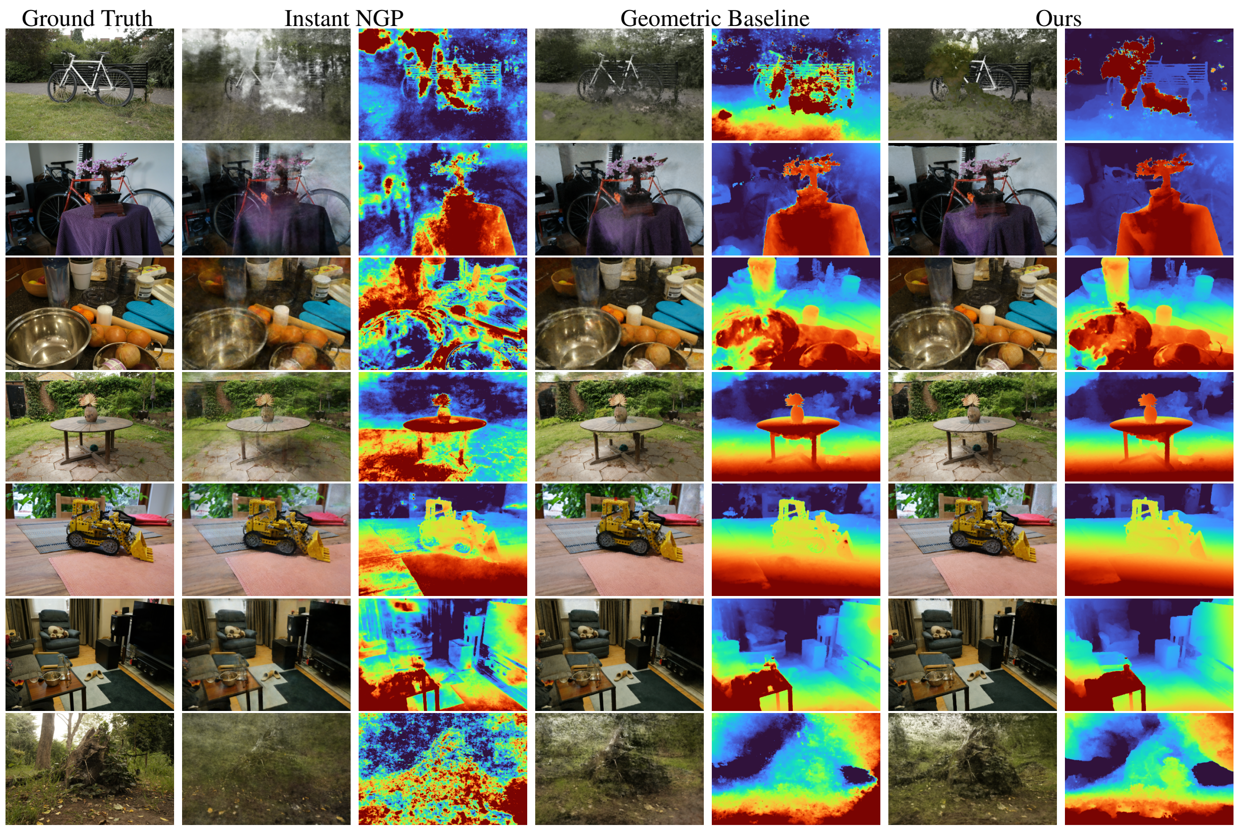

Figure 10. Additional qualitative results for the task of novel view synthesis on MipNeRF-360 dataset. NeRF models are trained with 27 views and rendered from one of test views.

