Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields Supplemental Material

Dor Verbin[1,2] Peter Hedman[2] Ben Mildenhall[2]
Todd Zickler[1] Jonathan T. Barron[2] Pratul P. Srinivasan[2]
[1]Harvard University [2]Google Research

A. Integrated Directional Encoding Proofs

We begin by proving the expression for the expected value of a spherical harmonic under a vMF distribution in Equation 7 in our main paper.

Claim 1. *The expected value of a spherical harmonic function $Y_\ell^m(\hat{\boldsymbol{\omega}})$ under a vMF distribution with mean $\hat{\boldsymbol{\omega}}_r$ and concentration parameter κ is:*

$$\mathbb{E}_{\hat{\boldsymbol{\omega}}\sim\text{vMF}(\hat{\boldsymbol{\omega}}_r,\kappa)}[Y_\ell^m(\hat{\boldsymbol{\omega}})] = A_\ell(\kappa)Y_\ell^m(\hat{\boldsymbol{\omega}}_r), \tag{S1}$$

where:

$$A_\ell(\kappa) = \frac{\kappa}{2\sinh\kappa}\int_{-1}^{1} P_\ell(u)e^{\kappa u}du, \tag{S2}$$

with P_ℓ the ℓth Legendre polynomial.

Proof. We begin by first aligning the mean direction of the distribution $\hat{\boldsymbol{\omega}}_r$ with the z-axis. We do this by applying a rotation matrix R to transform $\hat{\boldsymbol{\omega}}' = R\hat{\boldsymbol{\omega}}$ where we choose R that satisfies $R\hat{\boldsymbol{\omega}}_r = \hat{z}$. Since the Jacobian of this transformation has unit determinant, we can write:

$$\begin{aligned}\mathbb{E}_{\hat{\boldsymbol{\omega}}\sim\text{vMF}(\hat{\boldsymbol{\omega}}_r,\kappa)}[Y_\ell^m(\hat{\boldsymbol{\omega}})] &= c(\kappa)\int_{S^2} Y_\ell^m(\hat{\boldsymbol{\omega}})e^{\kappa\hat{\boldsymbol{\omega}}_r^\top\hat{\boldsymbol{\omega}}}d\hat{\boldsymbol{\omega}} \\ &= c(\kappa)\int_{S^2} Y_\ell^m(R^{-1}\hat{\boldsymbol{\omega}})e^{\kappa\cos\theta}d\hat{\boldsymbol{\omega}},\end{aligned} \tag{S3}$$

where $c(\kappa) = \frac{\kappa}{4\pi\sinh\kappa}$ is the normalization factor of the vMF distribution, and θ is the angle between $\hat{\boldsymbol{\omega}}_r$ and the z-axis.

A rotated spherical harmonic can be written as a linear combination of all spherical harmonics of the same degree, with coefficients specified by the Wigner D-matrix of the rotation:

$$Y_\ell^m(R^{-1}\hat{\boldsymbol{\omega}}) = \sum_{m'=-\ell}^{\ell} D_{mm'}^{(\ell)}(\hat{\boldsymbol{\omega}}_r)Y_\ell^{m'}(\hat{\boldsymbol{\omega}}). \tag{S4}$$

Plugging this into the expression from Equation S3:

$$\begin{aligned}&\mathbb{E}_{\hat{\boldsymbol{\omega}}\sim\text{vMF}(\hat{\boldsymbol{\omega}}_r,\kappa)}[Y_\ell^m(\hat{\boldsymbol{\omega}})] \\ &= c(\kappa)\sum_{m'=-\ell}^{\ell} D_{mm'}^{(\ell)}(\hat{\boldsymbol{\omega}}_r)\int_{S^2} Y_\ell^{m'}(\hat{\boldsymbol{\omega}})e^{\kappa\cos\theta}d\hat{\boldsymbol{\omega}}.\end{aligned} \tag{S5}$$

The azimuthal dependence of the integrand is periodic in 2π for any $m' \neq 0$, so the only integral that does not vanish is the one with $m' = 0$, yielding:

$$\begin{aligned}&\mathbb{E}_{\hat{\boldsymbol{\omega}}\sim\text{vMF}(\hat{\boldsymbol{\omega}}_r,\kappa)}[Y_\ell^m(\hat{\boldsymbol{\omega}})] \\ &= c(\kappa)D_{m0}^{(\ell)}(\hat{\boldsymbol{\omega}}_r)\int_{S^2} Y_\ell^0(\hat{\boldsymbol{\omega}})e^{\kappa\cos\theta}d\hat{\boldsymbol{\omega}}.\end{aligned} \tag{S6}$$

