NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis: Supplementary Materials

This document contains additional implementation details for our method and baseline methods used for comparison, as well as a more detailed breakdown of quantitative results presented in the main paper. Please view our included supplementary video for a brief overview of our method, convincing comparisons of rendered novel view paths, and additional qualitative results.

1 Additional Implementation Details

Network Architecture Fig. 1 details our simple fully-connected architecture.

Volume Bounds Our method renders views by querying the neural radiance field representation at continuous 5D coordinates along camera rays. For experiments with synthetic images, we scale the scene so that it lies within a cube of side length 2 centered at the origin, and only query the representation within this bounding volume. Our dataset of real images contains content that can exist anywhere between the closest point and infinity, so we use normalized device coordinates to map the depth range of these points into $[-1, 1]$. This shifts all the ray origins to the near plane of the scene, maps the perspective rays of the camera to parallel rays in the transformed volume, and uses disparity (inverse depth) instead of metric depth, so all coordinates are now bounded.

Training Details For real scene data, we regularize our network by adding random Gaussian noise with zero mean and unit variance to the output σ values (before passing them through the ReLU) during optimization, finding that this slightly improves visual performance for rendering novel views. We implement our model in Tensorflow [1].

Rendering Details To render new views at test time, we sample 64 points per ray through the coarse network and $64 + 128 = 192$ points per ray through the fine network, for a total of 256 network queries per ray. Our realistic synthetic dataset requires 640k rays per image, and our real scenes require 762k rays per image, resulting in between 150 and 200 million network queries per rendered image. On an NVIDIA V100, this takes approximately 30 seconds per frame.

2 Additional Baseline Method Details

Neural Volumes (NV) *[2]* We use the NV code open-sourced by the authors at https://github.com/facebookresearch/neuralvolumes and follow their procedure for training on a single scene without time dependence.

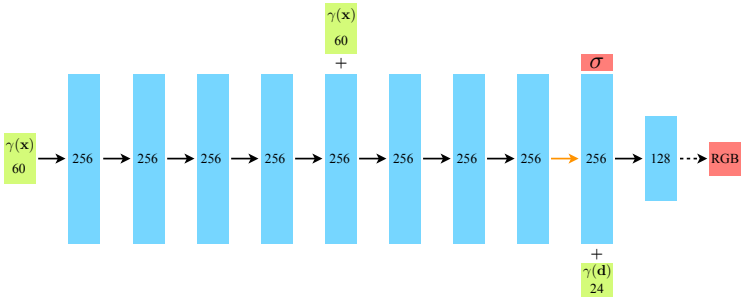

Fig. 1: A visualization of our fully-connected network architecture. Input vectors are shown in green, intermediate hidden layers are shown in blue, output vectors are shown in red, and the number inside each block signifies the vector's dimension. All layers are standard fully-connected layers, black arrows indicate layers with ReLU activations, orange arrows indicate layers with no activation, dashed black arrows indicate layers with sigmoid activation, and "+" denotes vector concatenation. The positional encoding of the input location ($\gamma(\mathbf{x})$) is passed through 8 fully-connected ReLU layers, each with 256 channels. We follow the DeepSDF [4] architecture and include a skip connection that concatenates this input to the fifth layer's activation. An additional layer outputs the volume density σ (which is rectified using a ReLU to ensure that the output volume density is nonnegative) and a 256-dimensional feature vector. This feature vector is concatenated with the positional encoding of the input viewing direction ($\gamma(\mathbf{d})$), and is processed by an additional fully-connected ReLU layer with 128 channels. A final layer (with a sigmoid activation) outputs the emitted RGB radiance at position $\mathbf{x}$, as viewed by a ray with direction $\mathbf{d}$.

Scene Representation Networks (SRN) *[6]* We use the SRN code open-sourced by the authors at https://github.com/vsitzmann/scene-representation-networks and follow their procedure for training on a single scene.

Local Light Field Fusion (LLFF) *[3]* We use the pretrained LLFF model open-sourced by the authors at https://github.com/Fyusion/LLFF.

Quantitative Comparisons The SRN implementation published by the authors requires a significant amount of GPU memory, and is limited to an image resolution of 512×512 pixels even when parallelized across 4 NVIDIA V100 GPUs. We compute quantitative metrics for SRN at 512×512 pixels for our synthetic datasets and 504×376 pixels for the real datasets, in comparison to 800×800 and 1008×752 respectively for the other methods that can be run at higher resolutions.

3 NDC ray space derivation

We reconstruct real scenes with "forward facing" captures in the normalized device coordinate (NDC) space that is commonly used as part of the triangle rasterization pipeline. This space is convenient because it preserves parallel lines while converting the z axis (camera axis) to be linear in disparity.

