Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction

Supplementary material

In Sec. A, we give more implementation details. In Sec. B, we present additional ablation studies for the important hyperparameters that affect the quality and convergence speed. In Sec. C, we show detailed results of our main ablation studies presented in the main paper. In Sec. D, we report our training time breakdown of the coarse and fine stage on each scene. In Sec. E, we show the detailed quantitative and qualitative results on each scene. Finally, we derive for the low-density initialization in Sec. F, and we prove that the proposed post-activated trilinear interpolation can produce sharp linear surfaces in Sec. G.

A. Additional implementation details

We choose the same hyperparameters generally for all scenes in the five datasets.

In the coarse stage, the expected number of voxels is $M^{(\text{c})} = 100^3$. The activated alpha values are initialized to be $\alpha^{(\text{init})(\text{c})} = 10^{-6}$.

In the fine stage, we use $M^{(\text{f})} = 160^3$ voxels. We use a higher $\alpha^{(\text{init})(\text{f})} = 0.01$ as the query points in the known free space are skipped. The shallow MLP layer comprises two hidden layers with 128 channels. The number of channels of the feature voxel grid $\boldsymbol{V}^{(\text{feat})(\text{f})}$ is set to $D = 12$. The number of frequencies of positional embedding for the query position $\boldsymbol{x}$ and the viewing direction $\boldsymbol{d}$ are $k_{\boldsymbol{x}} = 5, k_{\boldsymbol{d}} = 4$. We progressively scale the fine-stage voxel grid at the checkpoint pg_ckpt being chosen as the 1000th, 2000th, and 3000th training steps. We use threshold $\tau^{(\text{c})} = 10^{-3}$ to define the known free space and $\tau^{(\text{f})} = 10^{-4}$ to skip the low-density query points in unknown space.

We set the sampling step sizes to be half of the voxel sizes, *i.e.*, $\delta^{(\text{c})} = 0.5s^{(\text{c})}$ and $\delta^{(\text{f})} = 0.5s^{(\text{f})}$. During training, we randomly shift the query points on a ray by $\left(p \cdot \delta^{(\cdot)} \cdot \boldsymbol{d}/\|\boldsymbol{d}\|^2\right)$, where p is sampled uniformly in $[0, 1]$, and $\boldsymbol{d}$ is the viewing direction of the ray.

In addition to the photometric loss, we further incorporate per-point rgb loss and background entropy loss. The per-point rgb loss directly supervises each of the K ordered sampled points instead of the accumulated ray:

$$\mathcal{L}_{\text{pt\_rgb}} = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \sum_{i=1}^{K} \left(T_i \alpha_i \left\| \boldsymbol{c}_i - C(\boldsymbol{r}) \right\|_2^2 \right). \quad (1)$$

The background entropy loss regularizes the rendered background probability T_{K+1} to concentrate on either foreground or background:

$$\mathcal{L}_{\text{bg}} = -T_{K+1} \log(T_{K+1}) - (1 - T_{K+1}) \log(1 - T_{K+1}). \quad (2)$$

Finally, the overall training objective of the coarse stage is

$$\mathcal{L} = w_{\text{photo}} \cdot \mathcal{L}_{\text{photo}} + w_{\text{pt\_rgb}} \cdot \mathcal{L}_{\text{pt\_rgb}} + w_{\text{bg}} \cdot \mathcal{L}_{\text{bg}}, \quad (3)$$

where $w^{(\text{c})}_{\text{photo}}, w^{(\text{c})}_{\text{pt\_rgb}}, w^{(\text{c})}_{\text{bg}}$ are the loss weights.

The loss weights are set as follows: $w^{(\text{c})}_{\text{photo}} = 1$, $w^{(\text{c})}_{\text{pt\_rgb}} = 10^{-1}$, $w^{(\text{c})}_{\text{bg}} = 10^{-2}$, $w^{(\text{f})}_{\text{photo}} = 1$, $w^{(\text{f})}_{\text{pt\_rgb}} = 10^{-2}$, and $w^{(\text{f})}_{\text{bg}} = 10^{-3}$. We use the Adam optimizer with a batch size of 8,192 rays to optimize the coarse and fine scene representations for 10k and 20k iterations. The base learning rates are 0.1 for all voxel grids and 10^{-3} for the MLP. The exponential learning rate decay is applied such that the learning rates are downscaled by 0.1 at 20k iterations.

B. Additional ablation experiments

We use the *Robot* scene to conduct the additional ablation experiments. In all the additional ablation experiments, the fine stage are early stopped at 10k iterations. The reported training times are measured on our machine with a single RTX2080 Ti GPU.

Effect of the number of voxels. The hyperparameters $M^{(\text{c})}$ and $M^{(\text{f})}$ define the numbers of voxels in the coarse stage and the fine stage. In Tab. B1, we show the effect of different $M^{(\cdot)}$ setups on the final PSNRs and the training times. We can use more voxels for better quality or use fewer voxels for faster convergence speed.

Effect of the point sampling step size. In our design, the voxel sizes $s^{(\cdot)}$ are automatically derived from the hyperparameters of the number of voxels $M^{(\cdot)}$. We set the step-size-to-voxel-size ratio instead of directly assigning the step

$M^{(c)}$	$M^{(f)}$	PSNR↑	Training minutes↓
100^3	100^3	32.65	**5.1**
100^3	136^3	34.57	7.0
100^3	160^3	**35.54**	7.3

(a) Using fewer voxels in the fine stage results in inferior qualities but faster training speed.

$M^{(c)}$	$M^{(f)}$	PSNR↑	Training minutes↓
64^3	160^3	34.91	**6.1**
100^3	160^3	35.54	7.3
136^3	160^3	**35.84**	9.3

(b) The number of voxels in the coarse stage can also affect final results as the fine stage relies on the coarse geometry optimized in the coarse stage.

Table B1. Effect of the number of voxels in the coarse stage ($M^{(c)}$) and the fine stage ($M^{(f)}$). Using more voxels can improve quality with the cost of more computation and training time.

size for points sampling on rays, *i.e.*, $\delta^{(\cdot)} = 0.5s^{(\cdot)}$ means that the step size is half a voxel size. We show in Tab. B2 that the step sizes are also important hyperparameters for the speed-quality tradeoff. We can also improve the rendering quality of a trained model by using a finer step size. For the sake of simplicity, we leave future work to explore the effect of different hierarchical point sampling strategies [1, 12, 13].

Step size		PSNR↑	sec/frame↓
Training	Testing		
$\delta^{(\cdot)} = 0.50s^{(\cdot)}$	$\delta^{(\cdot)} = 0.50s^{(\cdot)}$	35.54	**0.64**
$\delta^{(\cdot)} = 0.50s^{(\cdot)}$	$\delta^{(\cdot)} = 0.25s^{(\cdot)}$	**35.72**	1.15

(a) We can improve quality by using a finer step size in test-time with the cost of slower rendering speed. The two results share a trained model and are only different in the testing step size.

Step size	PSNR↑	Training minutes↓
$\delta^{(\cdot)} = 1.00s^{(\cdot)}$	34.17	**5.2**
$\delta^{(\cdot)} = 0.75s^{(\cdot)}$	34.96	5.9
$\delta^{(\cdot)} = 0.50s^{(\cdot)}$	35.54	7.3
$\delta^{(\cdot)} = 0.25s^{(\cdot)}$	**36.01**	11.5

(b) Training with finer step size can improve results with the cost of more computation and training time.

Table B2. Effect of the step size in points sampling. The step size $\delta^{(\cdot)}$ is relative to the voxel size $s^{(\cdot)}$, and we apply the same step-size-to-voxel-size ratio in our coarse and fine stage. We can achieve better quality by using a smaller (finer) step size with the cost of more computation.

Effect of the progressive scaling in the fine stage. In the fine-stage, we scale the resolution of our voxel grids progressively. We show in Tab. B3 that such a strategy can improve the training efficiency with a slightly better rendering quality.

Effect of the free space skipping in the fine stage. There are three models in the fine stage—*i)* the optimized and

Progressive scaling	PSNR↑	Training minutes↓
	35.50	11.7
✓	**35.54**	**7.3**

Table B3. Scaling the grid resolutions progressively in the fine stage can improve training efficiency with a slightly better quality.

frozen coarse density voxel grid $\boldsymbol{V}^{(\text{density})(c)}$, *ii)* the finer density voxel grid $\boldsymbol{V}^{(\text{density})(f)}$, and *iii)* the model for the view-dependent color emission. Querying the optimized $\boldsymbol{V}^{(\text{density})(c)}$ is more efficient than querying the finer density $\boldsymbol{V}^{(\text{density})(f)}$; querying the view-dependent color emission is the slowest and computationally expensive. For each query point, we first query $\boldsymbol{V}^{(\text{density})(c)}$ and ignore the query point if the activated alpha is below a threshold $\tau^{(c)}$ (*i.e.*, the point is in the known free space). As the training progresses, we can filter out more points that are in the free space defined by the "sculpted" $\boldsymbol{V}^{(\text{density})(f)}$. Specifically, we ignore a query point if the activated alpha from $\boldsymbol{V}^{(\text{density})(f)}$ is below a threshold $\tau^{(f)}$. Thus, we only query for the view-dependent color if the query point passes the two criteria.