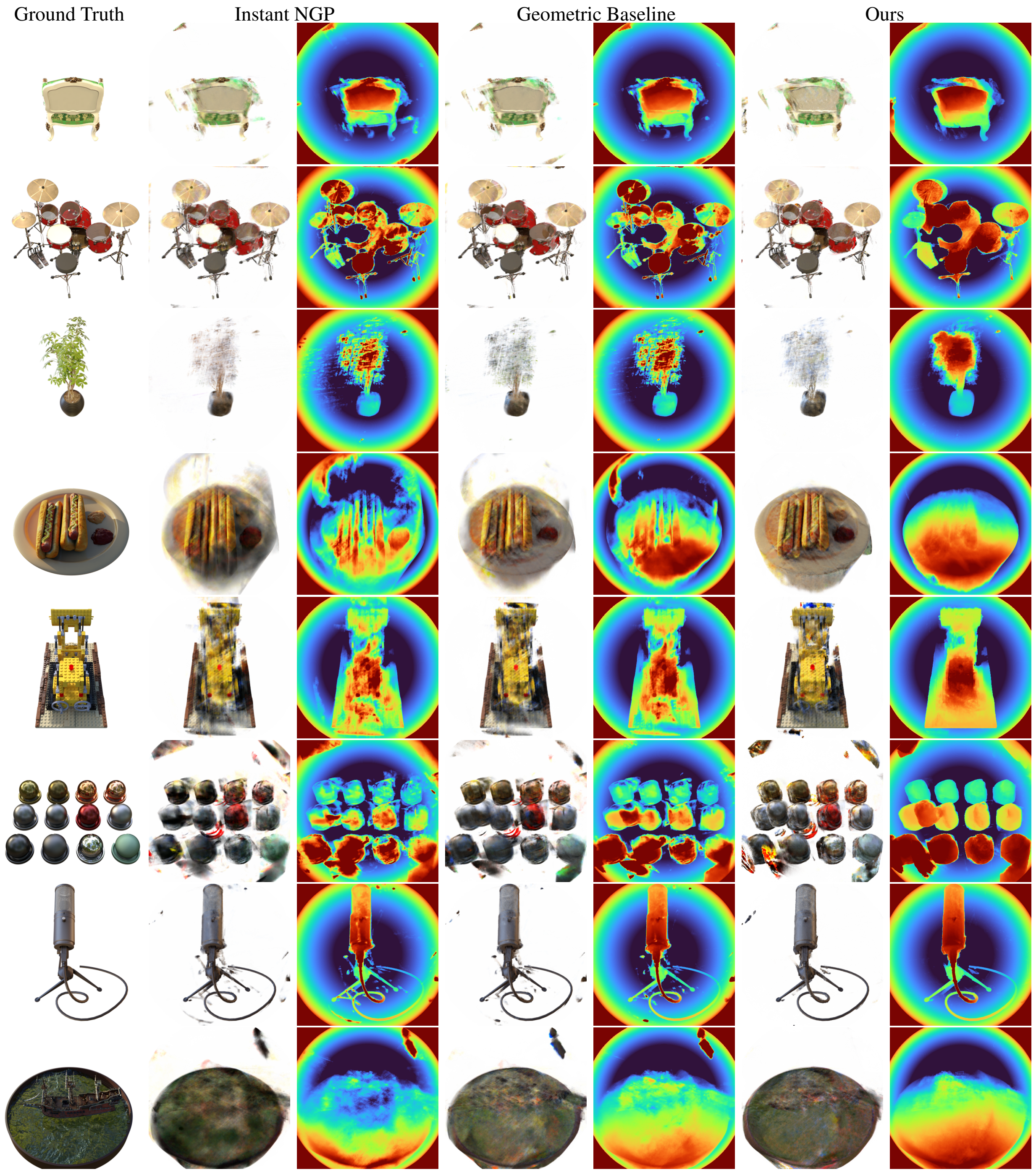

Figure 11. Additional qualitative results for the task of novel view synthesis on NeRF Synthetic Blender dataset. NeRF models are trained with 3 views and rendered from one of test views.