Plugging the known expression for the ℓth degree 0th order spherical harmonic $Y_\ell^0(\hat{\boldsymbol{\omega}}) = \sqrt{\frac{2\ell+1}{4\pi}}P_\ell(\cos\theta)$ and the corresponding elements of the Wigner D-matrix $D_{m0}^{(\ell)}(\hat{\boldsymbol{\omega}}_r) = \sqrt{\frac{4\pi}{2\ell+1}}Y_\ell^m(\hat{\boldsymbol{\omega}}_r)$, and integrating over the azimuthal angle, we get:

$$\begin{aligned}&\mathbb{E}_{\hat{\boldsymbol{\omega}}\sim\text{vMF}(\hat{\boldsymbol{\omega}}_r,\kappa)}[Y_\ell^m(\hat{\boldsymbol{\omega}})] \\ &= 2\pi c(\kappa)Y_\ell^m(\hat{\boldsymbol{\omega}}_r)\int_0^\pi P_\ell(\cos\theta)e^{\kappa\cos\theta}\sin\theta d\theta \\ &= 2\pi c(\kappa)Y_\ell^m(\hat{\boldsymbol{\omega}}_r)\int_{-1}^{1} P_\ell(u)e^{\kappa u}du \\ &= \frac{\kappa}{2\sinh\kappa}Y_\ell^m(\hat{\boldsymbol{\omega}}_r)\int_{-1}^{1} P_\ell(u)e^{\kappa u}du \\ &= A_\ell(\kappa)Y_\ell^m(\hat{\boldsymbol{\omega}}_r),\end{aligned} \tag{S7}$$

where the second equality was obtained using the change of variables $u = \cos\theta$. □

Next, we show that the integral has a closed-form solution, leading to a simple expression for the ℓth attenuation function, $A_\ell(\kappa)$.

Claim 2. *For any $\ell \in \mathbb{N}$ the attenuation function A_ℓ satisfies:*

$$A_\ell(\kappa) = \kappa^{-\ell} \sum_{i=0}^{\ell} \frac{(2\ell - i)!}{i!(\ell - i)!} (-2)^{i-\ell} b_i(\kappa), \tag{S8}$$

where $b_i(\kappa) = \kappa^i$ for even values of i and $b_i(\kappa) = \kappa^i \coth(\kappa)$ for odd i.

Proof. We prove this by first finding a recurrence relation for the attenuation functions, and then proving our expression by induction.

From Equation S2, the $(\ell-1)$th attenuation function can be written as:

$$\begin{aligned} A_{\ell-1}(\kappa) &= \frac{\kappa}{2\sinh\kappa} \int_{-1}^{1} P_{\ell-1}(u) e^{\kappa u} du \\ &= \frac{\kappa}{2\sinh\kappa} \int_{-1}^{1} \left(\frac{d}{du} \frac{P_\ell(u) - P_{\ell-2}(u)}{2\ell - 1} \right) e^{\kappa u} du \\ &= \frac{\kappa}{2\sinh\kappa} \left(\frac{P_\ell(u) - P_{\ell-2}(u)}{2\ell - 1} \right) e^{\kappa u} \Big|_{-1}^{1} \\ &\quad - \frac{\kappa^2}{2\sinh\kappa} \int_{-1}^{1} \left(\frac{P_\ell(u) - P_{\ell-2}(u)}{2\ell - 1} \right) e^{\kappa u} du \\ &= -\frac{\kappa}{2\ell - 1} \left(A_\ell(\kappa) - A_{\ell-2}(\kappa) \right). \end{aligned} \tag{S9}$$

where the second equality was obtained using a known recurrence relation of the Legendre polynomials, the third was obtained using integration by parts, and the fourth by using the fact that $P_\ell(\pm 1) - P_{\ell-2}(\pm 1) = 0$. Reordering, we get:

$$A_\ell(\kappa) = A_{\ell-2}(\kappa) - \frac{2\ell - 1}{\kappa} A_{\ell-1}(\kappa). \tag{S10}$$

We can easily find the first two attenuation functions by directly computing the integrals:

$$\begin{aligned} A_0(\kappa) &= \frac{\kappa}{2\sinh\kappa} \int_{-1}^{1} e^{\kappa u} du = 1, \\ A_1(\kappa) &= \frac{\kappa}{2\sinh\kappa} \int_{-1}^{1} u e^{\kappa u} du = \coth(\kappa) - \frac{1}{\kappa}. \end{aligned} \tag{S11}$$

Finally, we prove our claim by induction using the recurrence relation in Equation S10. It is easy to verify that our expression holds for $\ell = 0$ and $\ell = 1$ by plugging these values in Equation S8 and comparing with Equation S11.