Here we derive the transformation which is applied to rays to map them from camera space to NDC space. The standard 3D perspective projection matrix for homogeneous coordinates is:

$$M = \begin{pmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{1}$$

where n, f are the near and far clipping planes and r and t are the right and top bounds of the scene at the near clipping plane. (Note that this is in the convention where the camera is looking in the $-z$ direction.) To project a homogeneous point $(x, y, z, 1)^\top$, we left-multiply by M and then divide by the fourth coordinate:

$$\begin{pmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{n}{r}x \\ \frac{n}{t}y \\ \frac{-(f+n)}{f-n}z - \frac{-2fn}{f-n} \\ -z \end{pmatrix} \tag{2}$$

$$\text{project} \rightarrow \begin{pmatrix} \frac{n}{r}\frac{x}{-z} \\ \frac{n}{t}\frac{y}{-z} \\ \frac{(f+n)}{f-n} - \frac{2fn}{f-n}\frac{1}{-z} \end{pmatrix} \tag{3}$$

The projected point is now in normalized device coordinate (NDC) space, where the original viewing frustum has been mapped to the cube $[-1, 1]^3$.

Our goal is to take a ray $\mathbf{o} + t\mathbf{d}$ and calculate a ray origin $\mathbf{o}'$ and direction $\mathbf{d}'$ in NDC space such that for every t, there exists a new t' for which $\pi(\mathbf{o} + t\mathbf{d}) = \mathbf{o}' + t'\mathbf{d}'$ (where π is projection using the above matrix). In other words, the projection of the original ray and the NDC space ray trace out the same points (but not necessarily at the same rate).

Let us rewrite the projected point from Eqn. 3 as $(a_x x/z, a_y y/z, a_z + b_z/z)^\top$. The components of the new origin $\mathbf{o}'$ and direction $\mathbf{d}'$ must satisfy:

$$\begin{pmatrix} a_x \frac{o_x + td_x}{o_z + td_z} \\ a_y \frac{o_y + td_y}{o_z + td_z} \\ a_z + \frac{b_z}{o_z + td_z} \end{pmatrix} = \begin{pmatrix} o'_x + t'd'_x \\ o'_y + t'd'_y \\ o'_z + t'd'_z \end{pmatrix}. \tag{4}$$

To eliminate a degree of freedom, we decide that $t' = 0$ and $t = 0$ should map to the same point. Substituting $t = 0$ and $t' = 0$ Eqn. 4 directly gives our NDC

space origin $\mathbf{o}'$:

$$\mathbf{o}' = \begin{pmatrix} o'_x \\ o'_y \\ o'_z \end{pmatrix} = \begin{pmatrix} a_x \frac{o_x}{o_z} \\ a_y \frac{o_y}{o_z} \\ a_z + \frac{b_z}{o_z} \end{pmatrix} = \pi(\mathbf{o})\,. \tag{5}$$

This is exactly the projection $\pi(\mathbf{o})$ of the original ray's origin. By substituting this back into Eqn. 4 for arbitrary t, we can determine the values of t' and $\mathbf{d}'$:

$$\begin{pmatrix} t'd'_x \\ t'd'_y \\ t'd'_z \end{pmatrix} = \begin{pmatrix} a_x \frac{o_x+td_x}{o_z+td_z} - a_x \frac{o_x}{o_z} \\ a_y \frac{o_y+td_y}{o_z+td_z} - a_y \frac{o_y}{o_z} \\ a_z + \frac{b_z}{o_z+td_z} - a_z - \frac{b_z}{o_z} \end{pmatrix} \tag{6}$$

$$= \begin{pmatrix} a_x \frac{o_z(o_x+td_x)-o_x(o_z+td_z)}{(o_z+td_z)o_z} \\ a_y \frac{o_z(o_y+td_y)-o_y(o_z+td_z)}{(o_z+td_z)o_z} \\ b_z \frac{o_z-(o_z+td_z)}{(o_z+td_z)o_z} \end{pmatrix} \tag{7}$$

$$= \begin{pmatrix} a_x \frac{td_z}{o_z+td_z}\left(\frac{d_x}{d_z} - \frac{o_x}{o_z}\right) \\ a_y \frac{td_z}{o_z+td_z}\left(\frac{d_y}{d_z} - \frac{o_y}{o_z}\right) \\ -b_z \frac{td_z}{o_z+td_z}\frac{1}{o_z} \end{pmatrix} \tag{8}$$