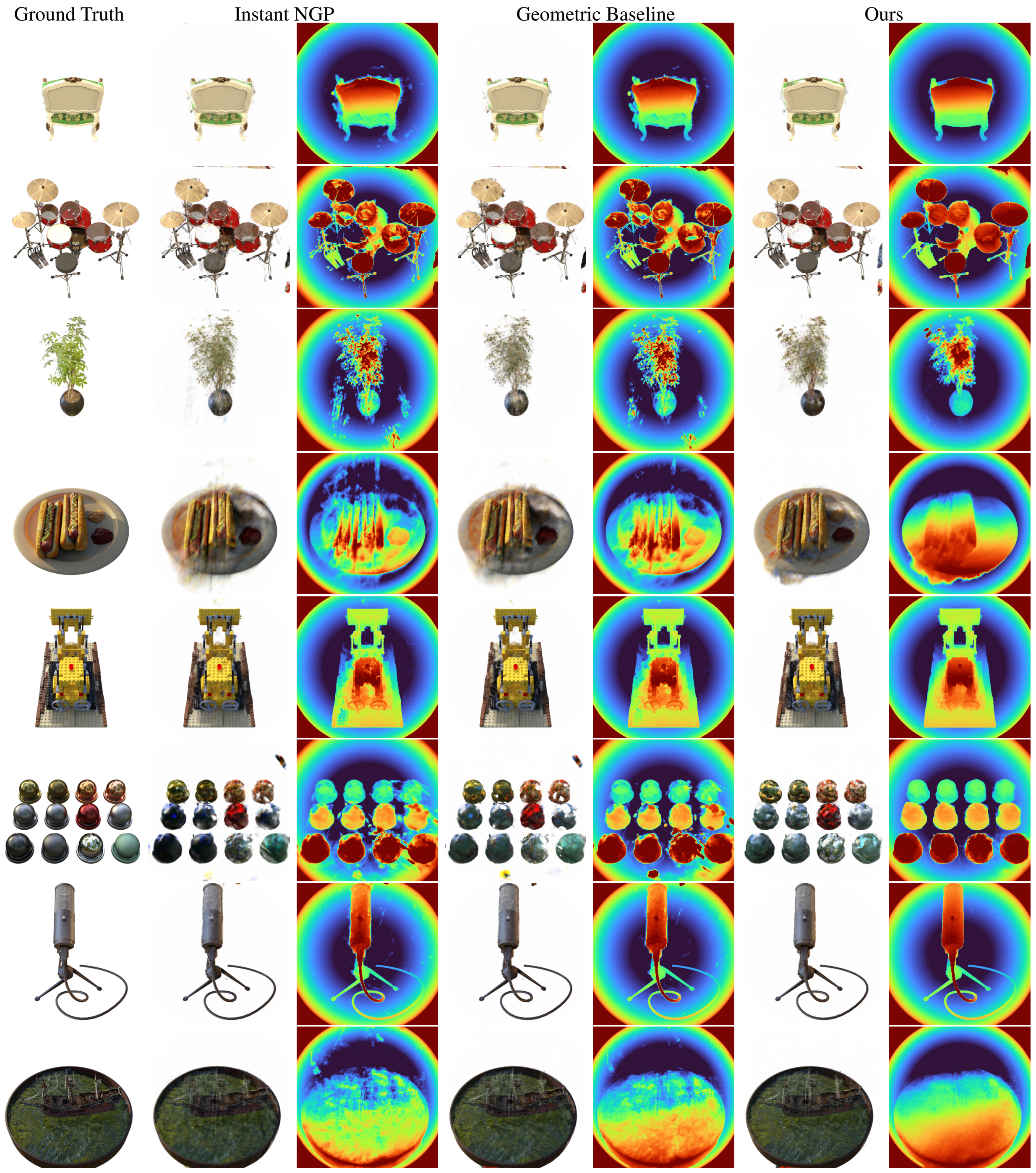

Figure 12. Additional qualitative results for the task of novel view synthesis on NeRF Synthetic Blender dataset. NeRF models are trained with 6 views and rendered from one of test views.

