SpecNeRF: Gaussian Directional Encoding for Specular Reflections
Supplementary Material

A. Supplementary Video

For more information regarding the method, please visit our project website at https://limacv.github.io/SpecNeRF_web/. We also provide a supplementary video for visual comparisons under a moving camera trajectory, which can be accessed at https://youtu.be/3nUooe3pVA0. We highly encourage readers to watch our video, where our method produces results with better specular reflection reconstruction.

B. Gaussian Directional Encoding Proofs

Recall that we define each Gaussian as:

$$\mathcal{G}(\mathbf{x}) = \exp\Big(-\left\|\mathcal{Q}(\mathbf{x}-\boldsymbol{\mu};\mathbf{q})\odot\boldsymbol{\sigma}^{-1}\right\|_2^2\Big), \quad (11)$$

where $\boldsymbol{\mu}$ is the position and $\boldsymbol{\sigma}$ the scale of the Gaussian. $\mathcal{Q}(\mathbf{x};\mathbf{q})$ applies the quaternion rotation $\mathbf{q}$ to a 3D vector $\mathbf{x}$. For ease of notation, we omit the subscript i (compared to the main paper) as the same equation is applied to every Gaussian. In practice, we optimize the inverse scale $\boldsymbol{\psi} = \boldsymbol{\sigma}^{-1}$ instead of directly using $\boldsymbol{\sigma}$, to improve numerical stability.

We further define the basis function $\mathcal{P}(\mathbf{o},\mathbf{d})$ over a given ray $\mathbf{o}+t\mathbf{d}$ with the Gaussian parameters $(\boldsymbol{\mu},\boldsymbol{\sigma},\mathbf{q})$ as:

$$\mathcal{P}(\mathbf{o},\mathbf{d}) = \max_{t\geq 0}\mathcal{G}(\mathbf{o}+t\mathbf{d}). \quad (12)$$

We start by applying the following variable substitution that converts the ray origin $\mathbf{o}$ and direction $\mathbf{d}$ from world-space into the space of the Gaussian (origin $\overline{\mathbf{o}}$ and direction $\overline{\mathbf{d}}$):

$$\overline{\mathbf{o}} = \mathcal{Q}(\mathbf{o}-\boldsymbol{\mu};\mathbf{q})\odot\boldsymbol{\psi}, \quad (13)$$
$$\overline{\mathbf{d}} = \mathcal{Q}(\mathbf{d};\mathbf{q})\odot\boldsymbol{\psi}. \quad (14)$$

It follows that

$$\begin{aligned}\mathcal{G}(\mathbf{o}+t\mathbf{d}) &= \exp\Big(-\left\|\mathcal{Q}(\mathbf{o}+t\mathbf{d}-\boldsymbol{\mu};\mathbf{q})\odot\boldsymbol{\psi}\right\|_2^2\Big) & (15)\\ &= \exp\Big(-\left\|\overline{\mathbf{o}}+t\overline{\mathbf{d}}\right\|_2^2\Big) & (16)\\ &= \exp\Big(-\overline{\mathbf{o}}^\top\overline{\mathbf{o}} - 2\overline{\mathbf{o}}^\top\overline{\mathbf{d}}t - \overline{\mathbf{d}}^\top\overline{\mathbf{d}}t^2\Big). & (17)\end{aligned}$$

Since the exponential function is monotonic, $\mathcal{G}$ is maximized when the quadratic function (in t)

$$f(t) = -\overline{\mathbf{o}}^\top\overline{\mathbf{o}} - 2\overline{\mathbf{o}}^\top\overline{\mathbf{d}}t - \overline{\mathbf{d}}^\top\overline{\mathbf{d}}t^2 \quad (18)$$

reaches its maximum. Since the quadratic coefficient, $-\overline{\mathbf{d}}^\top\overline{\mathbf{d}}$, is negative for any non-zero vector $\overline{\mathbf{d}}$, $f(t)$ reaches its maximum when $f'(t)=0$, i.e. for $t = t_0 = -\frac{\overline{\mathbf{o}}^\top\overline{\mathbf{d}}}{\overline{\mathbf{d}}^\top\overline{\mathbf{d}}}$. Furthermore, $\mathcal{G}(\mathbf{o}+t\mathbf{d})$ monotonically decreases for $t \geq t_0$. Given that $t\geq 0$, when $t_0 \leq 0$, the $\mathcal{G}(\mathbf{o}+t\mathbf{d})$ reaches maximum always at $t=0$. To sum up, the maximum value of $\mathcal{G}(\mathbf{o}+t\mathbf{d})$ falls into the following two cases:

$$\max_{t\geq 0}\mathcal{G}(\mathbf{o}+t\mathbf{d}) = \begin{cases}\exp\Big(-\left\|\overline{\mathbf{o}}+t_0\overline{\mathbf{d}}\right\|_2^2\Big) & t_0 > 0\\ \exp\Big(-\left\|\overline{\mathbf{o}}\right\|_2^2\Big) & \text{otherwise,}\end{cases} \quad (19)$$

