A. Rectangle Size Derivation

As mentioned in Section 4.4 of the main paper, a ray that reaches the environment map is assigned a color taken as the average color over an axis-aligned rectangle in spherical coordinates, where the shape of the rectangle depends on the ray's direction and the material's roughness at the ray's origin. We modify the derivation of the area of the rectangle from GPU Gems [10]. Let N be the number of samples, $p(\hat{\boldsymbol{\omega}}_i)$ be the probability density function of a given sample direction $\hat{\boldsymbol{\omega}}_i$ for viewing direction $\hat{\boldsymbol{\omega}}_o$, and let H and W be the height and width of the environment map (*i.e.* its polar and azimuthal resolutions). The density $d(\hat{\boldsymbol{\omega}}_i)$ of environment map pixels at a given direction must be inversely proportional to the Jacobian's determinant, $\sin\theta_i$, and it must also satisfy:

$$HW = \int_0^{2\pi}\int_0^{\pi} d(\hat{\boldsymbol{\omega}}_i)\sin\theta_i d\theta_i d\phi_i, \tag{16}$$

and therefore:

$$d(\hat{\boldsymbol{\omega}}_i) = \frac{HW}{2\pi^2\sin\theta_i}. \tag{17}$$

The number of pixels per sample, which is the area of the rectangle, is then the total solid angle per sample, $Np(\hat{\boldsymbol{\omega}}_i)$ multiplied by the number of pixels per solid angle:

$$\Delta\theta\cdot\Delta\phi = \frac{Np(\hat{\boldsymbol{\omega}}_i)}{d(\hat{\boldsymbol{\omega}}_i)}, \tag{18}$$

where $\Delta\theta$ is the polar size of the rectangle, and $\Delta\phi$ is its azimuthal size, *i.e.* the rectangle is $\Delta\theta\times\Delta\phi$, in equirectangular coordinates.

As mentioned in Section 4.4 of the main paper, the aspect ratio of the rectangle is set to:

$$\frac{\Delta\theta}{\Delta\phi} = \sin\theta_i, \tag{19}$$

which yields:

$$\Delta\theta = \sqrt{2\pi^2\frac{N}{HW}p(\hat{\boldsymbol{\omega}}_i)}\cdot\sin\theta_i, \tag{20}$$

$$\Delta\phi = \sqrt{2\pi^2\frac{N}{HW}p(\hat{\boldsymbol{\omega}}_i)}. \tag{21}$$

B. BSDF Neural Network Parameterization

Once we have sampled the incoming light directions $\hat{\boldsymbol{\omega}}_i$ and their respective values $L(\mathbf{p},\hat{\boldsymbol{\omega}}_i)$, we transform them into the local shading frame to calculate the value of the neural shading network h. We parameterize the neural network with 2 hidden layers of width 64 as $h(\mathbf{p},\hat{\boldsymbol{\omega}}_o,\hat{\boldsymbol{\omega}}_i,\hat{\mathbf{n}})$, where $\mathbf{p}$ is the position, $\hat{\boldsymbol{\omega}}_o,\hat{\boldsymbol{\omega}}_i$ are the outgoing and incoming light directions, respectively, and $\hat{\mathbf{n}}$ is the normal. However, rather than feeding $\hat{\boldsymbol{\omega}}_o$ and $\hat{\boldsymbol{\omega}}_i$ to the network directly, we follow the schema laid out by Rusinkiewicz [34] and parameterize the input using the halfway vector $\hat{\boldsymbol{h}}$ and difference vector $\hat{\boldsymbol{d}}$ within the local shading frame $F(\hat{\mathbf{n}})$, which takes the world space to a frame of reference in which the normal vector points upwards:

$$T = [0,0,1]^\top\times\hat{\mathbf{n}} \tag{22}$$

$$F(\hat{\mathbf{n}}) = \begin{bmatrix} T, & \hat{\mathbf{n}}\times T, & \hat{\mathbf{n}}\end{bmatrix}^\top \tag{23}$$

$$\hat{\boldsymbol{h}} = F(\hat{\mathbf{n}})\frac{\hat{\boldsymbol{\omega}}_i+\hat{\boldsymbol{\omega}}_o}{\|\hat{\boldsymbol{\omega}}_i+\hat{\boldsymbol{\omega}}_o\|_2} \tag{24}$$

$$\hat{\boldsymbol{d}} = F(\hat{\boldsymbol{h}})\hat{\boldsymbol{\omega}}_i \tag{25}$$

where $\times$ is the cross product. Finally, we encode these two directions using spherical harmonics up to degree 4 (as done in Ref-NeRF [41] for encoding view directions), concatenate the feature vector $\boldsymbol{x}$ from the field at point $\mathbf{p}$, and pass this as input to the network h.

C. Optimization and Architecture

To calculate the normal vectors of the density field, we apply a finite difference kernel, convolved with a 3×3 Gaussian smoothing kernel with $\sigma=1$, then linearly interpolate between samples to get the resulting gradient in the 3D volume. We supervise our method using photometric loss, along with the orientation loss of Equation 7. Like TensoRF, we use a learning rate of 0.02 for the rank 1 and 2 tensor components, and a learning rate of 10^{-3} for everything else. We use Adam [19] with $\beta_1=0.9,\beta_2=0.99,\varepsilon=10^{-15}$. Similar to Ref-NeRF [41], we use log-linear learning rate decay with a total decay of $d_w=10^{-3}$ and a warmup of $N_w=100$ steps and a decay multiplier of $m_w=0.1$ over $N_T=3\cdot10^4$ total iterations. This gives us the following formula for the learning rate multiplier for some iteration i:

$$\left[m_w+(1-m_w)\sin\frac{\pi}{2}\text{clip}\left(\frac{i}{N_w},0,1\right)\right]e^{i/N_T\log(d_w)} \tag{26}$$

We initialize the environment map to a constant value of 0.5. Finally, we upsample the resolution of TensoRF from 32^3 up to 300^3 cube-root-linearly at steps $500,1000,2000,3000,4000,5500,7000$, and don't shrink the volume to fit the model.