Factoring out a common expression that depends only on t gives us:

$$t' = \frac{td_z}{o_z + td_z} = 1 - \frac{o_z}{o_z + td_z} \tag{9}$$

$$\mathbf{d}' = \begin{pmatrix} a_x\left(\frac{d_x}{d_z} - \frac{o_x}{o_z}\right) \\ a_y\left(\frac{d_y}{d_z} - \frac{o_y}{o_z}\right) \\ -b_z \frac{1}{o_z} \end{pmatrix}. \tag{10}$$

Note that, as desired, $t' = 0$ when $t = 0$. Additionally, we see that $t' \to 1$ as $t \to \infty$. Going back to the original projection matrix, our constants are:

$$a_x = -\frac{n}{r} \tag{11}$$

$$a_y = -\frac{n}{t} \tag{12}$$

$$a_z = \frac{f+n}{f-n} \tag{13}$$

$$b_z = \frac{2fn}{f-n} \tag{14}$$

Using the standard pinhole camera model, we can reparameterize as:

$$a_x = -\frac{f_{cam}}{W/2} \tag{15}$$

$$a_y = -\frac{f_{cam}}{H/2} \tag{16}$$

where W and H are the width and height of the image in pixels and f_{cam} is the focal length of the camera.

In our real forward facing captures, we assume that the far scene bound is infinity (this costs us very little since NDC uses the z dimension to represent *inverse* depth, i.e., disparity). In this limit the z constants simplify to:

$$a_z = 1 \tag{17}$$

$$b_z = 2n\,. \tag{18}$$

Combining everything together:

$$\mathbf{o}' = \begin{pmatrix} -\frac{f_{cam}}{W/2}\frac{o_x}{o_z} \\ -\frac{f_{cam}}{H/2}\frac{o_y}{o_z} \\ 1 + \frac{2n}{o_z} \end{pmatrix} \tag{19}$$

$$\mathbf{d}' = \begin{pmatrix} -\frac{f_{cam}}{W/2}\left(\frac{d_x}{d_z} - \frac{o_x}{o_z}\right) \\ -\frac{f_{cam}}{H/2}\left(\frac{d_y}{d_z} - \frac{o_y}{o_z}\right) \\ -2n\frac{1}{o_z} \end{pmatrix}. \tag{20}$$

One final detail in our implementation: we shift $\mathbf{o}$ to the ray's intersection with the near plane at $z = -n$ (before this NDC conversion) by taking $\mathbf{o}_n = \mathbf{o} + t_n\mathbf{d}$ for $t_n = -(n + o_z)/d_z$. Once we convert to the NDC ray, this allows us to simply sample t' linearly from 0 to 1 in order to get a linear sampling in disparity from n to ∞ in the original space.

4 Additional Results

Per-scene breakdown Tables 1, 2, 3, and 4 include a breakdown of the quantitative results presented in the main paper into per-scene metrics. The per-scene breakdown is consistent with the aggregate quantitative metrics presented in the paper, where our method quantitatively outperforms all baselines. Although LLFF achieves slightly better LPIPS metrics, we urge readers to view our supplementary video where our method achieves better multiview consistency and produces fewer artifacts than all baselines.

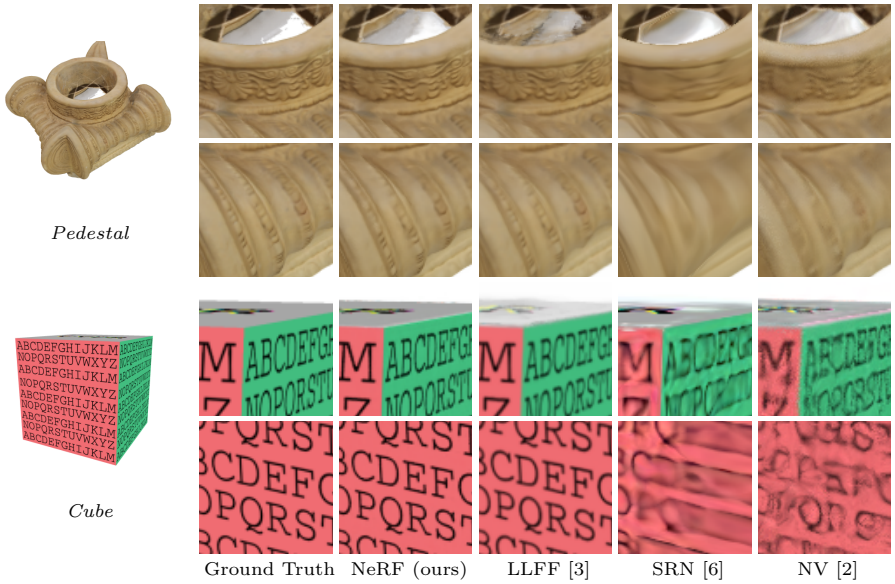

Fig. 2: Comparisons on test-set views for scenes from the DeepVoxels [5] synthetic dataset. The objects in this dataset have simple geometry and perfectly diffuse reflectance. Because of the large number of input images (479 views) and simplicity of the rendered objects, both our method and LLFF [3] perform nearly perfectly on this data. LLFF still occasionally presents artifacts when interpolating between its 3D volumes, as in the top inset for each object. SRN [6] and NV [2] do not have the representational power to render fine details.