By substituting $-\frac{\overline{\mathbf{o}}^\top\overline{\mathbf{d}}}{\overline{\mathbf{d}}^\top\overline{\mathbf{d}}}$ for t_0, this equation is the same as Equation 6 in the main paper.

C. Implementation details

Figure 12 zooms into our model's network architecture and clarifies the role of each used MLP.

C.1. Model Structure

We list the model structure parameters in Table 3. We use separate MLP heads to predict each property at each sample location. Note that we use a lower resolution configuration for normal hash encoding, because we find that constraining the smoothness of the normal stabilizes the optimization process and leads to better specular reflection reconstruction.

Normal parameterization. To predict normals, we first output a 3-element vector $\mathbf{n}'_{\text{raw}}$ using the normal MLP without any output activation and normalize it to get the predicted normal $\mathbf{n}' = \mathbf{n}'_{\text{raw}}/\left\|\mathbf{n}'_{\text{raw}}\right\|_2$. However, in practice, we find that this will occasionally lead to a normal flipping issue when $\mathbf{n}'_{\text{raw}}$ is numerically small and $\mathbf{n}'$ will flip its direction with only a very small deviation of $\mathbf{n}'_{\text{raw}}$ during training. Figure 13 visualizes this issue. The flip of the predicted normal will further lead to suboptimal normals derived from the density gradient due to the normal prediction loss $\mathcal{L}_{\text{norm}}$. To alleviate this normal flip issue, we correct the direction of the predicted normal by forcing the angle between the final normal $\mathbf{n}'$ and the view direction to be smaller than 90°:

$$\mathbf{n}' = -\operatorname{sign}(\mathbf{d}\cdot\mathbf{n}'_{\text{raw}})\frac{\mathbf{n}'_{\text{raw}}}{\left\|\mathbf{n}'_{\text{raw}}\right\|_2}, \quad (20)$$

where $\mathbf{d}$ is the ray direction. We can see from Figure 13 that this normal correction operation helps us prevent the normal flip, and yields a better normal prediction.

C.2. Training and Rendering Configuration

We use the Adam optimizer [21] to train our NeRF model using the default parameter configurations in PyTorch [37] for the optimizer, except that we set the learning rate to 0.005. When rendering a pixel, we first shoot rays from

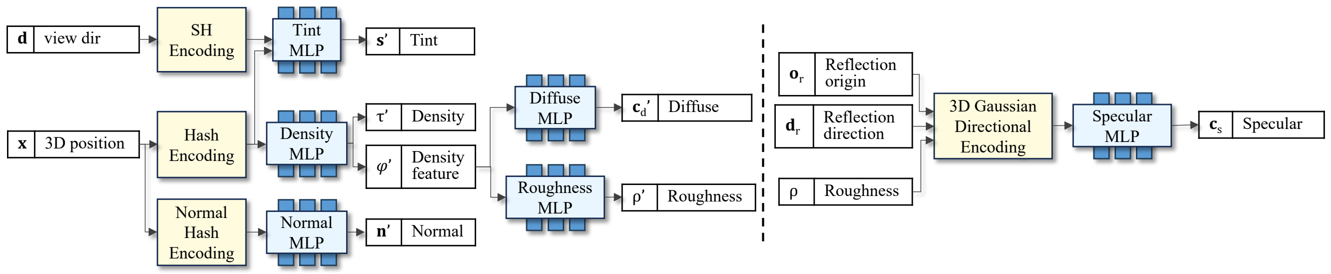

Figure 12. We zoom in the MLPs and some important modules as in Figure 2. The detailed module configurations are shown in Table 3.

Table 3. The value for each parameter. The module names are consistent with those shown in Figure 12. For all MLPs, we use ReLU activations in hidden layers.