To further reduce the variance of the estimated value of the rendering equation (see Equation 15), we use quasi-random sampling sequences. Specifically, we use a Sobol sequence [36] with Owens scrambling [30], which gives the procedural sequence necessary for assigning an arbitrary number of secondary ray samples to each primary ray sample. We then apply Cranley-Patterson rotation [12] to avoid needing to redraw samples.

D. Additional Results

Tables 2-5 contain full per-scene metrics for our method as well as ablations and baselines. Visual comparisons are also provided in Figures 7-17.

PSNR ↑	teapot	toaster	car	ball	coffee	helmet	chair	lego	materials	mic	hotdog	ficus	drums	ship
PhySG[1]	35.83	18.59	24.40	27.24	23.71	27.51	21.87	17.10	18.02	19.16	24.49	15.25	14.35	18.06
NVDiffRec[1]	40.13	24.10	27.13	30.77	30.58	26.66	32.03	29.07	25.03	30.72	33.05	31.18	24.53	24.68
NVDiffRecMC[1]	37.91	21.93	25.84	28.89	29.06	25.57	28.13	26.46	25.64	29.03	30.56	25.32	22.78	18.59
Ref-NeRF[2]	47.90	25.70	30.82	47.46	34.21	29.68	35.83	36.25	35.41	36.76	37.72	33.91	25.79	30.28
Ours, no integral image	42.61	18.36	25.32	21.70	31.15	24.82	30.35	30.16	25.62	30.03	33.34	28.44	24.04	25.78
Ours, analytical derivative	43.57	21.57	27.72	22.75	31.08	28.61	30.49	30.23	28.70	31.19	33.55	27.83	24.15	25.40
Ours, single bounce	45.23	26.91	30.13	38.38	31.39	34.32	32.57	32.83	30.92	32.49	35.07	29.24	24.99	27.32
Ours, no neural	45.21	25.73	29.03	37.41	30.99	29.63	30.62	31.00	29.37	31.29	33.88	28.10	24.52	26.44
Ours	45.29	27.52	30.28	38.41	31.47	34.38	32.27	32.98	31.19	32.41	35.23	29.24	24.96	27.37

[1] requires object masks during training. [2] view synthesis method, not inverse rendering. Red is best, followed by orange, then yellow.

Table 3: **PSNR Results on the *Shiny Blender* dataset from Ref-NeRF [41] and *Blender* dataset from NeRF [27].**

SSIM ↑	teapot	toaster	car	ball	coffee	helmet	chair	lego	materials	mic	hotdog	ficus	drums	ship
PhySG[1]	.990	.805	.910	.947	.922	.953	.890	.812	.837	.904	.894	.861	.823	.756
NVDiffRec[1]	.993	.898	.938	.949	.959	.931	.969	.949	.923	.977	.973	.970	.916	.833
NVDiffRecMC[1]	.990	.842	.913	.849	.942	.877	.932	.909	.911	.961	.945	.937	.906	.732
Ref-NeRF[2]	.998	.922	.955	.995	.974	.958	.984	.981	.983	.992	.984	.983	.937	.880
Ours, no integral image	.994	.734	.895	.753	.959	.880	.946	.946	.896	.962	.954	.953	.905	.794
Ours, analytical derivative	.995	.798	.925	.790	.959	.930	.948	.943	.936	.972	.958	.950	.910	.787
Ours, single bounce	.996	.909	.951	.983	.962	.971	.964	.966	.957	.978	.969	.959	.922	.835
Ours, no neural	.996	.903	.945	.980	.959	.947	.949	.952	.945	.972	.960	.954	.916	.816
Ours	.996	.917	.951	.983	.960	.969	.956	.963	.959	.977	.964	.952	.917	.828

[1] requires object masks during training. [2] view synthesis method, not inverse rendering. Red is best, followed by orange, then yellow.

Table 4: **SSIM Results on the *Shiny Blender* dataset from Ref-NeRF [41] and *Blender* dataset from NeRF [27].**

LPIPS ↓	teapot	toaster	car	ball	coffee	helmet	chair	lego	materials	mic	hotdog	ficus	drums	ship
PhySG[1]	.022	.194	.091	.179	.150	.089	.122	.208	.182	.108	.163	.144	.188	.343
NVDiffRec[1]	.022	.180	.057	.194	.097	.134	.027	.037	.104	.033	.038	.030	.070	.208
NVDiffRecMC[1]	.029	.243	.086	.346	.131	.215	.080	.075	.096	.057	.089	.076	.096	.319
Ref-NeRF[2]	.004	.095	.041	.059	.078	.075	.017	.018	.022	.007	.022	.019	.059	.139
Ours, no integral image	.013	.285	.077	.399	.065	.180	.055	.031	.074	.042	.051	.039	.077	.180
Ours, analytical derivative	.011	.235	.053	.353	.071	.118	.052	.031	.048	.028	.047	.043	.075	.190
Ours, single bounce	.008	.114	.033	.047	.063	.050	.032	.018	.026	.020	.034	.033	.065	.135
Ours, no neural	.008	.115	.039	.058	.071	.090	.053	.026	.036	.027	.045	.036	.070	.161
Ours	.010	.104	.034	.046	.069	.055	.044	.024	.026	.022	.046	.044	.068	.149

[1] requires object masks during training. [2] view synthesis method, not inverse rendering. Red is best, followed by orange, then yellow.