In Tab. B4, we show the effect of the free space skipping strategy. In the case of $\tau^{(c)} = 0$, the coarse voxel grid $\boldsymbol{V}^{(\text{density})(c)}$ is only used to determine a tighten BBox enclosing the scene. We finally run out of memory (OOM) at the fine stage if no skipping rule is applied ($\tau^{(c)} = 0, \tau^{(f)} = 0$). When only $\tau^{(f)}$ is applied, we can still sculpt out many irrelevant query points during the progressive scaling and save the memory before our voxel grids are scaled to the finest resolution. The rendering quality is degraded when $\tau^{(c)} = 0$, which suggests that the imposed priors in the coarse stage are helpful to the final quality (we cannot apply the low-density initialization and the free space skipping at the same time). Finally, we achieve the best rendering quality and convergence speed when both $\tau^{(c)}$ and $\tau^{(f)}$ are applied.

Free space skipping		PSNR↑	Training minutes↓
$\tau^{(c)}$	$\tau^{(f)}$		
0	0	-	OOM
0	10^{-4}	34.89	7.8
10^{-3}	0	**35.54**	15.6
10^{-3}	10^{-4}	**35.54**	**7.3**

Table B4. Effect of the free space skipping strategy in the fine stage training.

Ablation studies for the view-dependent color modeling. Our hybrid representation for the view-dependent color emissions in the fine stage comprises a feature voxel grid and a shallow MLP. In Tab. B5, we show different strategies and hyperparameters to model view-dependent colors. Applying only the shallow MLP results in worse outputs as our MLP is much "shallower" than the MLP in NeRF—our two layers with 128 channels versus NeRF's eight layers with

256 channels. Using only the voxel grid to model view-invariant diffused colors can converge in much less time, but the rendering quality is degraded. In NeRF-based scene reconstruction, spherical harmonic representation [9, 20], learnable basis [18], or deferred rendering [5] are also shown to be effective, but we leave the explorations for future work. We also compare the number of dimensions D of our feature grid, the number of layers, and the number of channels of the shallow MLP. Using a larger model leads to better rendering quality with the cost of more computation.

Colors rep. in fine stage			PSNR↑	Training minutes↓
implicit (shallow MLP)			26.57	6.2
explicit (diffused only)			32.15	**2.9**
hybrid				
D	MLP layers	MLP ch.		
12	1	128	35.11	7.0
12	2	128	35.54	7.3
12	3	128	35.70	7.8
12	2	64	35.35	7.0
12	2	192	35.71	7.7
24	2	128	**35.90**	8.8

Table B5. Different representations and hyperparameters for modeling the view-dependent color emissions.

Effect of the auxiliary losses. We show in Tab. B6 that incorporating the per-point rgb loss and the background entropy loss added to the main photometric loss can improve our results. The per-point rgb loss supervises the color emission of each query points directly instead of the accumulated color. The background entropy loss enforces the rendered background probability to concentrate on the background or foreground. We achieve the best results when adding the two auxiliary losses.

Per-point rgb loss	Background entropy loss	PSNR↑
		34.84
✓		35.37
	✓	35.53
✓	✓	**35.54**

Table B6. Effect of the auxiliary losses.

C. Main ablation studies details

In Tab. C1, we show detailed results for the ablation experiments presented in the main paper. We subsample two scenes for each dataset to conduct the main ablation experiments—*Materials* and *Mic* from Synthetic-NeRF dataset; *Robot* and *Lifestyle* from Synthetic-NSVF dataset; *Character* and *Statues* from BlendedMVS dataset, and *Ignatius* and *Truck* from Tanks and Temples dataset.

We show that the proposed post-activation significantly improves our quality. The low-density initialization is shown to be essential to our method and is an important hyperparameter. The view-count-based learning rate can slightly improve the results, and the PSNR without the view-count prior can degrade up to -1.22 in the worst case.

D. Additional training time details

D.1. Training time comparisons

Mip-NeRF [1] requires similar run-time as NeRF [12] but achieves much better quality. As a side benefit, Mip-NeRF can attain NeRF's PSNR in less time. To compare the convergence speed with Mip-NeRF, we use the code open-sourced by the authors (https://github.com/google/mipnerf). We re-train early-stopped Mip-NeRFs on our machine with only a single RTX2080 Ti GPU. The comparison is shown in Tab. D1. The early-stopped Mip-NeRF that is trained for 6 hours achieves an average PSNR of 30.85. On the other hand, our method is only optimized for about 15 minutes and achieves an average PSNR of 31.95.

D.2. The detailed training time of our model

We detail our training times on each scene and each stage in Tab. D2. The coarse stage optimization accounts for about $15-20\%$ of the overall training time. The per-scene training times are about less than 15 minutes, except for the *Tanks&Temples* [6], which takes just a few more minutes.

E. Per-scene analysis

E.1. Per-scene quantitative results

We report the per-scene quantitative comparisons in Tab. E1 for **Synthetic-NeRF** [12] dataset, Tab. E2 for **Synthetic-NSVF** [8] dataset, Tab. E3 for **BlendedMVS** [19] dataset, Tab. E4 for **Tanks&Temples** [6] dataset, and Tab. E5 for **DeepVoxels** [15] dataset. Our method achieves comparable results to most of the recent methods, except the Mip-NeRF [1] and JaxNeRF+ [2]. Moreover, all the methods after NeRF on the tables take quite a few hours to train for each scene, while our method only takes about 15 minutes per-scene optimization time as reported in Tab. D2.

E.2. Per-scene qualitative results

We provide more qualitative results in Fig. E2 for **Synthetic-NSVF** [8] dataset, Fig. E3 for **BlendedMVS** [19] dataset, and Fig. E5 for **DeepVoxels** [15] dataset. In Fig. E1, we compare our results with the results provided by JaxNeRF [2] and PlenOctrees [20] on **Synthetic-NeRF** [12] dataset. Both JaxNeRF and PlenOctrees are stronger baselines than the original NeRF [12]. In Fig. E4, we also compare our results with PlenOctrees on the **Tanks&Temples** [6] dataset. In the qualitative comparisons, there is no consistent superior method across all the scenes, which matches the results in quantitative comparisons.

Interp.	$\alpha^{(\text{init})(c)}$	View. lr.	Overall PSNR↑	Overall Δ_{avg}	Overall Δ_{worst}	Overall Δ_{best}	Synthetic-NeRF *Materials* PSNR↑	Synthetic-NeRF *Materials* Δ	Synthetic-NeRF *Mic* PSNR↑	Synthetic-NeRF *Mic* Δ	Synthetic-NSVF *Robot* PSNR↑	Synthetic-NSVF *Robot* Δ	Synthetic-NSVF *Lifestyle* PSNR↑	Synthetic-NSVF *Lifestyle* Δ	BlendedMVS *Character* PSNR↑	BlendedMVS *Character* Δ	BlendedMVS *Statues* PSNR↑	BlendedMVS *Statues* Δ	Tanks and Temples *Ignatius* PSNR↑	Tanks and Temples *Ignatius* Δ	Tanks and Temples *Truck* PSNR↑	Tanks and Temples *Truck* Δ
post-act.	10^{-6}	✓	**30.52**	-	-	-	**29.57**	-	33.20	-	**36.36**	-	**33.79**	-	30.31	-	**25.62**	-	28.16	-	**27.15**	
nearest	10^{-6}	✓	27.33	-3.19	-7.50	-0.34	28.17	-1.40	29.05	-4.15	28.86	-7.50	28.85	-4.94	27.28	-3.03	23.69	-1.93	27.82	-0.34	24.95	-2.20
pre-act.	10^{-6}	✓	29.58	-0.94	-2.88	-0.14	29.24	-0.33	32.43	-0.77	33.48	-2.88	31.85	-1.94	29.73	-0.58	25.04	-0.58	28.02	-0.14	26.86	-0.29
in-act.	10^{-6}	✓	29.28	-1.24	-3.30	0.00	27.34	-2.23	32.47	-0.73	33.06	-3.30	31.77	-2.02	29.74	-0.57	24.84	-0.78	28.16	0.00	26.89	-0.26
post-act.	-	✓	25.37	-5.15	-9.98	-1.61	27.40	-2.17	30.35	-2.85	26.43	-9.93	23.81	-9.98	24.70	-5.61	19.65	-5.97	26.55	-1.61	24.10	-3.05
	10^{-3}	✓	26.85	-3.67	-8.40	-0.23	28.96	-0.61	32.97	-0.23	29.09	-7.27	25.39	-8.40	25.12	-5.19	21.22	-4.40	27.23	-0.93	24.86	-2.29
	10^{-4}	✓	29.01	-1.51	-4.30	+0.05	29.35	-0.22	**33.23**	+0.03	32.06	-4.30	30.04	-3.75	28.47	-1.84	23.70	-1.92	**28.21**	+0.05	26.99	-0.16
	10^{-5}	✓	30.36	-0.16	-1.22	+0.03	**29.60**	+0.03	**33.22**	+0.02	36.32	-0.04	**33.75**	-0.04	30.32	+0.01	24.40	-1.22	**28.17**	+0.01	27.10	-0.05
	10^{-6}		30.35	-0.17	-1.22	+0.02	**29.57**	0.00	**33.22**	+0.02	36.33	-0.03	33.74	-0.05	**30.33**	+0.02	24.40	-1.22	28.06	-0.10	**27.12**	-0.03
	10^{-7}	✓	**30.43**	-0.09	-0.56	+0.20	**29.57**	0.00	33.15	-0.05	**36.56**	+0.20	33.50	-0.29	**30.40**	+0.09	**25.06**	-0.56	28.12	-0.04	27.07	-0.08

Table C1. We detail the per-scene results of different ablation setups presented in the main paper. We also show their difference (Δ) to our final setting (the first row). The Δ_{worst} highlights the worst-case degradation overall scenes, where most of the settings lead to more than 1 dB degradation in the worst case. The Δ_{best} shows the best case difference, where some settings can slightly improve the results on some scenes. We highlight the top 3 results of each column in **gold**, **silver**, and bronze.