We now assume that the relation holds for any $\ell - 2$ and $\ell - 1$ and prove it for $\ell \geq 2$. We can do this by considering the coefficient of κ^{-m} for every $m \in \{0, ..., \ell\}$ of the right hand side of Equation S10, and show that it is identical to the one from Equation S8. We begin by separately considering $m = 0$, $m = \ell - 1$ and $m = \ell$, and then any other m. Note that for some m values κ^{-m} is multiplied by $\coth(\kappa)$, but this factor always matches in $A_{\ell-2}(\kappa)$, in $\frac{1}{\kappa} A_{\ell-1}(\kappa)$, and in $A_\ell(\kappa)$, so we neglect it.

For $m = 0$, the only contribution to the coefficient of κ^0 is from $A_{\ell-2}$, which gives a coefficient of 1, matching with the κ^0 coefficient of A_ℓ.

For $m = \ell - 1$ we only get a contribution from the second term, and the coefficient is:

$$\begin{aligned} &-(2\ell - 1) \frac{(2\ell - 3)!}{1!(\ell - 2)!} (-2)^{2-\ell} \\ &\quad = \frac{(2\ell - 1)(\ell - 1)(2\ell - 3)!}{1!(\ell - 1)(\ell - 2)!} \cdot 2 \cdot (-2)^{1-\ell} \\ &\quad = \frac{(2\ell - 1)(2\ell - 2)(2\ell - 3)!}{1!(\ell - 1)(\ell - 2)!} (-2)^{1-\ell} \\ &\quad = \frac{(2\ell - 1)!}{1!(\ell - 1)!} (-2)^{1-\ell}, \end{aligned} \tag{S12}$$

which matches with that of A_ℓ from Equation S8.

Similarly, for $m = \ell$ the contribution is also from the second term, with the coefficient:

$$\begin{aligned} &-(2\ell - 1) \frac{(2\ell - 2)!}{0!(\ell - 1)!(-2)^{1-\ell}} \\ &\quad = 2 \frac{(2\ell - 1)!}{0!(\ell - 1)!} (-2)^{\ell} \\ &\quad = 2 \frac{2\ell (2\ell - 1)!}{0! \cdot 2\ell \cdot (\ell - 1)!} (-2)^{\ell} \\ &\quad = \frac{(2\ell)!}{0!\ell!} (-2)^{-\ell}, \end{aligned} \tag{S13}$$

and again this matches with the coefficient of A_ℓ.

Finally, for any $m \in \{1, ..., \ell - 2\}$, we have that the coefficient of κ^{-m} on the right hand side of Equation S10 is:

$$\begin{aligned} &\frac{(\ell + m - 2)!}{(\ell - m - 2)! m!} (-2)^{-m} \\ &\quad - (2\ell - 1) \frac{(\ell + m - 2)!}{(\ell - m)!(m - 1)!} (-2)^{1-m} \\ &= \frac{(-2)^{-m} (\ell + m - 2)!}{(\ell - m)! m!} \\ &\quad \cdot [(\ell - m)(\ell - m - 1) + 2m(2\ell - 1)] \\ &= \frac{(-2)^{-m} (\ell + m - 2)!}{(\ell - m)! m!} (\ell + m)(\ell + m - 1) \\ &= \frac{(\ell + m)!}{(\ell - m)! m!} (-2)^{-m}, \end{aligned} \tag{S14}$$

which is identical to the coefficient in Equation S8, finishing our proof.