Table 5: **LPIPS Results on the *Shiny Blender* dataset from Ref-NeRF [41] and *Blender* dataset from NeRF [27].**

MAE° ↓	teapot	toaster	car	ball	coffee	helmet	chair	lego	materials	mic	hotdog	ficus	drums	ship
PhySG[1]	6.634	9.749	8.844	0.700	22.514	2.324	18.569	40.244	18.986	26.053	28.572	35.974	21.696	43.265
NVDiffRec[1]	0.358	6.908	3.783	2.756	3.869	8.766	4.968	12.670	7.068	2.850	8.228	3.909	5.641	17.027
NVDiffRecMC[1]	0.551	5.746	2.020	0.592	6.490	3.518	4.659	12.570	2.428	2.807	7.449	3.535	5.439	17.865
Ref-NeRF[2]	9.234	42.870	14.927	1.548	12.240	29.484	19.852	24.469	9.531	24.938	13.211	41.052	27.853	31.707
Ours, no integral image	1.194	24.230	8.374	30.961	5.997	13.855	4.987	8.902	9.679	3.813	5.108	5.431	7.882	17.854
Ours, analytical derivative	0.879	12.994	3.444	15.734	6.534	4.983	4.463	9.331	3.294	2.607	4.936	5.247	7.059	17.222
Ours, single bounce	0.812	4.691	2.600	1.563	5.274	1.927	3.194	8.508	2.934	2.501	3.643	4.970	5.537	14.515
Ours, no neural	0.637	6.298	2.767	1.561	5.884	3.234	3.950	9.147	3.233	2.568	4.379	4.986	6.079	14.560
Ours	0.752	4.474	2.598	1.563	5.352	1.924	3.195	8.452	2.868	2.523	3.546	4.949	5.326	14.726

[1] requires object masks during training. [2] view synthesis method, not inverse rendering. Red is best, followed by orange, then yellow.

Table 6: **MAE Results on the *Shiny Blender* dataset from Ref-NeRF [41] and *Blender* dataset from NeRF [27].**

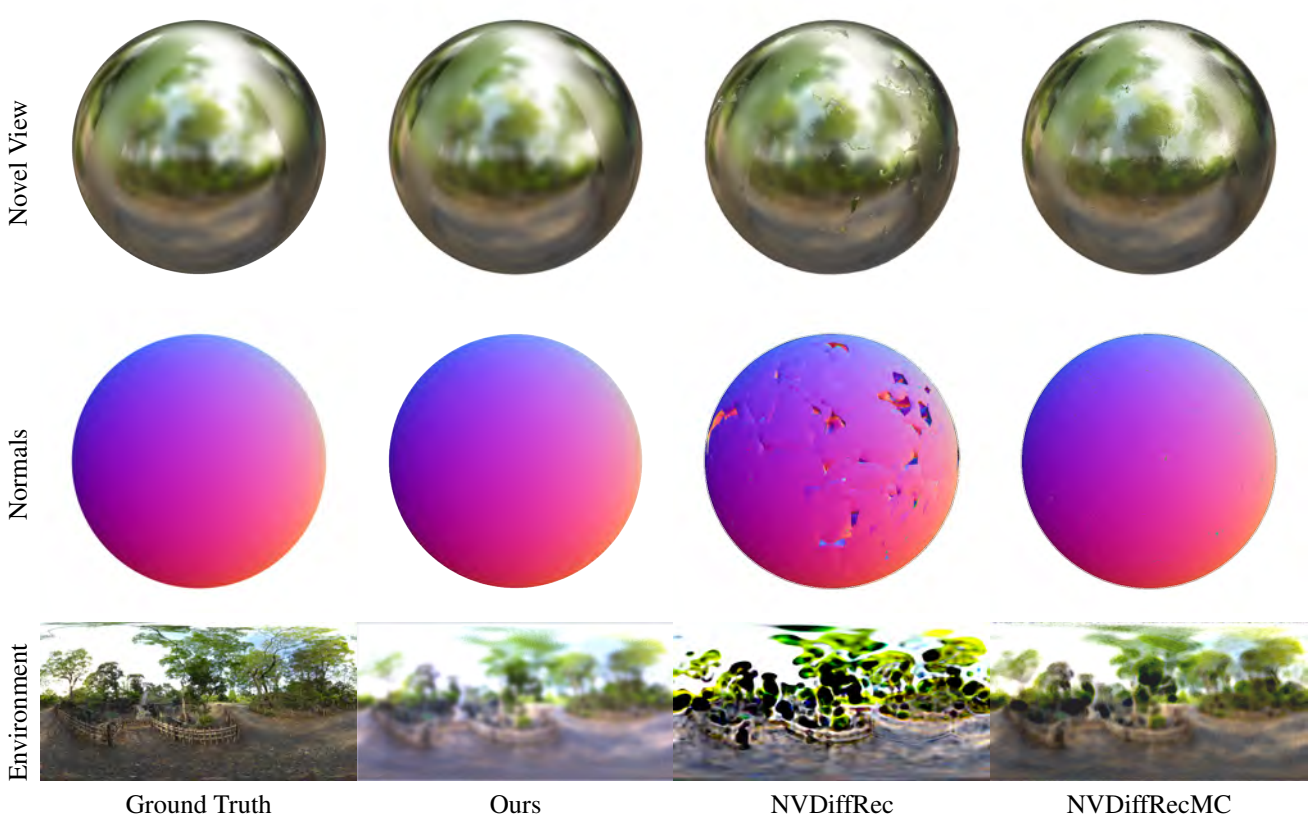

Figure 7: **Results on the *ball* scene**, compared to NVDiffRec [29] and NVDiffRecMC [17].