Methods		Avg.	*Chair*	*Drums*	*Ficus*	*Hotdog*	*Lego*	*Materials*	*Mic*	*Ship*
Mip-NeRF [1] (early-stopped)	PSNR↑	30.85	32.25	24.95	30.66	35.56	32.92	29.28	32.61	28.56
	training minutes↓	361.2	361.1	363.0	360.1	359.0	363.7	360.7	359.3	363.0
Ours	PSNR↑	**31.95**	**34.09**	**25.44**	**32.78**	**36.74**	**34.64**	**29.57**	**33.20**	**29.13**
	training minutes↓	**14.2**	**12.5**	**12.0**	**13.8**	**15.5**	**13.2**	**15.4**	**11.0**	**20.2**

Table D1. We compare the training time with the early-stopped Mip-NeRF, which is a stronger baseline than NeRF. The training time is measured on our machine with only a single RTX2080 Ti GPU. Our method with about 15 minutes of training time outperforms the early-stopped Mip-NeRF with 6 hours of training by a large margin.

Stage	Avg.	*Chair*	*Drums*	*Ficus*	*Hotdog*	*Lego*	*Materials*	*Mic*	*Ship*
Coarse	2.3	2.5	2.4	2.0	2.3	2.3	2.1	2.3	2.3
Fine	11.9	10.1	9.7	11.7	13.2	10.9	13.3	8.7	17.9
All	14.2	12.5	12.0	13.8	15.5	13.2	15.4	11.0	20.2

(a) Training times (minutes) on the eight scenes of **Synthetic-NeRF** [12] dataset.

Stage	Avg.	*Wineholder*	*Steamtrain*	*Toad*	*Robot*	*Bike*	*Palace*	*Spaceship*	*Lifestyle*
Coarse	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5
Fine	10.7	9.6	12.8	9.8	9.5	10.1	11.2	12.1	10.1
All	13.2	12.1	15.3	12.3	12.0	12.6	13.7	14.6	12.6

(b) Training times (minutes) on the eight scenes of **Synthetic-NSVF** [8] dataset.

Stage	Avg.	*Jade*	*Fountain*	*Character*	*Statues*
Coarse	2.6	2.0	2.7	3.2	2.3
Fine	11.2	11.8	10.6	11.5	11.0
All	13.8	13.8	13.3	14.7	13.2

(c) Training times (minutes) on the four scenes of **BlendedMVS** [19] dataset.

Stage	Avg.	*Ignatius*	*Truck*	*Barn*	*Cate.*	*Family*
Coarse	3.6	3.4	3.6	3.8	3.6	3.4
Fine	14.2	12.4	14.3	18.2	14.1	11.9
All	17.7	15.7	17.8	22.0	17.7	15.3

(d) Training times (minutes) on the five scenes of **Tanks&Temples** [6] dataset.

Stage	Avg.	*Chair*	*Pedestal*	*Cube*	*Vase*
Coarse	2.4	2.5	2.5	2.1	2.5
Fine	11.9	12.8	11.6	11.1	12.2
All	14.3	15.2	14.1	13.2	14.7

(e) Training times (minutes) on the four scenes of **DeepVoxels** [15] dataset.

Table D2. We report the detail of our per-scene optimization time in minutes measured on our machine with a single RTX2080 Ti GPU.

Methods	Avg.	*Chair*	*Drums*	*Ficus*	*Hotdog*	*Lego*	*Materials*	*Mic*	*Ship*
PSNR↑									
SRN [16]	22.26	26.96	17.18	20.73	26.81	20.85	18.09	26.85	20.60
NV [10]	26.05	28.33	22.58	24.79	30.71	26.08	24.22	27.78	23.93
NeRF [12]	31.01	33.00	25.01	30.13	36.18	32.54	29.62	32.91	28.65
JaxNeRF [2]	31.69	34.08	25.03	30.43	36.92	33.28	29.91	34.53	29.36
JaxNeRF+ [2]	**33.00**	**35.35**	**25.65**	32.77	**37.58**	**35.35**	30.29	**36.52**	**30.48**
Mip-NeRF [1]	**33.09**	**35.14**	**25.48**	**33.29**	**37.48**	**35.70**	**30.71**	**36.51**	**30.41**
AutoInt [7]	25.55	25.60	20.78	22.47	32.33	25.09	25.90	28.10	24.15
FastNeRF [3]	29.97	32.32	23.75	27.79	34.72	32.28	28.89	31.77	27.69
SNeRG [5]	30.38	33.24	24.57	29.32	34.33	33.82	27.21	32.60	27.97
KiloNeRF [14]	31.00	-	-	-	-	-	-	-	-
PlenOctrees [20]	31.71	34.66	25.31	30.79	36.79	32.95	29.76	33.97	29.42
NSVF [8]	31.75	33.19	25.18	31.23	37.14	32.29	**32.68**	34.27	27.93
Ours	31.95	34.09	25.44	**32.78**	36.74	34.64	29.57	33.20	29.13
SSIM↑									
SRN [16]	0.846	0.910	0.766	0.849	0.923	0.809	0.808	0.947	0.757
NV [10]	0.893	0.916	0.873	0.910	0.944	0.880	0.888	0.946	0.784
NeRF [12]	0.947	0.967	0.925	0.964	0.974	0.961	0.949	0.980	0.856
JaxNeRF [2]	0.953	0.975	0.925	0.967	0.979	0.968	0.952	0.987	0.868
JaxNeRF+ [2]	**0.962**	**0.982**	**0.936**	**0.980**	**0.983**	**0.979**	**0.956**	**0.991**	**0.887**
Mip-NeRF [1]	**0.961**	**0.981**	0.932	**0.980**	**0.982**	**0.978**	**0.959**	**0.991**	0.882
AutoInt [7]	0.911	0.928	0.861	0.898	0.974	0.900	0.930	0.948	0.852
FastNeRF [3]	0.941	0.966	0.913	0.954	0.973	0.964	0.947	0.977	0.805
SNeRG [5]	0.950	0.975	0.929	0.967	0.971	0.973	0.938	0.982	0.865
KiloNeRF [14]	0.95	-	-	-	-	-	-	-	-
PlenOctrees [20]	0.958	**0.981**	**0.933**	0.970	**0.982**	0.971	0.955	0.987	**0.884**
NSVF [8]	0.953	0.968	0.931	0.973	0.980	0.960	0.973	0.987	0.854
Ours	0.957	0.977	0.930	0.978	0.980	0.976	0.951	0.983	0.879
LPIPS↓ (Vgg)									
SRN [16]	0.170	0.106	0.267	0.149	0.100	0.200	0.174	0.063	0.299
NV [10]	0.160	0.109	0.214	0.162	0.109	0.175	0.130	0.107	0.276
NeRF [12]	0.081	0.046	0.091	0.044	0.121	0.050	0.063	0.028	0.206
JaxNeRF [2]	0.068	0.035	0.085	0.038	0.079	0.040	0.060	0.019	0.185
Mip-NeRF [1]	**0.043**	**0.021**	**0.065**	**0.020**	**0.027**	**0.021**	**0.040**	**0.009**	**0.138**
PlenOctrees [20]	**0.053**	**0.022**	**0.076**	0.038	**0.032**	0.034	0.059	**0.017**	**0.144**
Ours	**0.053**	0.027	0.077	**0.024**	0.034	**0.028**	**0.058**	**0.017**	0.161
LPIPS↓ (Alex)									
NSVF [8]	**0.047**	**0.043**	**0.069**	**0.017**	**0.025**	**0.029**	**0.021**	**0.010**	**0.162**
Ours	**0.035**	**0.016**	**0.061**	**0.015**	**0.017**	**0.014**	**0.026**	**0.014**	**0.118**

Table E1. Quantitative results on each scene from the **Synthetic-NeRF** [12] dataset. We highlight the top 3 results of each column under each metric in **gold**, **silver**, and bronze.