□

Computing the attenuation function using Equation S8 is inefficient, and it is numerically unstable due to catastrophic

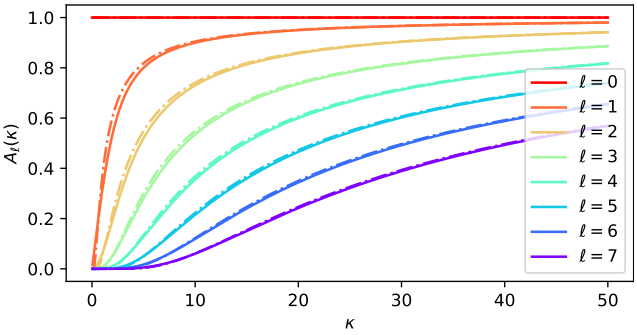

Figure S1. The first eight attenuation functions $A_\ell(\kappa)$. The exact expression (in solid lines) matches the approximation (in dashed lines), with an increasing accuracy as ℓ increases.

cancellation. In Equation 8 of the main paper we present an approximation which is guaranteed to be close to the exact functions from Equation S8, for large values of κ. We finish this section with a proof of this claim. Additionally, Figure S1 shows that our approximation closely resembles the exact attenuation functions for all values of $\kappa > 0$ and all orders ℓ, and that its quality improves as ℓ increases.

Claim 3. *For large values of κ, our approximation is exact up to an $O(1/\kappa^2)$ term,* i.e.*:*

$$A_\ell(\kappa) = \exp\left(-\frac{\ell(\ell+1)}{2\kappa}\right) + O\left(\frac{1}{\kappa^2}\right) \tag{S15}$$

Proof. Using the fact that $\coth(\kappa) = 1 + O(e^{-2\kappa})$, and using the $i = \ell$ and $i = \ell - 1$ terms from the sum in Equation S8 of Claim 2, we have that:

$$A_\ell(\kappa) = 1 - \frac{\ell(\ell+1)}{2\kappa} + O\left(\frac{1}{\kappa^2}\right). \tag{S16}$$

Using a 1st order Taylor approximation for the exponential function yields:

$$\exp\left(-\frac{\ell(\ell+1)}{2\kappa}\right) = 1 - \frac{\ell(\ell+1)}{2\kappa} + O\left(\frac{1}{\kappa^2}\right), \tag{S17}$$

proving our claim. □

B. Interpretations of Normal Vector Penalty

As described in the main paper, the regularization term in Equation 11 penalizes "backwards-facing" normal vectors that contribute to the ray's final rendered color.

Note that Equation 11 applies this penalty to the normals output by the spatial MLP as opposed to the normal vectors directly computed from the gradient of the density field. While this penalty is remarkably effective at improving recovered normal vectors, we find that directly applying it to the gradient density normals sometimes affects the optimization dynamics for fine geometric structures. Even when NeRF recovers a concentrated surface at the end of optimization, it typically passes through a fuzzier volumetric representation during optimization, and directly penalizing the gradient of the volume density can sometimes adversely affect this trajectory. Our approach of applying the orientation penalty to the predicted normals gives the spatial MLP two options at every location: it can either predict normals that agree with the density gradient ones, in which case our orientation loss is effectively applied to the density field; or it can predict normals that deviate from those computed from the density field, and incur a normal prediction penalty $\mathcal{R}_p$. We find that applying the penalty to predicted normals allows our method to adaptively apply the orientation loss, resulting in accurate normals without loss of fine details (see ablation studies).

Besides providing smooth normals for computing reflection directions and providing a mechanism for an adaptive corner-preserving orientation loss, the predicted normals also allow the model to directly encode view directions by predicting a constant normal direction $\hat{\mathbf{n}}'(\mathbf{x}) = \hat{\mathbf{c}}$, leading to a 1-to-1 mapping from view direction to reflection direction (see Equation 4). While this is discouraged by the normal prediction loss $\mathcal{R}_p$, our model can exploit this behavior in regions that are not well-described by a surface at a scale whose geometry the density field can capture.

C. Optimization and Additional Model Details

Optimization Our implementation is based on the official JAX [2] implementation of mip-NeRF [1].

In all of our experiments on synthetic data we apply the normal orientation loss from Equation 11 and the normal prediction loss from Equation 10 of the main paper with weights of 0.1 and $3 \cdot 10^{-4}$ respectively, relative to the standard NeRF data loss from Equation 2. In our experiments on real captured data, we apply a slightly higher weight of 10^{-3} on the normal prediction loss. We use the same weights during both the coarse and fine stages.

During training, we add i.i.d. Gaussian noise with standard deviation 0.1 to the bottleneck $\mathbf{b}$. We find that this slightly stabilizes our model's results in some cases by preventing it from using the bottleneck early in training.

We optimize all experiments on the Blender scenes using single-image batching to be consistent with results reported in the NeRF and mip-NeRF papers. For our Shiny Blender dataset, we sample random batches of rays from all images.