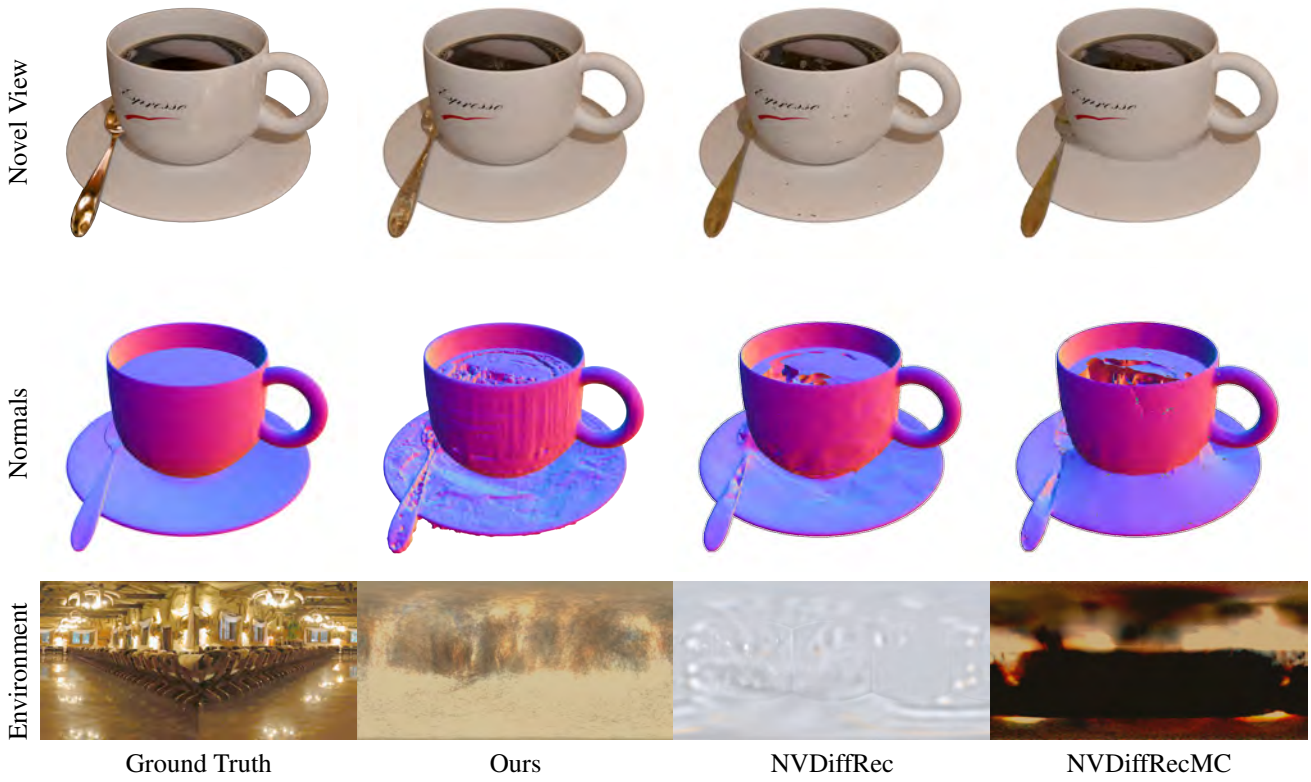

Figure 8: **Results on the *coffee* scene**, compared to NVDiffRec [29] and NVDiffRecMC [17].

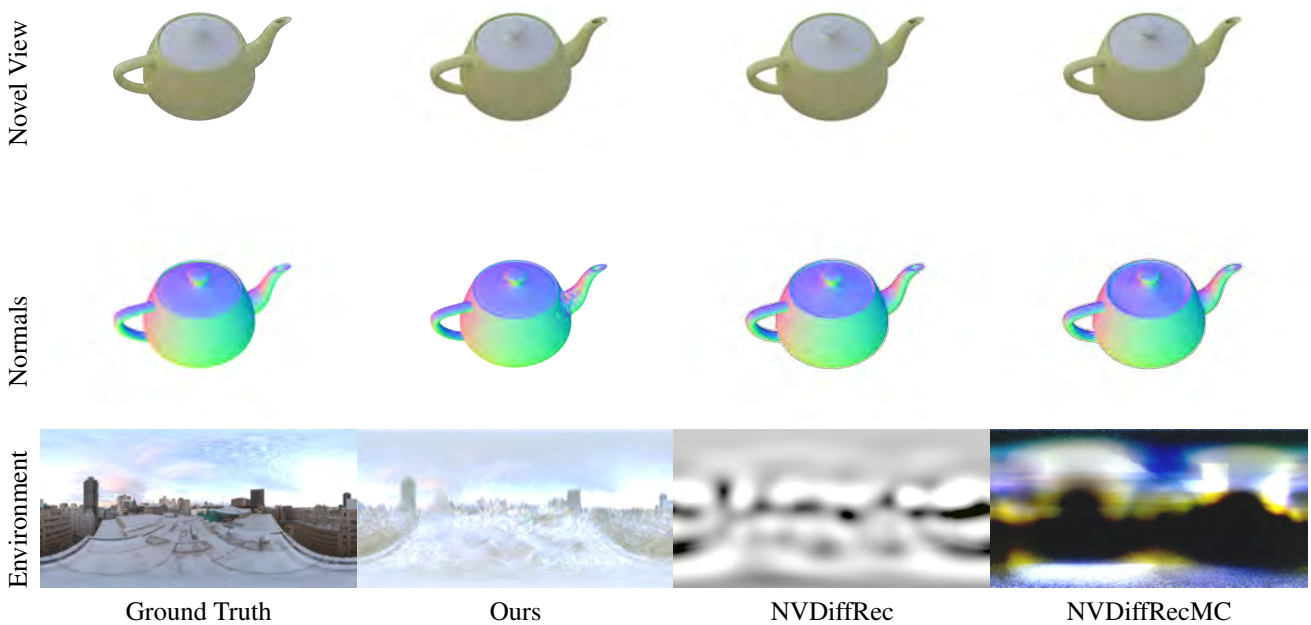

Figure 9: **Results on the *teapot* scene**, compared to NVDiffRec [29] and NVDiffRecMC [17].