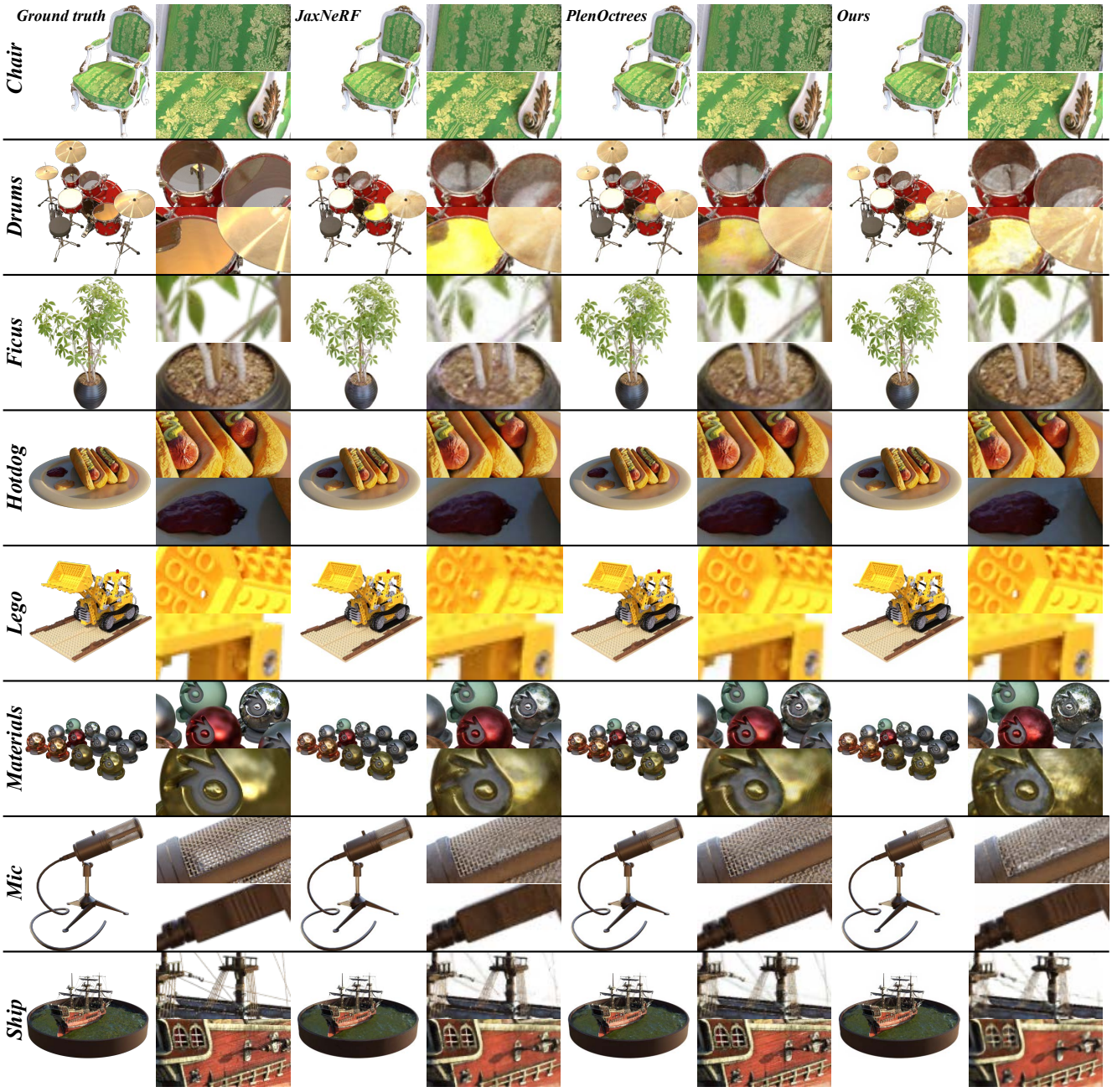


Figure E1. Qualitative comparisons on the on the **Synthetic-NeRF** [12] dataset. We manually resize, crop, and compress the images. JaxNeRF and PlenOctrees are both stronger baselines than NeRF. No method produces consistently better fine detail on all scenes. JaxNeRF recovers the shadow better in *Hotdog* and no blocking artifact in *Materials*; PlenOctrees shows better fine texture on *Mic* and *Ship*; our method reconstructs the fine detail of *Ficus* and *Lego* better.

Methods	Avg.	*Wineholder*	*Steamtrain*	*Toad*	*Robot*	*Bike*	*Palace*	*Spaceship*	*Lifestyle*
PSNR↑									
SRN [16]	24.33	20.74	25.49	25.36	22.27	23.76	24.45	27.99	24.58
NV [10]	25.83	21.32	25.31	24.63	24.74	26.65	26.38	29.90	27.68
NeRF [12]	30.81	28.23	30.84	29.42	28.69	31.77	31.76	34.66	31.08
KiloNeRF [14]	33.37	-	-	-	-	-	-	-	-
NSVF [8]	**35.13**	**32.04**	**35.13**	**33.25**	**35.24**	**37.75**	**34.05**	**39.00**	**34.60**
Ours	**35.08**	**30.26**	**36.56**	**33.10**	**36.36**	**38.33**	**34.49**	**37.71**	**33.79**
SSIM↑									
SRN [16]	0.882	0.850	0.923	0.822	0.904	0.926	0.792	0.945	0.892
NV [10]	0.892	0.828	0.900	0.813	0.927	0.943	0.826	0.956	0.941
NeRF [12]	0.952	0.920	0.966	0.920	0.960	0.970	0.950	0.980	0.946
KiloNeRF [14]	0.97	-	-	-	-	-	-	-	-
NSVF [8]	**0.979**	**0.965**	**0.986**	**0.968**	**0.988**	**0.991**	**0.969**	**0.991**	**0.971**
Ours	**0.975**	**0.949**	**0.989**	**0.966**	**0.992**	**0.991**	**0.962**	**0.988**	**0.965**
LPIPS↓ (Vgg)									
Ours	0.033	0.055	0.019	0.047	0.013	0.011	0.043	0.019	0.054
LPIPS↓ (Alex)									
SRN [16]	0.141	0.224	0.082	0.204	0.120	0.075	0.240	0.061	0.120
NV [10]	0.124	0.204	0.121	0.192	0.096	0.067	0.173	0.056	0.088
NeRF [12]	0.043	0.096	0.031	0.069	0.038	0.019	0.031	0.016	0.047
NSVF [8]	**0.015**	**0.020**	**0.010**	**0.032**	**0.007**	**0.004**	**0.018**	**0.006**	**0.020**
Ours	**0.019**	**0.038**	**0.010**	**0.030**	**0.005**	**0.004**	**0.027**	**0.009**	**0.027**

Table E2. Quantitative results on each scene from the **Synthetic-NSVF** [8] dataset. We highlight the top 3 results of each column under each metric in **gold**, **silver**, and bronze.

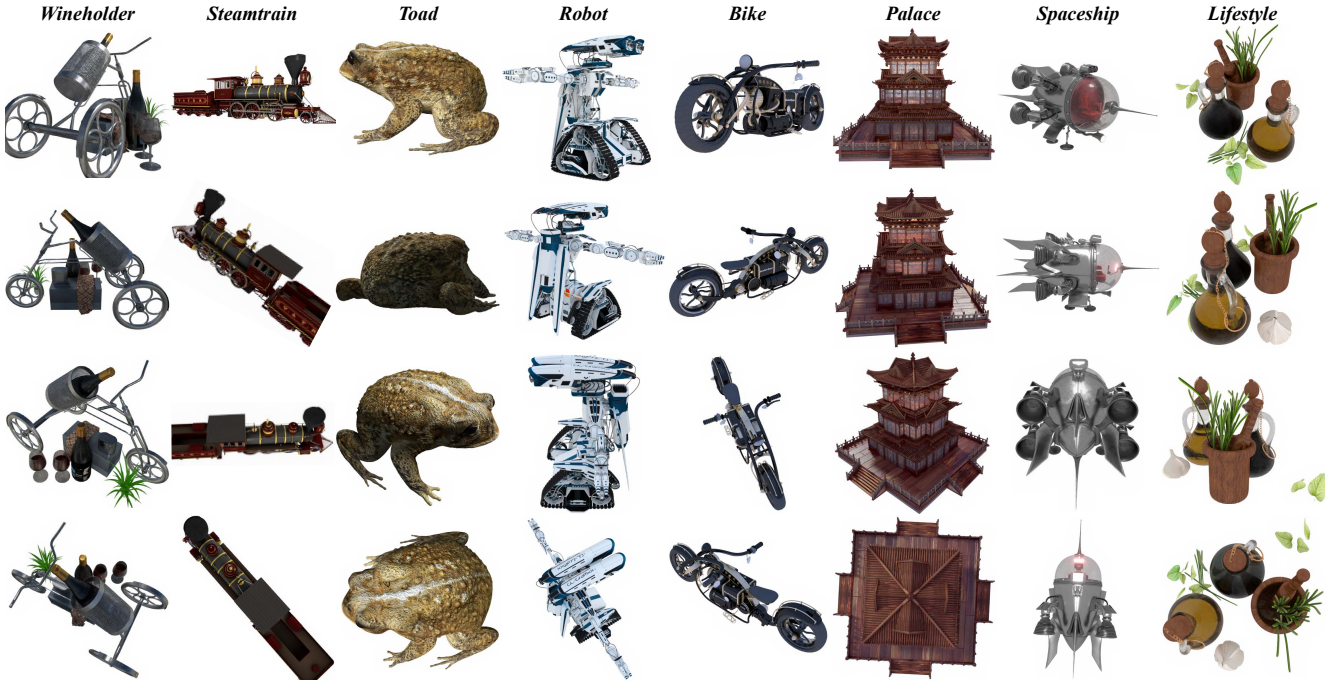


Figure E2. The synthesized novel views by our method on the **Synthetic-NSVF** [8] dataset. We manually resize, crop, and compress the images.