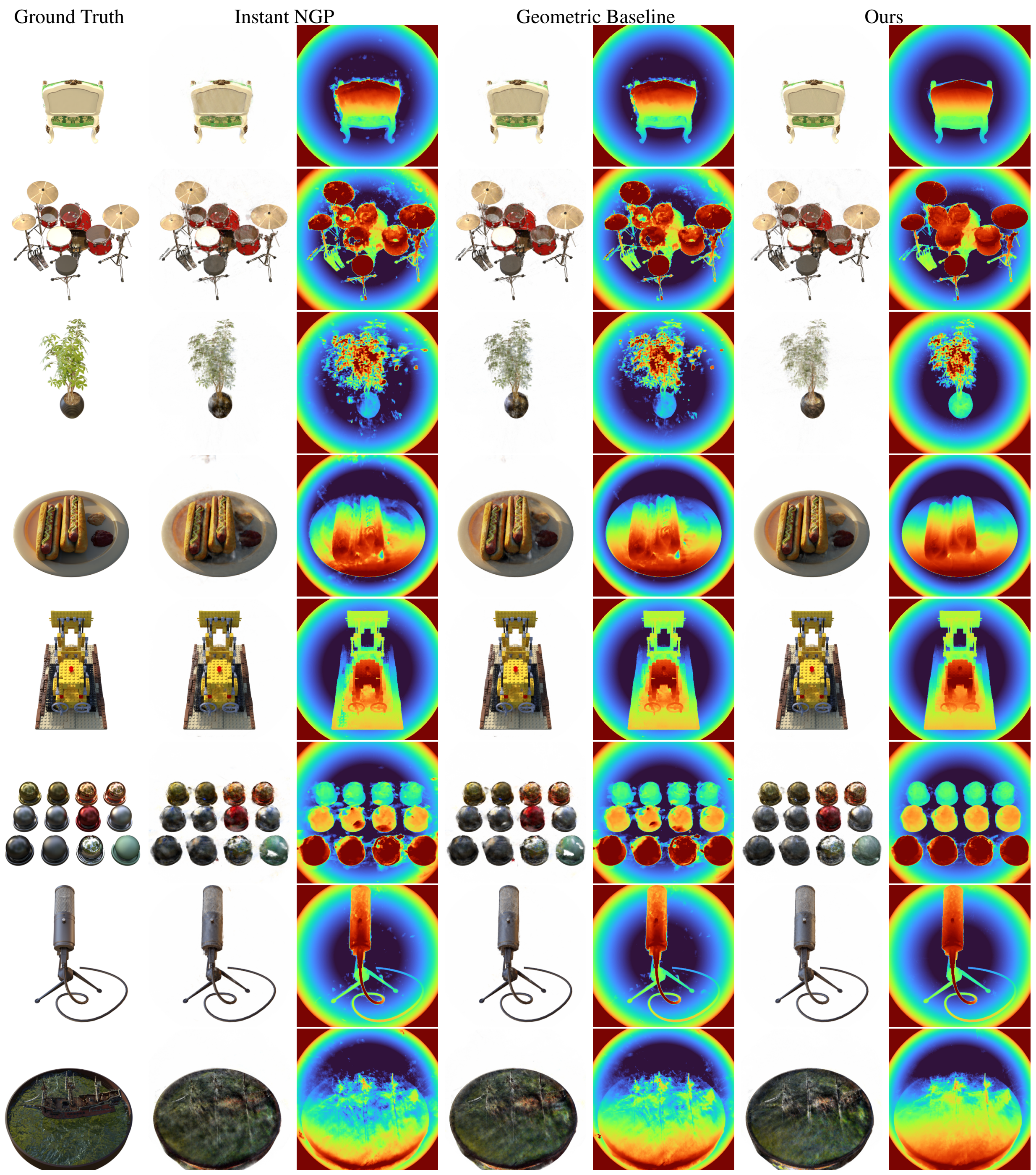

Figure 13. Additional qualitative results for the task of novel view synthesis on NeRF Synthetic Blender dataset. NeRF models are trained with 9 views and rendered from one of test views.