We train our model, ablations of our model, and all mip-NeRF baselines using a slightly modified version of mip-NeRF's learning schedule: 250k optimization iterations with a batch size of 2^{14}, using the Adam [3] optimizer with hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-6}$, a learning rate that is annealed log-linearly from 2×10^{-3} to 2×10^{-5} with a warm-up phase of 512 iterations, and gradient clipping to a norm of 10^{-3}.

	chair	lego	materials	mic	hotdog	ficus	drums	ship
PhySG [7]	24.00	20.19	18.86	22.33	24.08	19.02	20.99	15.35
VolSDF [6]	30.57	29.46	29.13	30.53	35.11	22.91	20.43	25.51
Mip-NeRF [1]	35.12	35.92	30.64	36.76	37.34	33.19	25.36	30.52
Ours, standard encoding	35.86	36.33	35.22	35.84	38.18	33.60	26.03	30.17
Ours	35.83	36.25	35.41	36.76	37.72	33.91	25.79	30.28

Table S1. Per-scene test set PSNRs on the Blender dataset [5].

	chair	lego	materials	mic	hotdog	ficus	drums	ship
PhySG [7]	0.898	0.821	0.838	0.933	0.912	0.873	0.884	0.727
VolSDF [6]	0.949	0.951	0.954	0.969	0.972	0.929	0.893	0.842
Mip-NeRF [1]	0.981	0.980	0.959	0.992	0.982	0.980	0.933	0.885
Ours, standard encoding	0.984	0.981	0.983	0.991	0.984	0.982	0.939	0.878
Ours	0.984	0.981	0.983	0.992	0.984	0.983	0.937	0.880

Table S2. Per-scene test set SSIMs on the Blender dataset [5].

	chair	lego	materials	mic	hotdog	ficus	drums	ship
PhySG [7]	0.093	0.172	0.142	0.082	0.117	0.112	0.113	0.322
VolSDF [6]	0.056	0.054	0.048	0.191	0.043	0.068	0.119	0.191
Mip-NeRF [1]	0.020	0.018	0.040	0.008	0.026	0.021	0.064	0.135
Ours, standard encoding	0.017	0.018	0.023	0.008	0.022	0.020	0.059	0.143
Ours	0.017	0.018	0.022	0.007	0.022	0.019	0.059	0.139

Table S3. Per-scene test set LPIPS on the Blender dataset [5].

	chair	lego	materials	mic	hotdog	ficus	drums	ship
PhySG [7]	18.569	40.244	18.986	26.053	28.572	35.974	21.696	43.265
VolSDF [6]	14.085	26.619	8.277	19.579	12.170	39.801	21.458	16.974
Mip-NeRF [1]	28.044	30.532	64.074	36.489	29.303	53.524	32.374	37.667
Ours, standard encoding	20.018	26.471	10.162	25.921	13.920	41.557	27.766	34.212
Ours	19.852	24.469	9.531	24.938	13.211	41.052	27.853	31.707

Table S4. Per-scene test set normal MAEs on the Blender dataset [5].

Additional Model Details The three factors of our decomposition of color into diffuse and specular components use different nonlinearities to map the outputs of the MLPs to their appropriate ranges. The tint $\mathbf{s}$ and specular color $\mathbf{c}_s$ are computed using a sigmoid applied to the raw outputs from the MLPs:

$$\begin{aligned} \mathbf{c}_s &= \sigma(\tilde{\mathbf{c}}_s) \\ \mathbf{s} &= \sigma(\tilde{\mathbf{s}}), \end{aligned} \tag{S18}$$

where $\tilde{\mathbf{c}}_s$ is the three-channel raw specular color output by the directional MLP, and $\tilde{\mathbf{s}}$ is the three-channel raw tint output by the spatial MLP.

The diffuse color is computed by passing the corresponding three spatial MLP channels through an offset sigmoid function as follows:

$$\mathbf{c}_d = \sigma(\tilde{\mathbf{c}}_d - \log 3), \tag{S19}$$

where $\tilde{\mathbf{c}}_d$ is the spatial MLP's raw output. The $\log 3$ offset is designed to initialize the diffuse color around 0.25, which initializes the combined raw color to around 0.5.

The three factors are then combined into a single color using Equation 9 from the main paper.