Methods	Avg.	*Jade*	*Fountain*	*Character*	*Statues*
PSNR↑					
SRN [16]	20.51	18.57	21.04	21.98	20.46
NV [10]	23.03	22.08	22.71	24.10	23.22
NeRF [12]	24.15	21.65	25.59	25.87	23.48
KiloNeRF [14]	**27.39**	-	-	-	-
NSVF [8]	26.89	**26.96**	**27.73**	**27.95**	**24.97**
Ours	**28.02**	**27.68**	**28.48**	**30.31**	**25.62**
SSIM↑					
SRN [16]	0.770	0.715	0.717	0.853	0.794
NV [10]	0.793	0.750	0.762	0.876	0.785
NeRF [12]	0.828	0.750	0.860	0.900	0.800
KiloNeRF [14]	**0.92**	-	-	-	
NSVF [8]	0.898	**0.901**	**0.913**	**0.921**	**0.858**
Ours	**0.922**	**0.914**	**0.926**	**0.963**	**0.883**
LPIPS↓ (Vgg)					
Ours	0.101	0.108	0.115	0.045	0.137
LPIPS↓ (Alex)					
SRN [16]	0.294	0.323	0.291	0.208	0.354
NV [10]	0.243	0.292	0.263	0.140	0.277
NeRF [12]	0.192	0.264	0.149	0.149	0.206
NSVF [8]	**0.114**	**0.094**	**0.113**	**0.074**	**0.171**
Ours	**0.075**	**0.075**	**0.086**	**0.029**	**0.110**

Table E3. Quantitative results on each scene from the **BlendedMVS** [19] dataset. We highlight the top 3 results of each column under each metric in **gold**, **silver**, and bronze.

Figure E3. The synthesized novel views by our method on the **BlendedMVS** [19] dataset. We manually resize, crop, and compress the images.

Methods	Avg.	*Ignatius*	*Truck*	*Barn*	*Caterpillar*	*Family*
PSNR↑						
SRN [16]	24.10	26.70	22.62	22.44	21.14	27.57
NV [10]	23.70	26.54	21.71	20.82	20.71	28.72
NeRF [12]	25.78	25.43	25.36	24.05	23.75	30.29
JaxNeRF [2]	27.94	27.95	26.66	**27.39**	25.24	32.47
KiloNeRF [14]	**28.41**	-	-	-	-	-
PlenOctrees [20]	27.99	**28.19**	26.83	26.80	25.29	32.85
NSVF [8]	**28.48**	27.91	**26.92**	**27.16**	**26.44**	**33.58**
Ours	**28.41**	**28.16**	**27.15**	27.01	**26.00**	**33.75**
SSIM↑						
SRN [16]	0.847	0.920	0.832	0.741	0.834	0.908
NV [10]	0.834	0.922	0.793	0.721	0.819	0.916
NeRF [12]	0.864	0.920	0.860	0.750	0.860	0.932
JaxNeRF [2]	0.904	0.940	0.896	**0.842**	0.892	0.951
KiloNeRF [14]	0.91	-	-	-	-	-
PlenOctrees [20]	**0.917**	**0.948**	**0.914**	**0.856**	**0.907**	**0.962**
NSVF [8]	0.901	0.930	0.895	0.823	0.900	0.954
Ours	**0.911**	**0.944**	**0.906**	0.838	**0.906**	**0.962**
LPIPS↓ (Vgg)						
JaxNeRF [2]	0.168	0.102	0.173	**0.286**	0.189	0.092
PlenOctrees [20]	**0.131**	**0.080**	**0.130**	**0.226**	**0.148**	**0.069**
Ours	**0.155**	**0.083**	**0.160**	0.294	**0.167**	**0.070**
LPIPS↓ (Alex)						
SRN [16]	0.251	0.128	0.266	0.448	0.278	0.134
NV [10]	0.260	0.117	0.312	0.479	0.280	0.111
NeRF [12]	0.198	0.111	0.192	0.395	0.196	0.098
NSVF [8]	**0.155**	**0.106**	**0.148**	**0.307**	**0.141**	**0.063**
Ours	**0.148**	**0.090**	**0.145**	**0.290**	**0.152**	**0.064**

Table E4. Quantitative results on each scene from the **Tanks&Temples** [6] dataset dataset. We highlight the top 3 results of each column under each metric in **gold**, **silver**, and bronze.

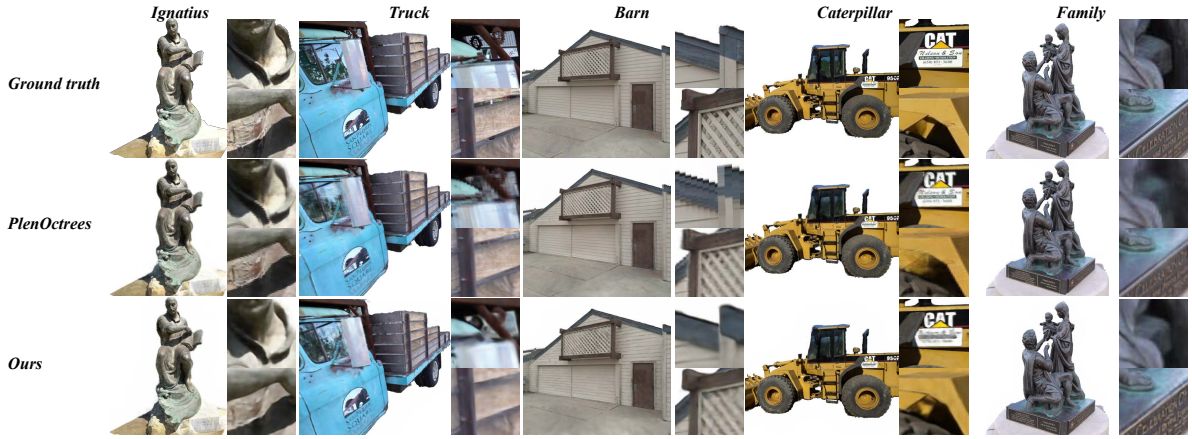


Figure E4. Qualitative comparisons on the **Tanks&Temples** [6] dataset. We manually resize, crop, and compress the images. Our quality is comparable to PlenOctrees, and no method produces consistent finer detail. Besides, we do not show blocking artifacts.

Methods	Avg.	*Chair*	*Pedestal*	*Cube*	*Vase*
PSNR↑					
Nearest Neighbor	20.94	20.69	21.49	18.32	23.26
NV [10]	29.62	35.15	36.47	26.48	20.39
DeepVoxels [15]	30.55	33.45	32.35	28.42	27.99
DeepVoxels++ [4]	37.31	40.87	38.93	36.51	32.91
SRN [16]	33.20	36.67	35.91	28.74	31.46
LLFF [11]	34.38	36.11	35.87	32.58	32.97
NeRF [12]	40.15	**42.65**	**41.44**	**39.19**	**37.32**
IBRNet [17]	**42.93**	-	-	-	-
Ours	**45.83**	**48.48**	**48.51**	**43.77**	**42.54**
SSIM↑					
Nearest Neighbor	0.89	0.94	0.87	0.83	0.92
NV [10]	0.929	0.980	0.963	0.916	0.857
DeepVoxels [15]	0.97	0.99	0.97	0.97	0.96
DeepVoxels++ [4]	0.99	0.99	0.98	0.99	0.98
SRN [16]	0.963	0.982	0.957	0.944	0.969
LLFF [11]	0.985	**0.992**	0.983	0.983	0.983
NeRF [12]	0.991	0.991	**0.986**	**0.996**	**0.992**
IBRNet [17]	**0.997**	-	-	-	-
Ours	**0.998**	**0.998**	**0.998**	**0.998**	**0.998**
LPIPS↓ (Vgg)					
SRN [16]	0.073	0.093	0.081	0.074	0.044
NV [10]	0.099	0.096	0.069	0.113	0.117
LLFF [11]	0.048	0.051	0.039	0.064	0.039
NeRF [12]	0.023	**0.047**	**0.024**	**0.006**	**0.017**
IBRNet [17]	**0.009**	-	-	-	-
Ours	**0.006**	**0.015**	**0.003**	**0.002**	**0.004**
LPIPS↓ (Alex)					
Ours	0.002	0.005	0.001	0.001	0.002

Table E5. Quantitative results on each scene from the **DeepVoxels** [15] dataset. We highlight the top 3 results of each column under each metric in **gold**, **silver**, and bronze.

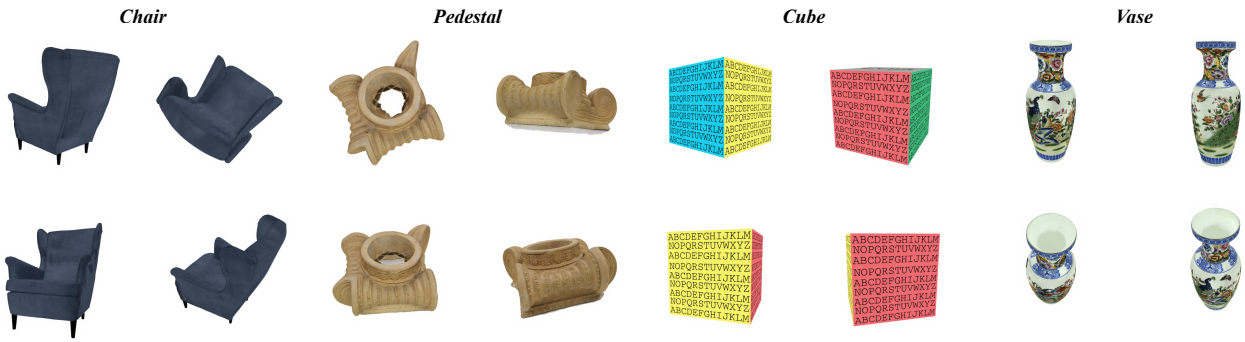


Figure E5. The synthesized novel views by our method on the **DeepVoxels** [15] dataset.