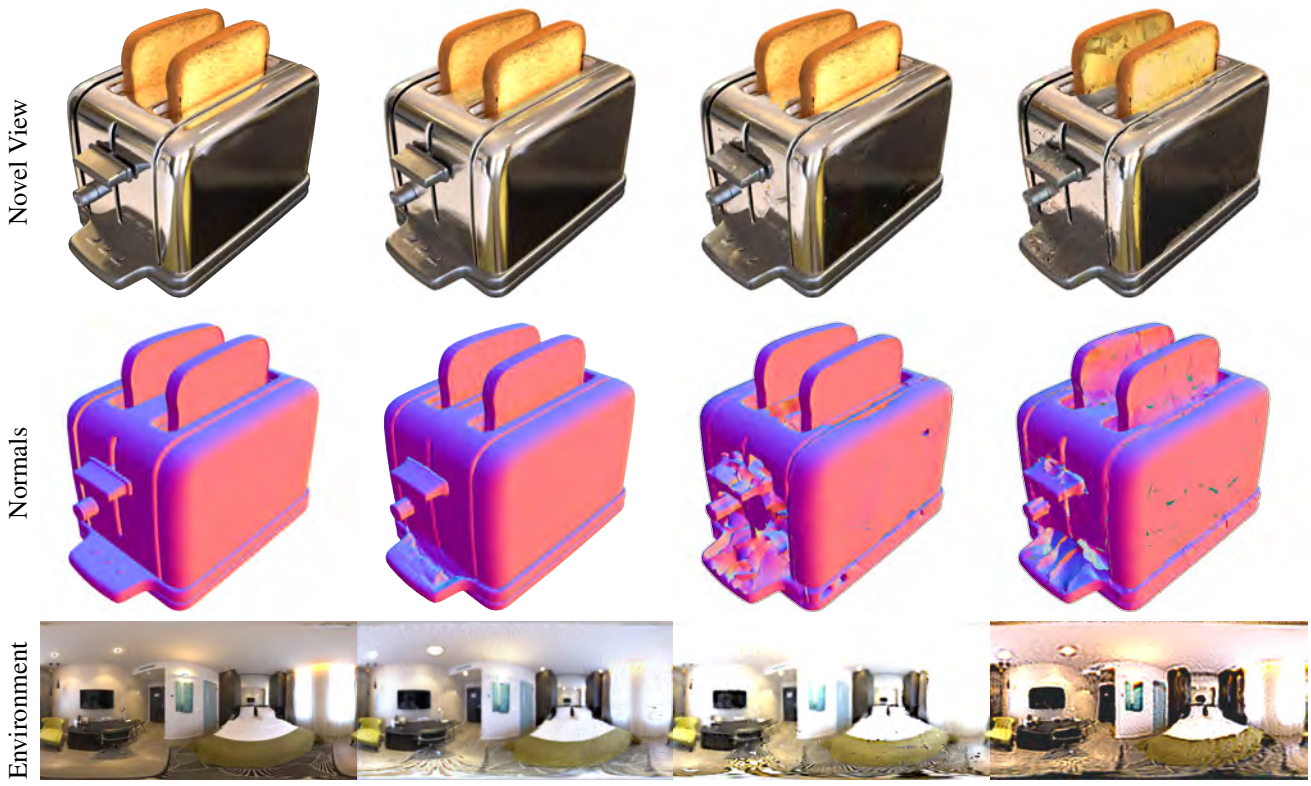

Figure 10: **Results on the *toaster* scene**, compared to NVDiffRec [29] and NVDiffRecMC [17].

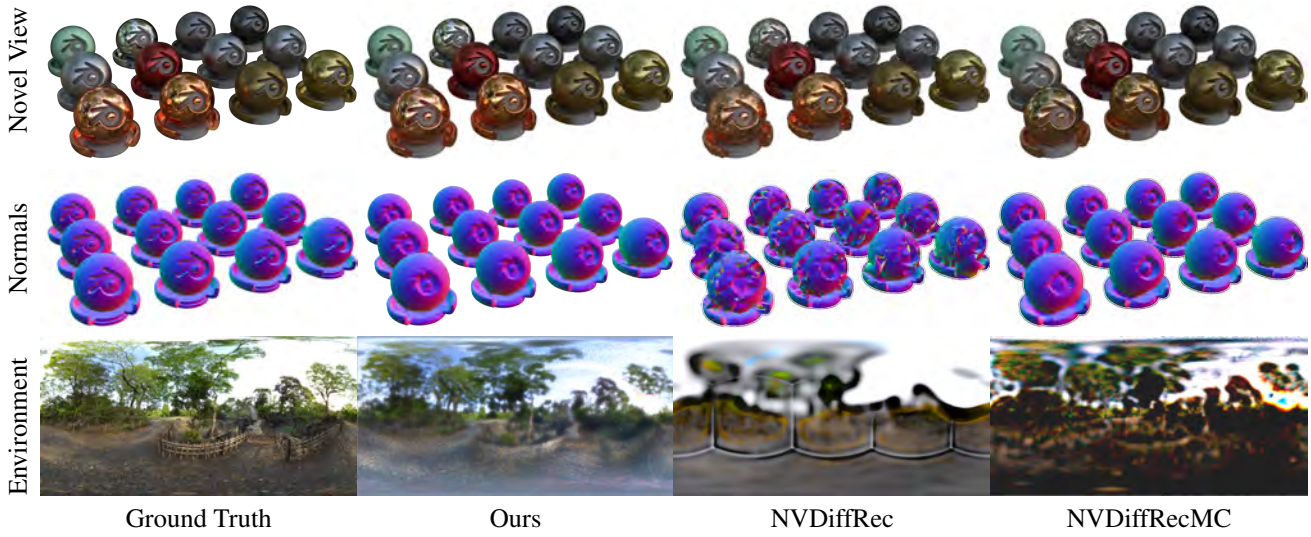

Figure 11: **Results on the *materials* scene**, compared to NVDiffRec [29] and NVDiffRecMC [17].