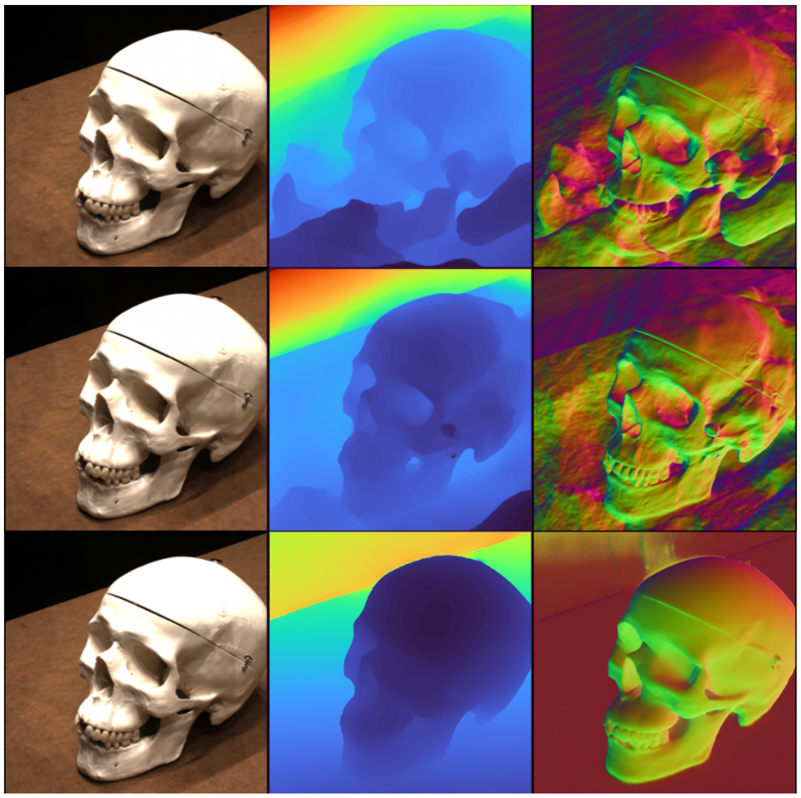

Figure 14. Rendered training view (left), depth maps (middle) and normals (right) for the DTU ‘skull’ scene. The top row shows the baseline plus our diffusion regularizer, the middle row shows the baseline alone, and the bottom row shows the ground truth.

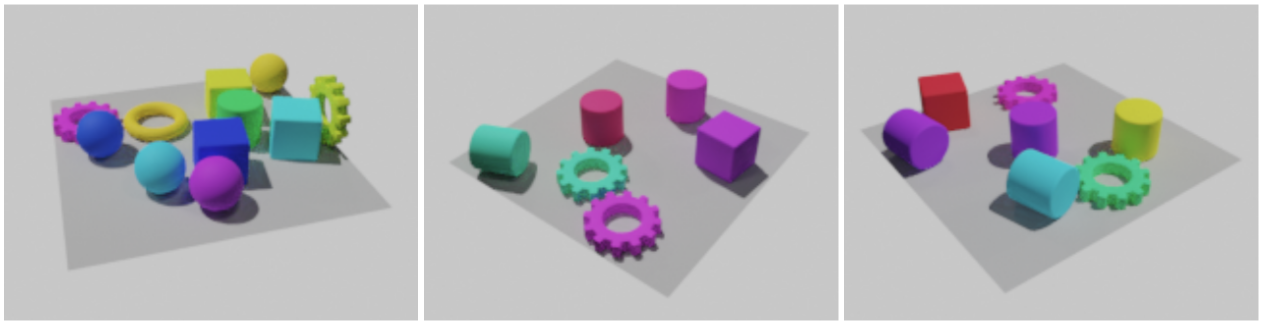

Figure 15. Training views of three different scenes in the KLEVR dataset.