Module	Configuration	Value
SH Encoding	Order	3
Tint MLP	# of hidden layer	2
	# of neuron per layer	64
	Output activation	Sigmoid
Hash Encoding	# of levels	16
	Hash table size	2^{22}
	# of feature dim. per entry	2
	Coarse resolution	128
	Scale factor per level	1.4
Density MLP	# of hidden layer	1
	# of neuron per layer	64
	Output activation	Exp
	Density feature dim.	16
Diffuse MLP	# of hidden layer	2
	# of neuron per layer	64
	Output activation	Sigmoid
Roughness MLP	# of hidden layer	2
	# of neuron per layer	64
	Output activation	Softplus
Normal Hash Encoding	# of levels	4
	Hash table size	2^{19}
	# of feature dim. per entry	4
	Coarse resolution	16
	Scale factor per level	1.5
Normal MLP	# of hidden layer	1
	# of neuron per layer	64
	Output activation	None
Specular MLP	# of hidden layer	2
	# of neuron per layer	64
	Output activation	Sigmoid

the camera origin to the pixel locations, and then sample points along each ray. Similar to Nerfstudio [44], we use two levels of proposal sampling, guided by two density fields.

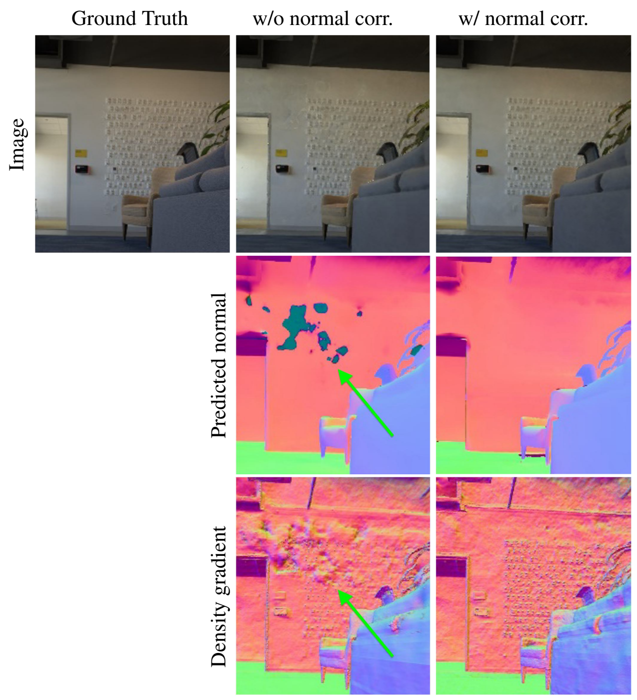

Figure 13. An example of the normal flip issue. As indicated by the green arrow, the predicted normal is occasionally flipped due to small perturbation during training, which leads to artifacts in rendering images and the density gradient. Our normal correction (normal corr.) prevents the flip issue by optionally reversing the normal direction based on the view direction.

Specifically, in the first round, we sample 256 points using exponential distance. We set the far distance to a constant value of 800 meters, and we determine the near distance for each scene using the minimum distance between all the structure-from-motion points and the viewing cameras. Then, in each iteration of the proposal sampling process, we feed the samples into the proposal network sampler and generate new samples based on the integration weights of the input samples. We sample 96 samples in the first iteration of the proposal process, followed by 48 samples in the second. The model structures of the proposal networks follow those in the "nerfactor" model in Nerfstudio [44].