F. Derivation of low-density initialization

In this section, we derive the bias term b in the low-density initialization to make the activated alpha value very close to zero (*i.e.*, $\alpha^{(\text{init})(\text{c})} \approx 0$) at the beginning of our coarse stage optimization. More specifically, $\alpha^{(\text{init})(\text{c})}$ is a hyperparameter, and $\left(1-\alpha^{(\text{init})(\text{c})}\right)$ means the decay factor of the accumulated transmittance for a ray tracing forward a distance of the voxel size $s^{(\text{c})}$.

Below we omit the stage-specific superscript for simplicity. Let the accumulated distance of s be divided into N segments, each with a distance of δ_i, *i.e.*,

$$\sum_{i=1}^{N} \delta_i = s \ .$$

Recall that, in practice, we initialize all raw values $\ddot{\sigma}$ in $\boldsymbol{V}^{(\text{density})(\text{c})}$ to 0 and thus the initial activated densities σ_i are all equal to $\log\left(1+\exp(b)\right)$. Then, the decay factor of the accumulated transmittance can be written as

$$\begin{aligned}
\prod_{i=1}^{N}(1-\alpha_i) &= \prod_{i=1}^{N}\left((1-(1-\exp(-\sigma_i\delta_i)))\right) \\
&= \prod_{i=1}^{N}\exp(-\sigma_i\delta_i) \\
&= \prod_{i=1}^{N}\exp\Big(\log\left(1+\exp(b)\right)\cdot(-\delta_i)\Big) \\
&= \exp\left(\sum_{i=1}^{N}\Big(\log\left(1+\exp(b)\right)\cdot(-\delta_i)\Big)\right) \\
&= \exp\left(\log\left(1+\exp(b)\right)\cdot\sum_{i=1}^{N}(-\delta_i)\right) \\
&= \exp\Big(\log\left(1+\exp(b)\right)\cdot(-s)\Big) \\
&= \left(1+\exp(b)\right)^{-s} \ .
\end{aligned}$$

We want to find the b such that the decay factor is $(1-\alpha^{(\text{init})})$ for passing through a distance of the voxel size s. Therefore, we have

$$b = \log\left(\left(1-\alpha^{(\text{init})}\right)^{-\frac{1}{s}}-1\right) \ ,$$

which is the equation presented in our main paper to impose the prior of low-density initialization.

G. Derivation of post-activation

By expanding the post-activated trilinear interpolation, we have

$$\begin{aligned}
\alpha^{(\text{post})} &= 1-\exp\left(-\delta\log\left(1+\exp\left(\text{interp}\left(\boldsymbol{x},\boldsymbol{V}\right)\right)\right)\right) \\
&= 1-\left(1+\exp\left(\text{interp}\left(\boldsymbol{x},\boldsymbol{V}\right)\right)\right)^{-\delta} \ ,
\end{aligned} \tag{4}$$

where δ is a volume rendering related value and is treated as a constant in later derivation, and interp is the interpolation operation of a spatial point $\boldsymbol{x}$ in grid $\boldsymbol{V}$. The bias term b is omitted here for simplicity. The overall activation maps an interpolant ($\text{interp}\left(\boldsymbol{x},\boldsymbol{V}\right)\in\mathbb{R}$) into an alpha value ($\alpha^{(\text{post})}\in[0,1]$).

We want to prove that such a post-activation can produce sharp linear surface (decision boundary) in a single grid cell. Let first prove in the simplest 1D grid and then we generalize it to 2D grid. The case in 3D grid then can be proven easily using the derivations in 1D and 2D grid.

G.1. Derivation for a 1D grid cell

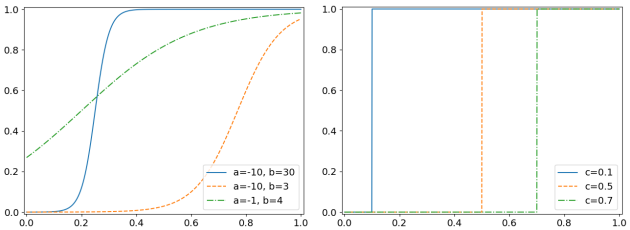
In 1D grid, $\boldsymbol{x}$ is a scalar. Without loss of generality, we assume $\boldsymbol{x}=0$ and $\boldsymbol{x}=1$ are the left and right bound of the 1D grid cell, respectively. Let $a,b\in\mathbb{R}$ be the grid values stored at the cell's left and right bound. Then Eq. (4) with 1D linear interpolation of the cell can be re-written for this specific case:

$$S(\boldsymbol{x};\ a,b) = 1-\left(1+\exp(a(1-\boldsymbol{x})+b\boldsymbol{x})\right)^{-\delta} \ . \tag{5}$$

Consider a target function $T(\boldsymbol{x};\ c)$ in the form of a shifted unit step function,

$$\begin{aligned}
T(\boldsymbol{x};\ c) &= \mathbb{1}(\boldsymbol{x}-c) \\
&= \begin{cases} 1\,, & \boldsymbol{x}>c\,, \\ 0\,, & \boldsymbol{x}\le c\,, \end{cases}
\end{aligned} \tag{6}$$

where $0<c<1$ is the position of the target linear surface (decision boundary) in the 1D grid cell. We only derive for the occupancy at right-hand side as the opposite direction can be trivially generalized. Visualization for S and T with some specific parameters is shown in Fig. G1.



(a) Three examples of $S(\boldsymbol{x};\ a,b)$ with $\delta=1$ (b) Three examples of $T(\boldsymbol{x};\ c)$

Figure G1. Visualization for some examples of S and T with different parameter settings.

We show that $S(\boldsymbol{x};\ a,b)$ can be made arbitrarily close to $T(\boldsymbol{x};\ c)$. Specifically, given any ϵ and Δ satisfying $0<\epsilon<1$ and $0<\Delta<\min(c,1-c)$, we can find the grid values a,b such that

$$|S(\boldsymbol{x};\ a,b)-T(\boldsymbol{x};\ c)|\le\epsilon,\ \forall|\boldsymbol{x}-c|\ge\Delta \ .$$

As the function S is a monotonically increasing function bounded in $[0, 1]$, the criterion can then be simplified into

$$1 - S(c + \Delta;\ a, b) \le \epsilon\ , \tag{7a}$$
$$S(c - \Delta;\ a, b) \le \epsilon\ . \tag{7b}$$

By expanding the inequality (7a), we get

$$\begin{aligned} &1 - S(c + \Delta;\ a, b) \\ &= 1 - \left(1 - (1 + \exp(a(1 - (c + \Delta)) + b(c + \Delta))^{-\delta}\right) \\ &= (1 + \exp(a(1 - (c + \Delta)) + b(c + \Delta))^{-\delta} \le \epsilon\ . \end{aligned}$$

We solve the inequality:

$$\begin{aligned} \left(\frac{1}{1 + \exp(a(1 - (c + \Delta)) + b(c + \Delta))}\right)^{\delta} &\le \epsilon \\ \frac{1}{1 + \exp(a(1 - (c + \Delta)) + b(c + \Delta))} &\le \epsilon^{1/\delta} \\ 1 + \exp(a(1 - (c + \Delta)) + b(c + \Delta)) &\ge \frac{1}{\epsilon^{1/\delta}} \\ \exp(a(1 - (c + \Delta)) + b(c + \Delta)) &\ge \frac{1}{\epsilon^{1/\delta}} - 1\ . \end{aligned}$$

Finally, we obtain

$$a(1 - (c + \Delta)) + b(c + \Delta) \ge \log\left(\frac{1}{\epsilon^{1/\delta}} - 1\right)\ . \tag{8}$$

By applying the same process to the inequality (7b), we get another inequality:

$$a(1 - (c - \Delta)) + b(c - \Delta) \le \log\left(\frac{1}{(1 - \epsilon)^{1/\delta}} - 1\right)\ . \tag{9}$$

There are infinite numbers of solutions satisfying the inequalities (8) and (9). Here, we introduce one more constraint to make the derivation and later extensions simpler:

$$S(c;\ a, b) = 0.5\ . \tag{10}$$

From Eq. (5) with this constraint we can then derive the linear relation between a, b:

$$b = a\frac{c - 1}{c} + \frac{\log(2^{\frac{1}{\delta}} - 1)}{c}\ . \tag{11}$$

Substitute b in the inequality (8), we get

$$\begin{aligned} &a(1 - (c + \Delta)) + b(c + \Delta) \\ &= a(1 - (c + \Delta)) + \left(a\frac{c - 1}{c} + \frac{\log(2^{\frac{1}{\delta}} - 1)}{c}\right)(c + \Delta) \\ &= a\frac{-\Delta}{c} + \log(2^{\frac{1}{\delta}} - 1)\frac{c + \Delta}{c} \ge \log\left(\frac{1}{\epsilon^{1/\delta}} - 1\right)\ . \end{aligned}$$