D. Evaluation Details

Normal Vectors To compute the normal vector for a ray, we sample normals along the ray $\{\hat{\mathbf{n}}_i\}$ using Equation 3 from the main paper, and use the volume rendering procedure from Equation 1 to compute a single normal vector:

$$\mathbf{N}(\mathbf{o}, \hat{\mathbf{d}}) = \sum_i w_i \hat{\mathbf{n}}(\mathbf{x}_i). \tag{S20}$$

	teapot	toaster	car	ball	coffee	helmet
PhySG [7]	35.83	18.59	24.40	27.24	23.71	27.51
Mip-NeRF [1]	46.00	22.37	26.50	25.94	30.36	27.39
Mip-NeRF, 8 layers	47.35	25.51	27.99	27.53	32.14	29.04
Mip-NeRF, 8 layers, w/ normals	47.09	25.14	27.97	26.79	32.12	29.21
Mip-NeRF, 8 layers, with $\mathcal{R}_o$	47.35	25.32	27.91	26.89	32.21	29.22
Ours, no reflection	44.74	24.04	27.41	20.94	31.95	27.76
Ours, no $\mathcal{R}_o$	46.80	25.78	28.43	27.06	32.58	29.06
Ours, no pred. normals	47.09	23.32	27.19	26.09	31.79	30.54
Ours, concat. viewdir	46.01	25.38	30.71	47.45	34.19	28.81
Ours, fixed lobe	46.82	25.57	30.09	47.25	34.37	29.00
Ours, no diffuse color	46.45	25.56	30.46	34.53	34.05	28.87
Ours, no tint	46.54	25.49	30.14	47.53	34.24	28.78
Ours, no roughness	45.28	25.39	30.44	36.33	33.19	29.72
Ours, standard encoding	46.55	26.74	30.53	47.56	34.45	29.59
Ours	47.90	25.70	30.82	47.46	34.21	29.68

Table S5. Per-scene test set PSNRs on our Shiny Blender dataset.

	teapot	toaster	car	ball	coffee	helmet
PhySG [7]	0.990	0.805	0.910	0.947	0.922	0.953
Mip-NeRF [1]	0.997	0.891	0.922	0.935	0.966	0.939
Mip-NeRF, 8 layers	0.997	0.925	0.936	0.949	0.971	0.957
Mip-NeRF, 8 layers, w/ normals	0.997	0.923	0.936	0.946	0.970	0.957
Mip-NeRF, 8 layers, with $\mathcal{R}_o$	0.997	0.924	0.935	0.947	0.971	0.957
Ours, no reflection	0.996	0.912	0.930	0.905	0.970	0.950
Ours, no $\mathcal{R}_o$	0.997	0.926	0.937	0.939	0.971	0.956
Ours, no pred. normals	0.997	0.898	0.926	0.865	0.967	0.962
Ours, concat. viewdir	0.997	0.919	0.956	0.995	0.974	0.952
Ours, fixed lobe	0.997	0.920	0.952	0.995	0.974	0.954
Ours, no diffuse color	0.997	0.920	0.953	0.977	0.973	0.954
Ours, no tint	0.997	0.921	0.951	0.995	0.974	0.954
Ours, no roughness	0.996	0.917	0.954	0.983	0.972	0.958
Ours, standard encoding	0.997	0.928	0.954	0.996	0.975	0.958
Ours	0.998	0.922	0.955	0.995	0.974	0.958

Table S6. Per-scene test set SSIMs on our Shiny Blender dataset.

	teapot	toaster	car	ball	coffee	helmet
PhySG [7]	0.022	0.194	0.091	0.179	0.150	0.089
Mip-NeRF [1]	0.008	0.123	0.059	0.168	0.086	0.108
Mip-NeRF, 8 layers	0.006	0.080	0.052	0.139	0.082	0.072
Mip-NeRF, 8 layers, w/ normals	0.006	0.085	0.052	0.144	0.082	0.072
Mip-NeRF, 8 layers, with $\mathcal{R}_o$	0.006	0.082	0.052	0.143	0.082	0.072
Ours, no reflection	0.007	0.091	0.052	0.192	0.082	0.080
Ours, no $\mathcal{R}_o$	0.008	0.086	0.051	0.161	0.082	0.076
Ours, no pred. normals	0.006	0.134	0.064	0.272	0.087	0.068
Ours, concat. viewdir	0.006	0.095	0.040	0.061	0.079	0.087
Ours, fixed lobe	0.009	0.096	0.043	0.061	0.080	0.080
Ours, no diffuse color	0.009	0.096	0.043	0.095	0.079	0.080
Ours, no tint	0.008	0.094	0.043	0.060	0.079	0.078
Ours, no roughness	0.009	0.099	0.042	0.086	0.081	0.074
Ours, standard encoding	0.007	0.092	0.041	0.060	0.077	0.074
Ours	0.004	0.095	0.041	0.059	0.078	0.075