	PSNR↑				SSIM↑				LPIPS↓			
	Chair	Pedestal	Cube	Vase	Chair	Pedestal	Cube	Vase	Chair	Pedestal	Cube	Vase
DeepVoxels [5]	33.45	32.35	28.42	27.99	0.99	0.97	0.97	0.96	–	–	–	–
SRN [6]	36.67	35.91	28.74	31.46	0.982	0.957	0.944	0.969	0.093	0.081	0.074	0.044
NV [2]	35.15	36.47	26.48	20.39	0.980	0.963	0.916	0.857	0.096	0.069	0.113	0.117
LLFF [3]	36.11	35.87	32.58	32.97	**0.992**	0.983	0.983	0.983	0.051	0.039	0.064	0.039
Ours	**42.65**	**41.44**	**39.19**	**37.32**	0.991	**0.986**	**0.996**	**0.992**	**0.047**	**0.024**	**0.006**	**0.017**

Table 1: Per-scene quantitative results from the DeepVoxels [5] dataset. The "scenes" in this dataset are all diffuse objects with simple geometry, rendered from texture-mapped meshes captured by a 3D scanner. The metrics for the DeepVoxels method are taken directly from their paper, which does not report LPIPS and only reports two significant figures for SSIM.

	PSNR↑							
	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship
SRN [6]	26.96	17.18	20.73	26.81	20.85	18.09	26.85	20.60
NV [2]	28.33	22.58	24.79	30.71	26.08	24.22	27.78	23.93
LLFF [3]	28.72	21.13	21.79	31.41	24.54	20.72	27.48	23.22
Ours	**33.00**	**25.01**	**30.13**	**36.18**	**32.54**	**29.62**	**32.91**	**28.65**

	SSIM↑							
	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship
SRN [6]	0.910	0.766	0.849	0.923	0.809	0.808	0.947	0.757
NV [2]	0.916	0.873	0.910	0.944	0.880	0.888	0.946	0.784
LLFF [3]	0.948	0.890	0.896	0.965	0.911	0.890	0.964	0.823
Ours	**0.967**	**0.925**	**0.964**	**0.974**	**0.961**	**0.949**	**0.980**	**0.856**

	LPIPS↓							
	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship
SRN [6]	0.106	0.267	0.149	0.100	0.200	0.174	0.063	0.299
NV [2]	0.109	0.214	0.162	0.109	0.175	0.130	0.107	0.276
LLFF [3]	0.064	0.126	0.130	**0.061**	0.110	0.117	0.084	0.218
Ours	**0.046**	**0.091**	**0.044**	0.121	**0.050**	**0.063**	**0.028**	**0.206**

Table 2: Per-scene quantitative results from our realistic synthetic dataset. The "scenes" in this dataset are all objects with more complex gometry and non-Lambertian materials, rendered using Blender's Cycles pathtracer.

	PSNR↑							
	Room	Fern	Leaves	Fortress	Orchids	Flower	T-Rex	Horns
SRN [6]	27.29	21.37	18.24	26.63	17.37	24.63	22.87	24.33
LLFF [3]	28.42	22.85	19.52	29.40	18.52	25.46	24.15	24.70
Ours	**32.70**	**25.17**	**20.92**	**31.16**	**20.36**	**27.40**	**26.80**	**27.45**

	SSIM↑							
	Room	Fern	Leaves	Fortress	Orchids	Flower	T-Rex	Horns
SRN [6]	0.883	0.611	0.520	0.641	0.449	0.738	0.761	0.742
LLFF [3]	0.932	0.753	**0.697**	0.872	0.588	**0.844**	0.857	**0.840**
Ours	**0.948**	**0.792**	0.690	**0.881**	**0.641**	0.827	**0.880**	0.828

	LPIPS↓							
	Room	Fern	Leaves	Fortress	Orchids	Flower	T-Rex	Horns
SRN [6]	0.240	0.459	0.440	0.453	0.467	0.288	0.298	0.376
LLFF [3]	**0.155**	**0.247**	**0.216**	0.173	**0.313**	**0.174**	**0.222**	**0.193**
Ours	0.178	0.280	0.316	**0.171**	0.321	0.219	0.249	0.268

Table 3: Per-scene quantitative results from our real image dataset. The scenes in this dataset are all captured with a forward-facing handheld cellphone.