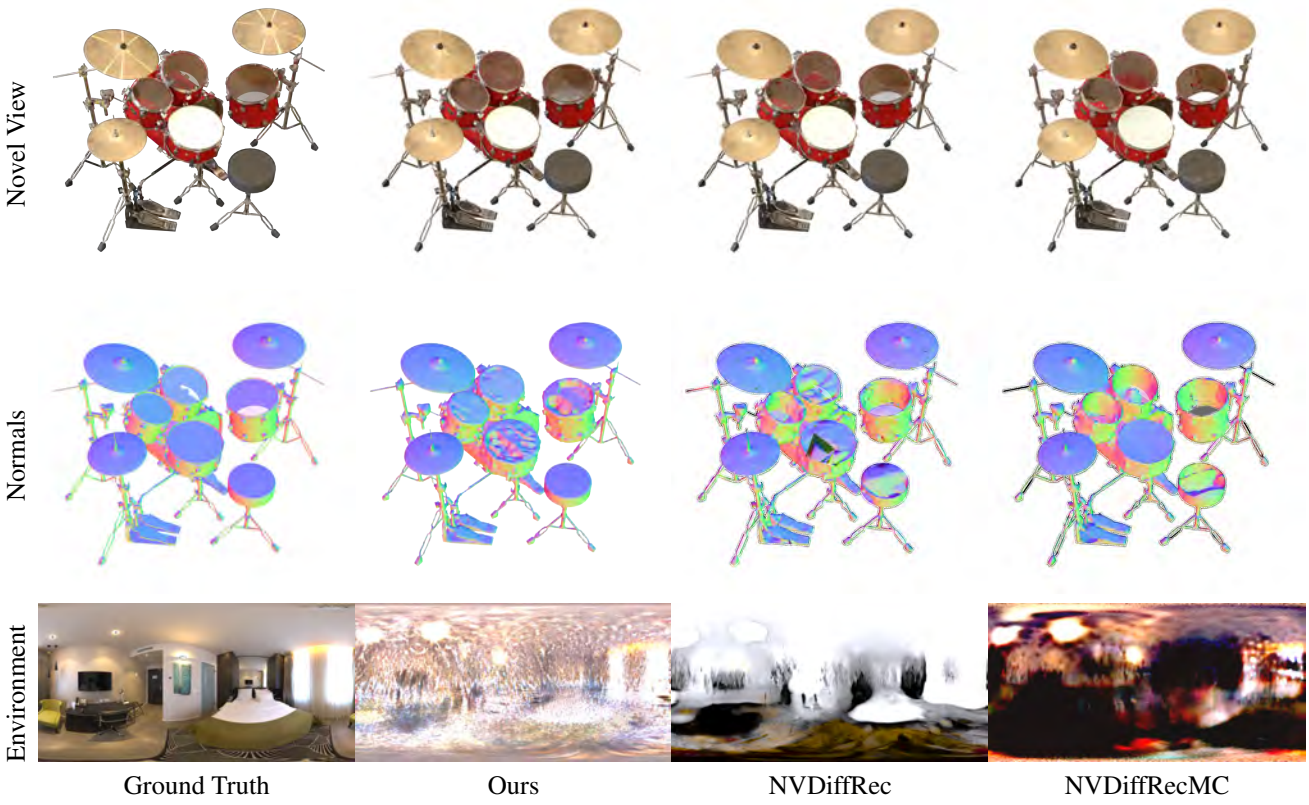

Figure 12: **Results on the *drums* scene**, compared to NVDiffRec [29] and NVDiffRecMC [17].

Figure 13: **Results on the *ficus* scene**, compared to NVDiffRec [29] and NVDiffRecMC [17].