Similarly, substitute b in the inequality (9), we get

$$\begin{aligned} &a(1 - (c - \Delta)) + b(c - \Delta) \\ &= a(1 - (c - \Delta)) + \left(a\frac{c - 1}{c} + \frac{\log(2^{\frac{1}{\delta}} - 1)}{c}\right)(c - \Delta) \\ &= a\frac{\Delta}{c} + \log(2^{\frac{1}{\delta}} - 1)\frac{c - \Delta}{c} \le \log\left(\frac{1}{(1 - \epsilon)^{1/\delta}} - 1\right)\ . \end{aligned}$$

In summary, by adding the extra constraint (10), the inequality (8) and (9) become

$$\begin{cases} a \le & \log(2^{\frac{1}{\delta}} - 1)\frac{c + \Delta}{\Delta} - \log\left(\frac{1}{\epsilon^{1/\delta}} - 1\right)\frac{c}{\Delta} \\ a \le & \log\left(\frac{1}{(1 - \epsilon)^{1/\delta}} - 1\right)\frac{c}{\Delta} - \log(2^{\frac{1}{\delta}} - 1)\frac{c - \Delta}{\Delta}\ , \end{cases} \tag{12}$$

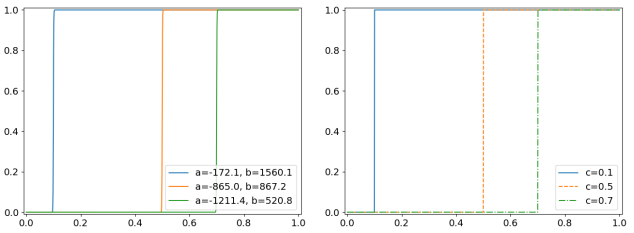
which defines the upper bound of a such that the conditions on the function S in (7a), (7b), and (10) can all be satisfied. The tighter upper bound is

$$a^{(\text{upper bound})} = \begin{cases} \log(2^{\frac{1}{\delta}} - 1)\frac{c + \Delta}{\Delta} - \log\left(\frac{1}{\epsilon^{1/\delta}} - 1\right)\frac{c}{\Delta}, & \text{if } \delta < 1 \\ \log\left(\frac{1}{(1 - \epsilon)^{1/\delta}} - 1\right)\frac{c}{\Delta} - \log(2^{\frac{1}{\delta}} - 1)\frac{c - \Delta}{\Delta}, & \text{otherwise} \end{cases}\ , \tag{13}$$

where the $\delta > 0$ is the pre-defined step size in volume rendering and we skip the detailed derivation of Eq. (13) here.

As a verification, we re-use the examples in Fig. G1b as the target functions and set the tolerance to $\epsilon = 10^{-4}, \Delta = 10^{-2}$ and the volume rendering related value to $\delta = 0.5$. We can directly find the grid values a, b using the derivations given above. We show the derived numbers and the resulting plot visualization in Fig. G2. It can be seen that the derived $S(\boldsymbol{x};\ a, b)$ can faithfully resemble the target $T(\boldsymbol{x};\ c)$.

$T(\boldsymbol{x};\ c)$	By Eq. (13)	By Eq. (11)
$c = 0.1$	$a \approx -172.1$	$b \approx 1560.1$
$c = 0.5$	$a \approx -865.0$	$b \approx 867.2$
$c = 0.7$	$a \approx -1211.4$	$b \approx 520.8$



(a) The derived $S(\boldsymbol{x};\ a, b)$. (b) Some specific $T(\boldsymbol{x};\ c)$

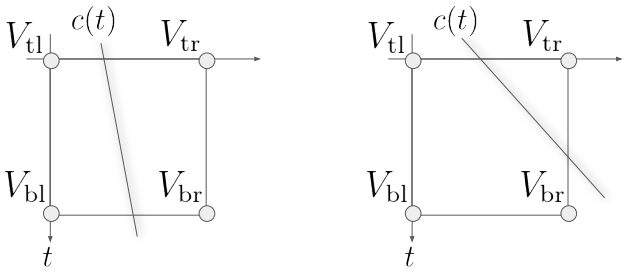
Figure G2. Using the derived upper bound (13) and Eq. (11) to directly find the grid values a, b that fit the target function $T(\boldsymbol{x};\ c)$.

G.2. Derivation for a 2D grid cell

We illustrate two situations that a linear boundary crossing a 2D grid cell in Fig. G3, where $V_{\text{tl}}, V_{\text{tr}}, V_{\text{bl}}, V_{\text{br}}$ are the top-left, top-right, bottom-left, and bottom-right values of a 2D grid cell and $c(t)$ is the linear boundary parameterized by the vertical position t. Let the coordinates of the top-left, top-right, bottom-left, and bottom-right corners be $(0, 0)$, $(1, 0)$, $(0, 1)$, and $(1, 1)$, so the top and bottom edges of the grid cell are on the horizontal lines $t = 0$ and $t = 1$, respectively. Without loss of generality, we assume the linear boundary always intersects the top edge of the 2D grid cell (*i.e.*, $0 < c(0) < 1$) and the left-hand-side $[0, c(0))$ of the boundary is free space. We can generalize to other cases via rotation and flipping, or by negating the grid values. We consider the 2D target function T with respect to the decision boundary inside the grid cell as a 2D unit step function, and parameterize it by the vertical coordinate t so that on each horizontal scan line the target function can be expressed as

$$T(\boldsymbol{x}(t);\, c(t)) = \mathbb{1}(\boldsymbol{x}(t) - c(t)) = \begin{cases} 1\,, & \boldsymbol{x}(t) > c(t)\,, \\ 0\,, & \boldsymbol{x}(t) \le c(t)\,. \end{cases} \tag{14}$$

The goal here is to show that, by choosing suitable values for $V_{\text{tl}}, V_{\text{tr}}, V_{\text{bl}}, V_{\text{br}}$, we are able to approximate the target function closely enough within the required tolerance using our post-activation scheme.



(a) **Case I:** The boundary intersects the top and bottom edge of the grid.

(b) **Case II:** The boundary intersects the top and right edge of the grid.

Figure G3. Two cases of a linear boundary $c(t)$ crossing a 2D grid cell, where $V_{\text{lt}}, V_{\text{rt}}, V_{\text{lb}}, V_{\text{rb}}$ are the grid values of the cell. Without loss of generality, we assume the linear boundary always intersects the top edge of the grid cell, *i.e.*, $0 < c(0) < 1$.

Case I: $0 < c(0) < 1$ **and** $0 < c(1) < 1$**.** An example is illustrated in Fig. G3a, where the linear boundary intersects the top edge and the bottom edge of a grid cell. The linear boundary c is a linear function of t defined by $c(t) = (1 - t) \cdot c(0) + t \cdot c(1)$. Based on the results we have derived for 1D, to prove this 2D case we only need to show that Eq. (11) and Eq. (13) are linear function of c, so that once we use the two equations to determine $V_{\text{tl}}, V_{\text{tr}}$ for approximating $c(0)$ and $V_{\text{bl}}, V_{\text{br}}$ for approximating $c(1)$, we can readily recover $c(t)$ from $c(0)$ and $c(1)$ on all horizontal scan lines $0 \le t \le 1$. That is, the approximation criteria in Eq. (7a) and Eq. (7b), where the target surface position is $c = c(t)$, are automatically satisfied by

$$a = a(t) = V_{\text{tl}}(1 - t) + V_{\text{bl}}t\,,$$
$$b = b(t) = V_{\text{tr}}(1 - t) + V_{\text{br}}t\,.$$

Eq. (13) is trivially a linear function of c given δ. By substituting a in Eq. (11) with $a^{(\text{upper bound})}$ from Eq. (13), we get

$$b = \log(2^{\frac{1}{\delta}} - 1)\frac{c + \Delta - 1}{\Delta} - \log\left(\frac{1}{\epsilon^{1/\delta}} - 1\right)\frac{c - 1}{\Delta}$$

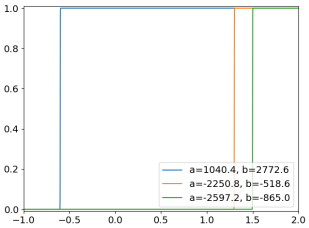
when $0 < \delta < 1$, and

$$b = \log\left(\frac{1}{(1 - \epsilon)^{1/\delta}} - 1\right)\frac{c - 1}{\Delta} - \log(2^{\frac{1}{\delta}} - 1)\frac{c - \Delta - 1}{\Delta}$$

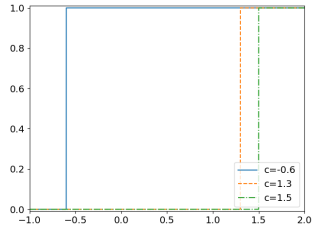
when $1 \le \delta$. Thus, b is also a linear function of c given δ.

Case II: $0 < c(0) < 1$ **and** $1 < c(1)$**.** We assume $0 < c < 1$ in the target function in Eq. (6), but it finally turns out that the derivations in Sec. G.1 can trivially generalize to $1 < c$. Actually, the derivations also work for $c < 0$ as long as we modify the signs in inequality (12) to '$\ge$', and the upper bound in Eq. (13) becomes lower bound. We verify the results with some examples shown in Fig. G4.