Table S7. Per-scene test set LPIPS on our Shiny Blender dataset.

	teapot	toaster	car	ball	coffee	helmet
PhySG [7]	6.634	9.749	8.844	0.700	22.514	2.324
Mip-NeRF [1]	66.470	42.787	40.954	104.765	29.427	77.904
Mip-NeRF, 8 layers	68.238	35.220	23.670	127.863	25.465	67.966
Mip-NeRF, 8 layers, w/ normals	67.999	34.093	23.548	130.755	26.527	66.701
Mip-NeRF, 8 layers, with $\mathcal{R}_o$	68.238	35.837	23.985	127.683	24.101	64.372
Ours, no reflection	19.263	19.325	13.643	15.142	9.730	20.038
Ours, no $\mathcal{R}_o$	43.116	43.113	37.134	106.003	29.301	56.710
Ours, no pred. normals	67.999	24.886	21.644	48.061	10.848	10.573
Ours, concat. viewdir	20.359	40.587	12.877	1.577	9.489	42.615
Ours, fixed lobe	35.791	42.183	18.410	1.536	24.045	36.785
Ours, no diffuse color	38.347	42.705	16.802	5.423	15.363	38.119
Ours, no tint	29.537	43.687	16.854	1.556	11.233	33.327
Ours, no roughness	33.821	44.666	17.440	3.557	26.578	29.723
Ours, standard encoding	23.123	41.415	16.214	1.969	9.749	29.409
Ours	9.234	42.870	14.927	1.548	12.240	29.484

Table S8. Per-scene test set MAEs on our Shiny Blender dataset.

We use Equation S20 to visualize normal maps, with gray-colored values corresponding to high variance normal vectors along a ray. For evaluating MAE, we use the normalized accumulated normals, $\hat{\mathbf{N}} = \mathbf{N}/\|\mathbf{N}\|$.