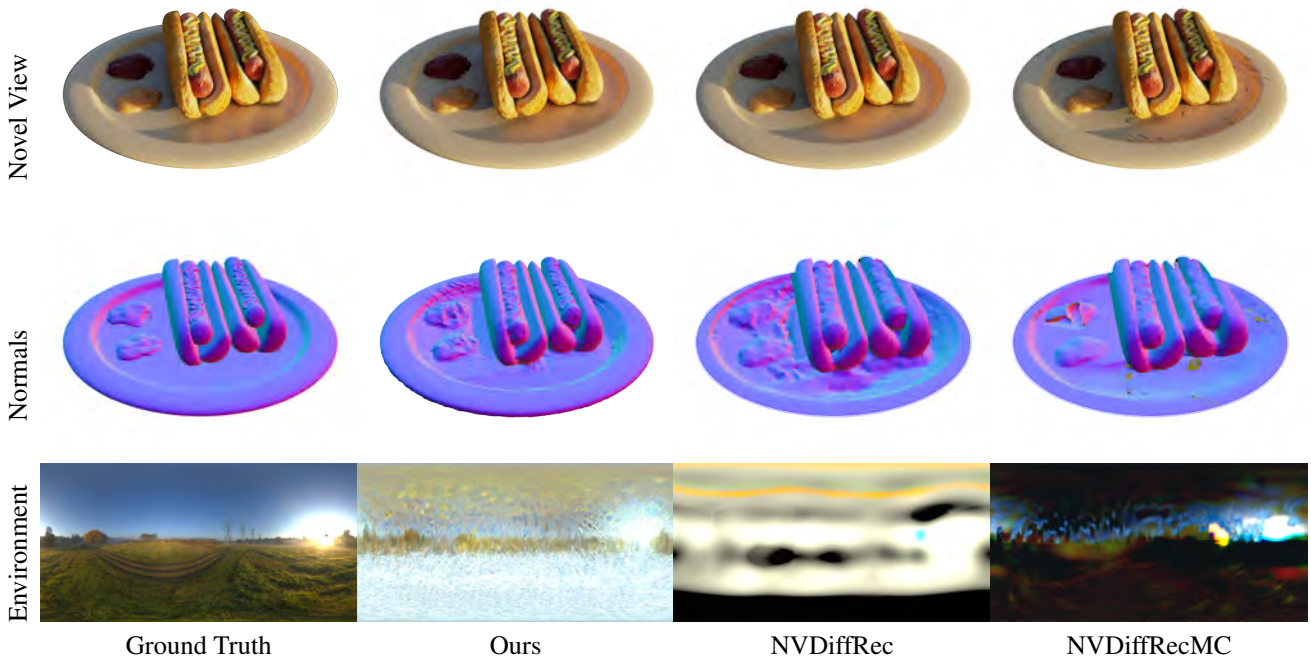

Figure 14: **Results on the *hotdog* scene**, compared to NVDiffRec [29] and NVDiffRecMC [17].

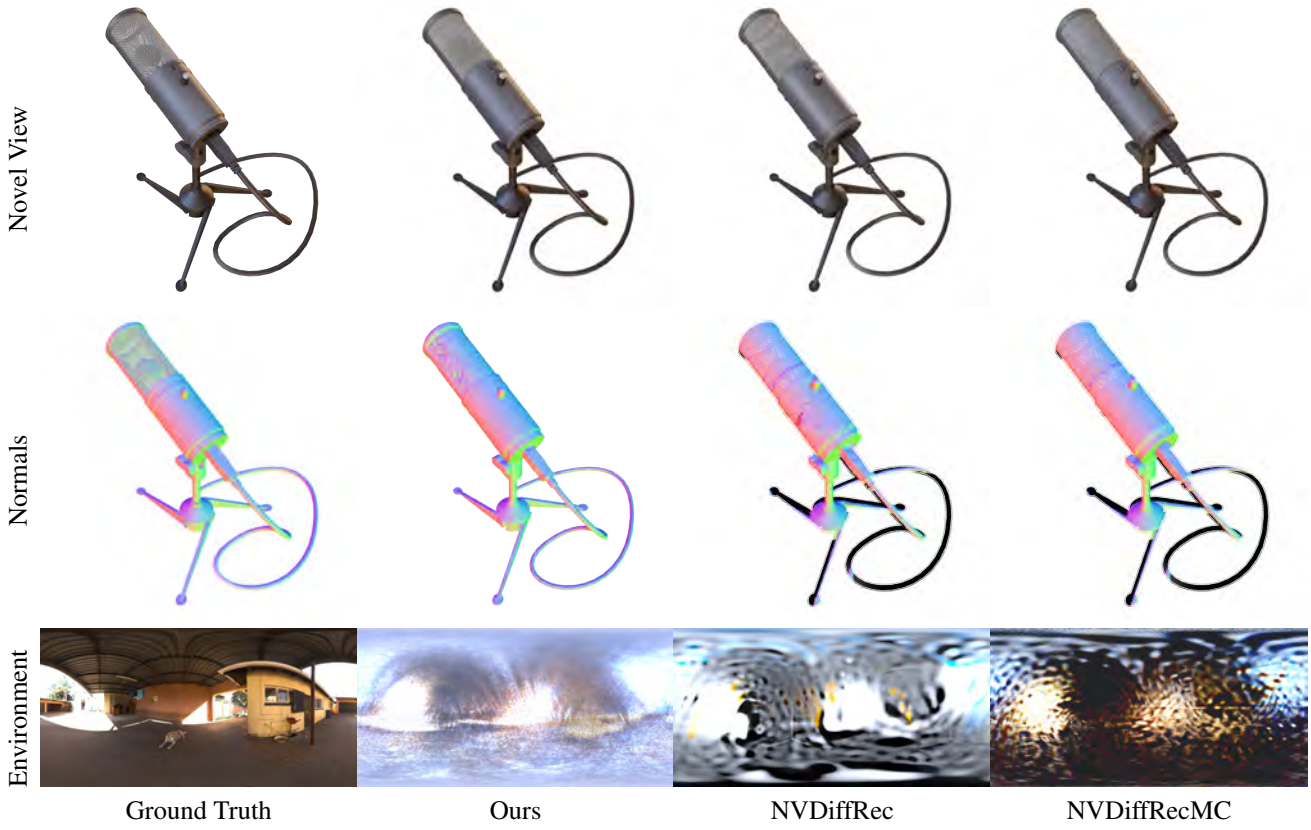

Figure 15: **Results on the *mic* scene**, compared to NVDiffRec [29] and NVDiffRecMC [17].