$T(\boldsymbol{x};\, c)$	By Eq. (13)	By Eq. (11)
$c = -0.6$	$a \approx 1040.4$	$b \approx 2772.6$
$c = 1.3$	$a \approx -2250.8$	$b \approx -518.6$
$c = 1.5$	$a \approx -2597.2$	$b \approx -865.0$



(a) The derived $S(\boldsymbol{x};\, a, b)$.



(b) Some specific $T(\boldsymbol{x};\, c)$.

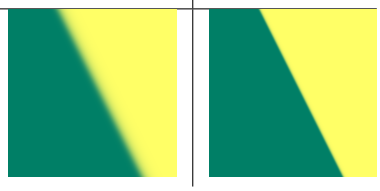
Figure G4. The derivation in Sec. G.1 is also applicable to extrapolation ($c < 0$ or $c > 1$) with minor modifications.

In summary, the derivation in Sec. G.1 also works for extrapolation with minor modifications and the derivation in **Case I** is directly applicable for **Case II**.

As a verification, we construct two examples in Fig. G5. The volume rendering step size is set to $\delta = 0.5$, the error bound is set to $\epsilon = 10^{-4}$, and we show two results with tolerance $\Delta = 0.20$ and $\Delta = 0.05$. For each target boundary, we first evaluate $c(0)$ and $c(1)$. The values of $V_{\text{tl}}, V_{\text{bl}}$ can then be derived from Eq. (13) and the values of $V_{\text{tr}}, V_{\text{br}}$ can be derived from Eq. (11).

(a) Target boundary $c(t) = 0.5 \cdot t + 0.3$.

ϵ	10^{-4}	10^{-4}
Δ	0.20	0.05
V_{tl}	-24.9	-518.6
V_{tr}	61.7	1213.6
V_{bl}	-68.2	-1384.7
V_{br}	18.4	347.5



(b) Target boundary $c(t) = 1.1 \cdot t + 0.2$.

ϵ	10^{-4}	10^{-4}
Δ	0.20	0.05
V_{tl}	-16.2	-345.3
V_{tr}	70.4	1386.9
V_{bl}	-111.5	-2250.8
V_{br}	-24.9	-518.6

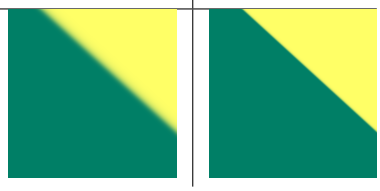


Figure G5. Using the extended 2D derivation to directly find the grid values $V_{\text{tl}}, V_{\text{tr}}, V_{\text{bl}}, V_{\text{br}}$ that fit the target boundary $c(t)$ under the required error bound ϵ and tolerance Δ.

G.3. Derivation for a 3D grid cell

Similar to the proof for 2D in Sec. G.2, we can assume that the 3D linear surface $c(t, u)$ intersects the top face of a 3D grid cell, as illustrated in Fig. G6. Without loss of generality, we assume the surface $c(t, u)$ intersects the horizontal planes $u = 0$ and $u = 1$, and the left-hand-side of the surface,

$$\{[t, u, v] \mid 0 \le t \le 1, 0 \le u \le 1, 0 \le v \le s(t, u)\} \ ,$$

is free space. The linear surface is $c(t, u) = (1-u) \cdot c(t, 0) + u \cdot c(t, 1)$. We can use the results in Sec. G.2 to determine the grid values of the top four corners with $c(t, 0)$ and the values of the bottom four corners with $c(t, 1)$. The linear boundary at every horizontal slice $0 \le u \le 1$ can be automatically satisfied, as we have shown in Sec. G.2.

G.4. Future extensions

Beyond linear surface (decision boundary). We only prove that the alpha field by post-activated interpolation can be arbitrarily close to a linear surface. We show in Fig. G7 that we can further tune the tolerances at the top edge and bottom edge of a grid to produce sharp and non-linear surfaces.

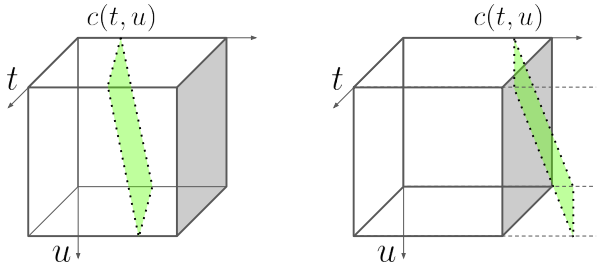


Figure G6. A linear surface $c(t, u)$ crossing a 3D grid cells. Without loss of generality, we assume the surface intersects with two horizontals planes, $u = 0$ and $u = 1$, which the top face and the bottom face of a 3D grid cell aligned with.

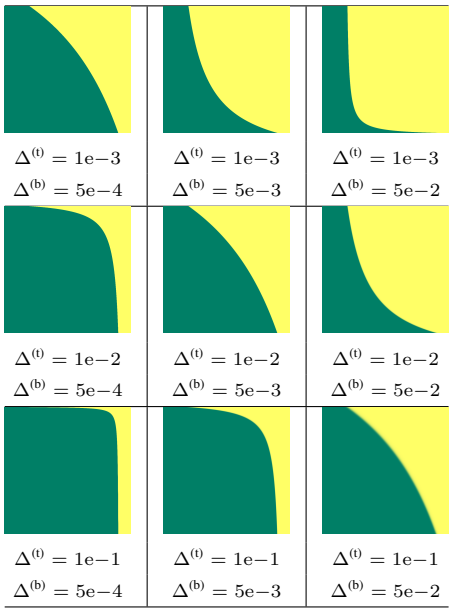


Figure G7. We fix $c(0) = 0.2, c(1) = 0.9, \delta = 0.5, \epsilon = 10^{-4}$ but use different tolerances for the top edge ($\Delta^{(t)}$) and the bottom edge ($\Delta^{(b)}$). Following the same procedure as in Sec. G.2 to determine the grid values, we can obtain a sharp non-linear surface.

Extending the proof or the capability of post-activation in future work could be helpful to geometry modeling.

Closed form solution when 3D available. In this work, we only consider the same input setup as NeRF, where only 2D observations and camera poses are available. In cases that the 3D model of the scene is available, an algorithm to convert the 3D model to our post-activated density voxel grid can be helpful. Our representation is directly compatible with gradient-based optimization and volume rendering to support follow-up applications, while 3D in other formats like mesh or point cloud may need more effort. We believe our derivations are useful for future work to develop a closed-form solution to convert the 3D in other formats into our representation.

References

[1] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, 2021. 2, 3, 4, 5

[2] Boyang Deng, Jonathan T. Barron, and Pratul P. Srinivasan. JaxNeRF: an efficient JAX implementation of NeRF, 2020. 3, 5, 9

[3] Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien P. C. Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *ICCV*, 2021. 5

[4] Tong He, John P. Collomosse, Hailin Jin, and Stefano Soatto. Deepvoxels++: Enhancing the fidelity of novel view synthesis from 3d voxel embeddings. In Hiroshi Ishikawa, Cheng-Lin Liu, Tomás Pajdla, and Jianbo Shi, editors, *ACCV*, 2020. 10

[5] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul E. Debevec. Baking neural radiance fields for real-time view synthesis. In *ICCV*, 2021. 3, 5

[6] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: benchmarking large-scale scene reconstruction. *ACM Trans. Graph.*, 2017. 3, 4, 9

[7] David B. Lindell, Julien N. P. Martel, and Gordon Wetzstein. Autoint: Automatic integration for fast neural volume rendering. In *CVPR*, 2021. 5

[8] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. In *NeurIPS*, 2020. 3, 4, 5, 7, 8, 9

[9] Yuan Liu, Sida Peng, Lingjie Liu, Qianqian Wang, Peng Wang, Christian Theobalt, Xiaowei Zhou, and Wenping Wang. Neural rays for occlusion-aware image-based rendering. *arxiv CS.CV 2107.13421*, 2021. 3

[10] Stephen Lombardi, Tomas Simon, Jason M. Saragih, Gabriel Schwartz, Andreas M. Lehrmann, and Yaser Sheikh. Neural volumes: learning dynamic renderable volumes from images. *ACM Trans. Graph.*, 2019. 5, 7, 8, 9, 10

[11] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: practical view synthesis with prescriptive sampling guidelines. *ACM Trans. Graph.*, 2019. 10

[12] 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. 2, 3, 4, 5, 6, 7, 8, 9, 10

[13] Michael Oechsle, Songyou Peng, and Andreas Geiger. UNISURF: unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *ICCV*, 2021. 2

[14] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *ICCV*, 2021. 5, 7, 8, 9

[15] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. Deepvoxels: Learning persistent 3d feature embeddings. In *CVPR*, 2019. 3, 4, 10

[16] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *NeurIPS*, 2019. 5, 7, 8, 9, 10

[17] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P. Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas A. Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *CVPR*, 2021. 10

[18] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. Nex: Real-time view synthesis with neural basis expansion. In *CVPR*, 2021. 3

[19] Yao Yao, Zixin Luo, Shiwei Li, Jingyang Zhang, Yufan Ren, Lei Zhou, Tian Fang, and Long Quan. Blendedmvs: A large-scale dataset for generalized multi-view stereo networks. In *CVPR*, 2020. 3, 4, 8

[20] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021. 3, 5, 9