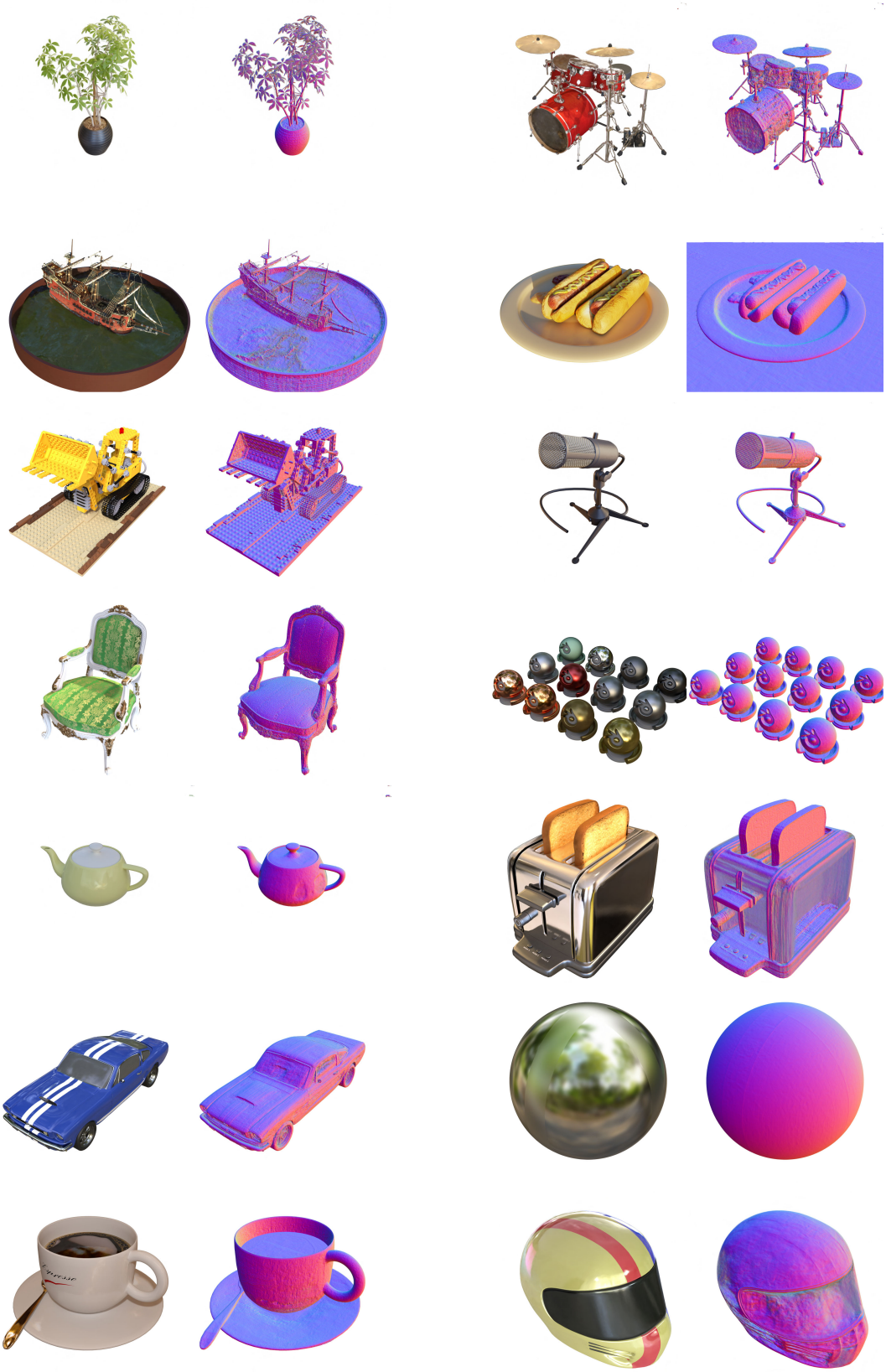

Figure S2. Rendered images and normals estimated by our method on all 14 scenes in the Blender and Shiny Blender datasets.

	sedan	*toy car*	*garden spheres*
Mip-NeRF, 8 layers	25.53 / **0.729** / **0.118**	24.00 / 0.663 / 0.177	23.40 / **0.620** / **0.125**
Ours	**25.65** / 0.720 / 0.119	**24.25** / **0.674** / **0.168**	**23.46** / 0.601 / 0.138

Table S9. Quantitative metrics (PSNR/SSIM/LPIPS) on the test sets of our real captured scenes.

E. Additional Results

Table S9 reports quantitative metrics on the test sets (every eighth image is held out for testing, as done in NeRF [5]) of our three real captured scenes. Tables S1, S2, S3, and S4 contain quantitative metrics on the original synthetic Blender dataset, and Tables S5, S6, S7, and S8 contain quantitative metrics on our new synthetic Shiny Blender dataset.

Figure S2 shows a rendered image and the normals estimated by our method for all 14 scenes from the Blender and Shiny Blender datasets.

Baseline Implementations For comparisons to mip-NeRF [1], we use the official open-source JAX implementation. For comparisons to PhySG [7], we use the official PyTorch code open-sourced by the authors. However, the original hyperparameters produce poor results on our datasets, so the PhySG authors helped us tune their hyperparameters for the Blender datasets. VolSDF [6] does not currently have publicly-available code, but the authors graciously ran their code on the Blender dataset and provided us with rendered test-set images and normals. We report the NSVF [4] and NeRF [5] results on the Blender dataset from the tables in the mip-NeRF paper.

F. Dataset Details

Our new "Shiny Blender" scenes were adapted from the the following BlendSwap models:

1. Coffee: created by *sleem*, CC-0 license (model #10827).
2. Toaster: created by *PrinterKiller*, CC-BY license (model #4989)
3. Car: created by *Xali*, CC-0 license (model #24359).
4. Helmet: created by *kveidem*, CC-0 license (model #21617).

Our dataset of real scenes was captured by the authors.

G. Societal Impact

Environmental impact is a significant concern when training NeRF-based models, as they typically require hours of training per scene. We hope that future improvements to NeRF and continued progress in developing efficient ML accelerators will ameliorate this issue. 3D scene reconstruction techniques such as ours can also be used for illicit surveillance. In particular, improved rendering of shiny surfaces has the potential to reveal details of objects outside of the cameras' fields of view by reconstructing their reflections off of visible objects.

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. *ICCV*, 2021. 3, 4, 6

[2] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. `http://github.com/google/jax`. 3

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 3

[4] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020. 6

[5] 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. *ECCV*, 2020. 4, 6

[6] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *NeurIPS*, 2021. 4, 6

[7] Kai Zhang, Fujun Luan, Qianqian Wang, Kavita Bala, and Noah Snavely. PhySG: Inverse rendering with spherical gaussians for physics-based material editing and relighting. *CVPR*, 2021. 4, 6