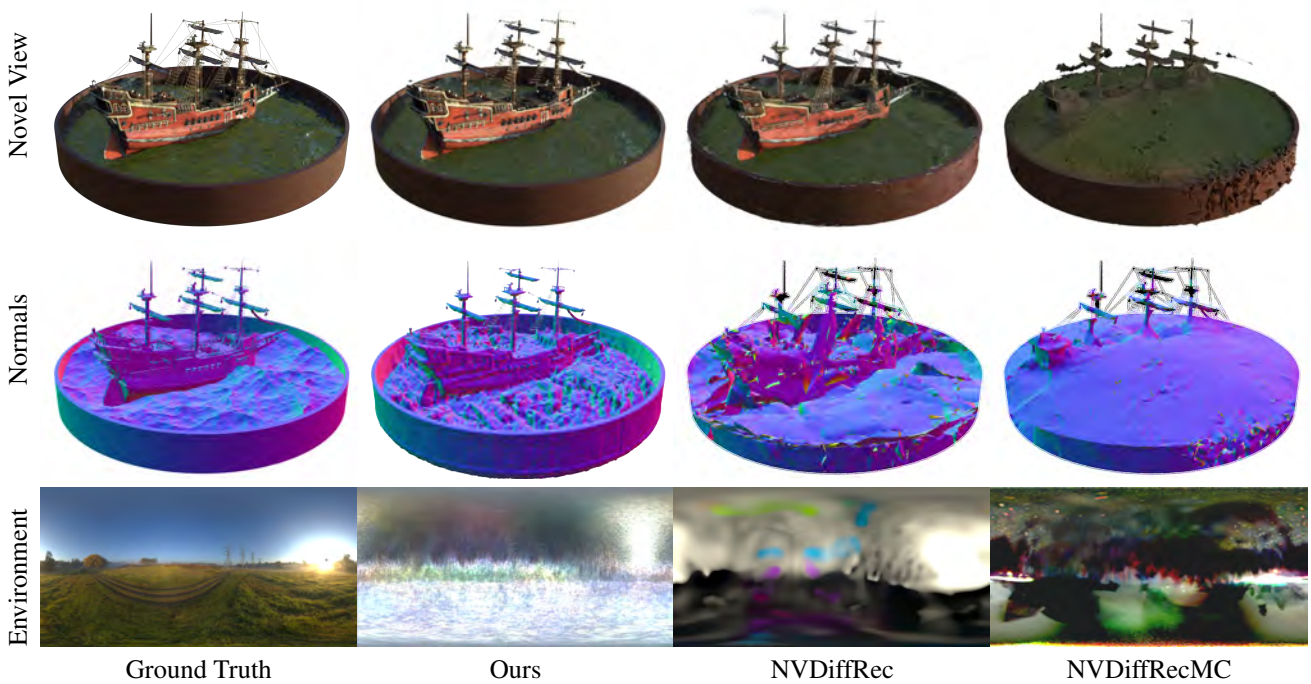

Figure 16: **Results on the *ship* scene**, compared to NVDiffRec [29] and NVDiffRecMC [17]. Since our method, NVDiffRec, and NVDiffRecMC do not model refraction, they are not able to handle the water well.

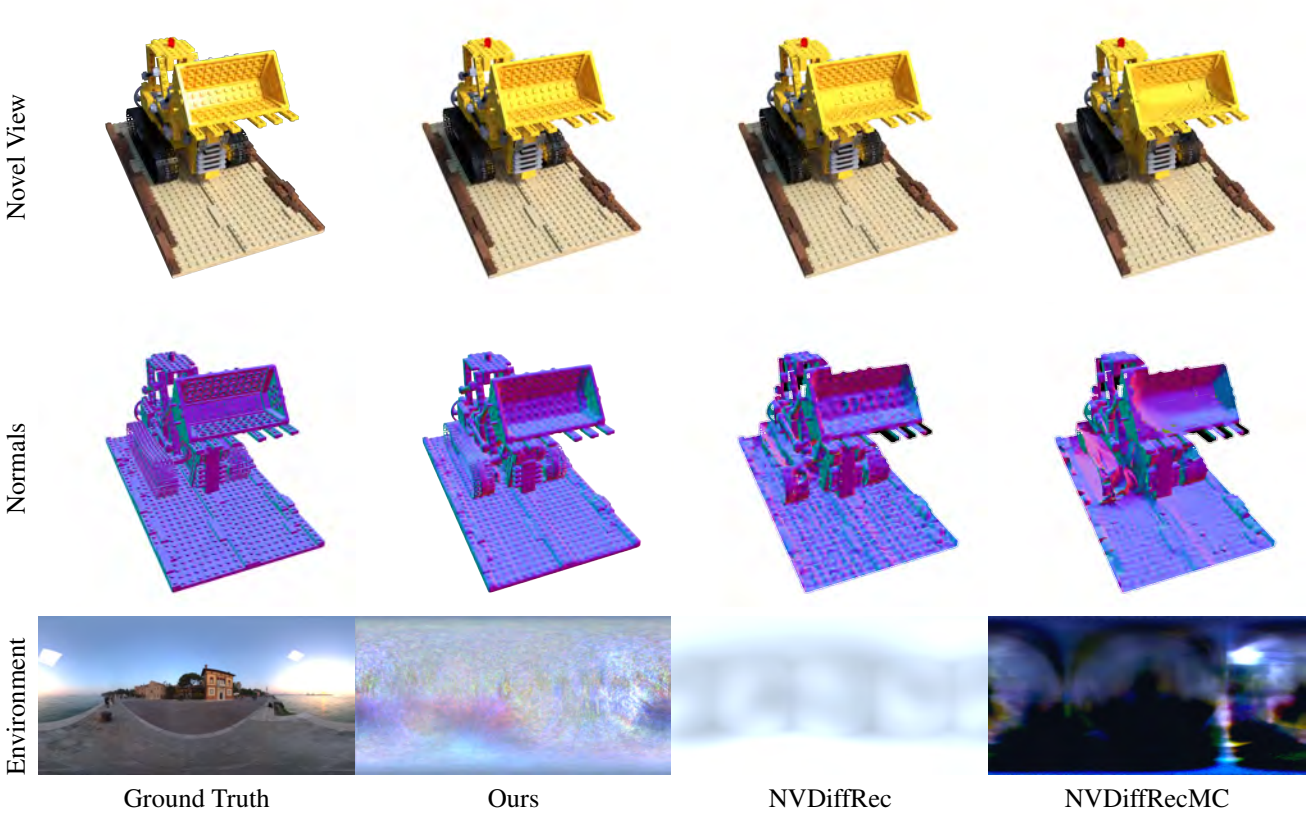

Figure 17: **Results on the *lego* scene**, compared to NVDiffRec [29] and NVDiffRecMC [17].