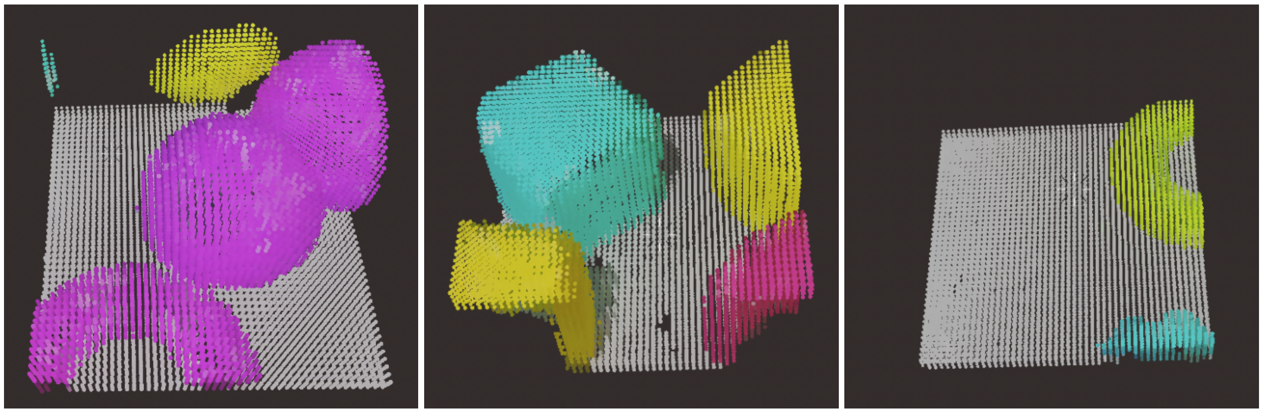

Figure 16. Three voxel grids generated by the DDM, showing each voxel of occupancy greater than $1/2$ as a point.

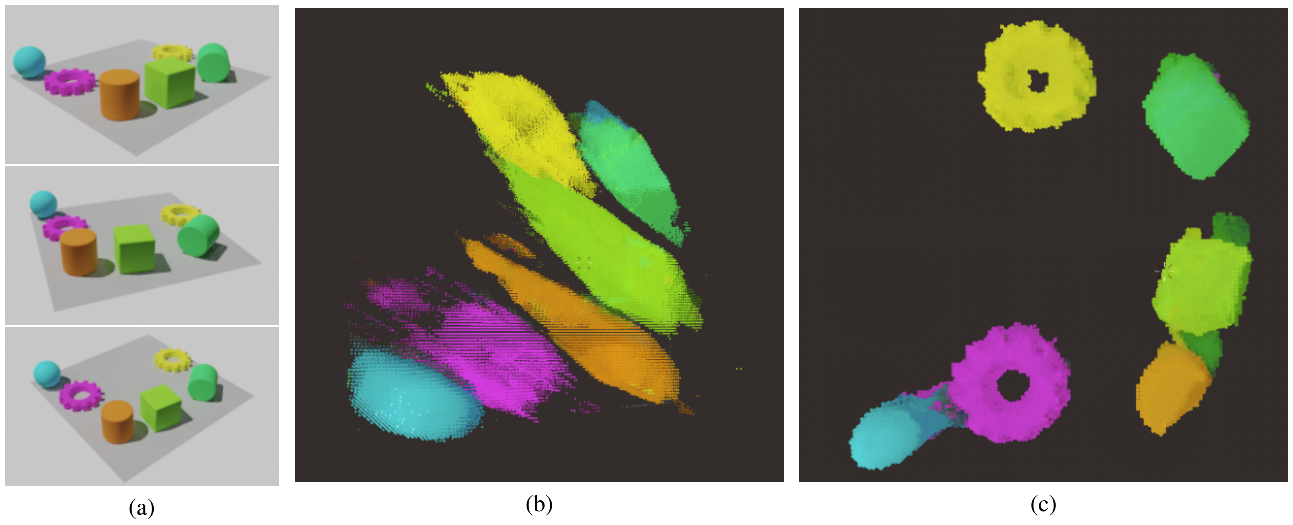

Figure 17. Results for regularisation of the Hypersim dataset with a 3D diffusion model, showing (a) the three training views we use to train NGP on this scene, (b) a point cloud extracted from a NeRF fitted without our 3D regulariser, (c) a point cloud extracted from a NeRF fitted using our 3D regulariser.