		PSNR↑							
		Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship
1)	No PE, VD, H	28.44	23.11	25.17	32.24	26.38	24.69	28.16	25.12
2)	No Pos. Encoding	30.33	24.54	29.32	33.16	27.75	27.79	30.76	26.55
3)	No View Dependence	30.06	23.41	25.91	32.65	29.93	24.96	28.62	25.72
4)	No Hierarchical	31.32	24.55	29.25	35.24	31.42	29.22	31.74	27.73
5)	Far Fewer Images	30.92	22.62	24.39	32.77	27.97	26.55	30.47	26.57
6)	Fewer Images	32.19	23.70	27.45	34.91	31.53	28.54	32.33	27.67
7)	Fewer Frequencies	32.19	**25.29**	**30.73**	36.06	30.77	**29.77**	31.66	28.26
8)	More Frequencies	32.87	24.65	29.92	35.78	32.50	29.54	32.86	28.34
9)	Complete Model	**33.00**	25.01	30.13	**36.18**	**32.54**	29.62	**32.91**	**28.65**

		SSIM↑							
		Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship
1)	No PE, VD, H	0.919	0.896	0.926	0.955	0.882	0.905	0.955	0.810
2)	No Pos. Encoding	0.938	0.918	0.953	0.956	0.903	0.933	0.968	0.824
3)	No View Dependence	0.948	0.906	0.938	0.961	0.947	0.912	0.962	0.828
4)	No Hierarchical	0.951	0.914	0.956	0.969	0.951	0.944	0.973	0.844
5)	Far Fewer Images	0.956	0.895	0.922	0.966	0.930	0.925	0.972	0.832
6)	Fewer Images	0.963	0.911	0.948	0.971	0.957	0.941	0.979	0.847
7)	Fewer Frequencies	0.959	**0.928**	**0.965**	0.972	0.947	**0.952**	0.973	0.853
8)	More Frequencies	0.967	0.921	0.962	0.973	0.961	0.948	**0.980**	0.853
9)	Complete Model	**0.967**	0.925	0.964	**0.974**	**0.961**	0.949	0.980	**0.856**

		LPIPS↓							
		Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship
1)	No PE, VD, H	0.095	0.168	0.084	0.104	0.178	0.111	0.084	0.261
2)	No Pos. Encoding	0.076	0.104	0.050	0.124	0.128	0.079	0.041	0.261
3)	No View Dependence	0.075	0.148	0.113	0.112	0.088	0.102	0.073	0.220
4)	No Hierarchical	0.065	0.177	0.056	0.130	0.072	0.080	0.039	0.249
5)	Far Fewer Images	0.058	0.173	0.082	0.123	0.081	0.079	0.035	0.229
6)	Fewer Images	0.051	0.166	0.057	0.121	0.055	0.068	0.029	0.223
7)	Fewer Frequencies	0.055	0.143	**0.038**	**0.087**	0.071	**0.060**	0.029	0.219
8)	More Frequencies	0.047	0.158	0.045	0.116	0.050	0.064	**0.027**	0.261
9)	Complete Model	**0.046**	**0.091**	0.044	0.121	**0.050**	0.063	0.028	**0.206**

Table 4: Per-scene quantitative results from our ablation study. The scenes used here are the same as in Table 2.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015)
2. Lombardi, S., Simon, T., Saragih, J., Schwartz, G., Lehrmann, A., Sheikh, Y.: Neural volumes: Learning dynamic renderable volumes from images. ACM Transactions on Graphics (SIGGRAPH) (2019)
3. Mildenhall, B., Srinivasan, P.P., Ortiz-Cayon, R., Kalantari, N.K., Ramamoorthi, R., Ng, R., Kar, A.: Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. ACM Transactions on Graphics (SIGGRAPH) (2019)
4. Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: DeepSDF: Learning continuous signed distance functions for shape representation. In: CVPR (2019)
5. Sitzmann, V., Thies, J., Heide, F., Nießner, M., Wetzstein, G., Zollhöfer, M.: Deepvoxels: Learning persistent 3D feature embeddings. In: CVPR (2019)
6. Sitzmann, V., Zollhoefer, M., Wetzstein, G.: Scene representation networks: Continuous 3D-structure-aware neural scene representations. In: NeurIPS (2019)