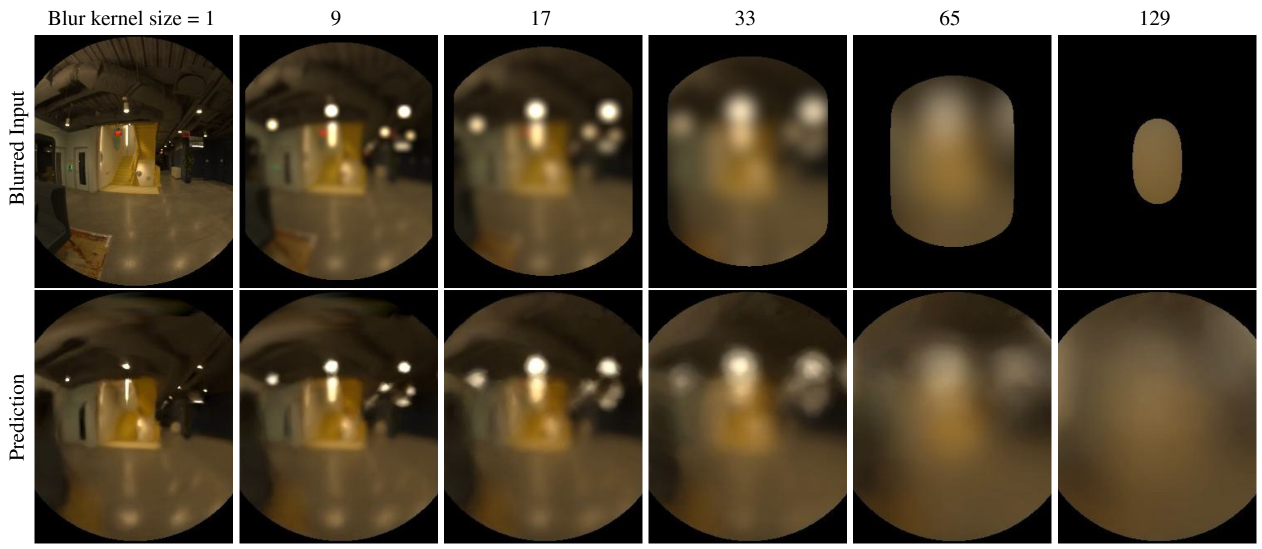

Figure 14. One example of the Gaussian initialization input (top) and predictions (bottom) for different scales.

C.3. Gaussian Parameter Optimization

To obtain optimal parameters for the Gaussian directional encoding, we use an initialization stage to seed the Gaussian parameters and specular MLP weights. The process is illustrated in Figure 15. We optimize a preconvolved incident light field composed of our 3D Gaussian directional encoding and the Specular MLP. We first apply a range of Gaussian blurs to all input images using a series of standard deviations, generating pyramids of blurry input images. In our experiments, we first scale the input images to have 360 pixels along the longest axis. Then, we apply OpenCV's `GaussianBlur` [8] with kernel sizes (1, 3, 5, 9, 17, 33, 65, 129). Regions that involve the image border during blurring are marked as invalid, resulting in a wider invalid border with larger kernel size. Figure 14 showcases one example view with some of the blur kernels used.

All valid blurred pixels compose our ray dataset for the initialization stage. In each iteration, we sample 25,600 pixels from the ray dataset, and generate the corresponding ray origin $\mathbf{o}$, direction $\mathbf{d}$, and the blur kernel size k. We train the Gaussian parameters and Specular MLP using Adam [21] with a learning rate of 0.001, and leave other parameters as default. We supervise the output color using the corresponding blurry color in the Gaussian pyramid using an L1 loss. We find this small network converges quickly, thus we only train for 8,000 iterations, which takes around half an hour to finish on one NVIDIA A100 GPU. We visualize the fitted preconvolved incident light field in Figure 14. The reconstructed preconvolved light field well-represents the input with multiple blur levels. We also visualize the fitted Gaussian blobs of two scenes in Figure 16. We can see that some Gaussian blobs are aligned with the underlying objects (e.g. the lamp on the ceiling).

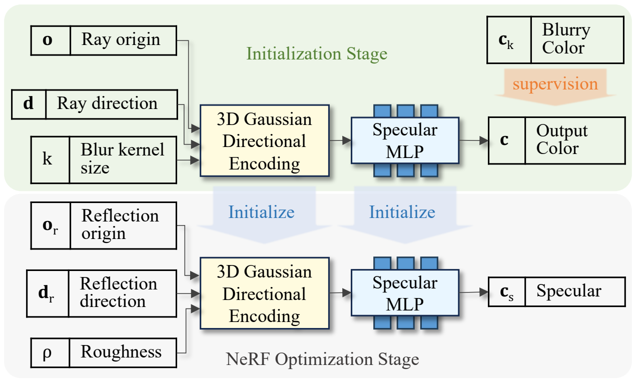

Figure 15. Illustration of the initialization stage. We optimize the Gaussian parameters and the Specular MLP using the Gaussian-blurred input images, and then use them as initialization for the NeRF optimization stage.

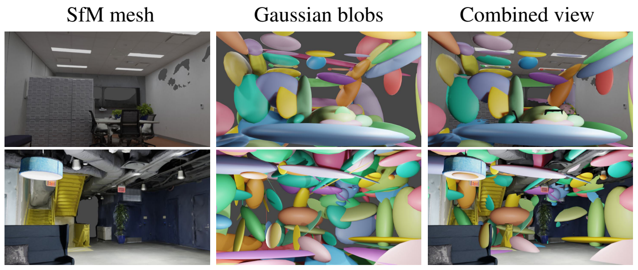

Figure 16. Visualization of the learned Gaussian blobs for two scenes. We assign a random color for each Gaussian blob for better visibility.

C.4. Losses

Recall that in our experiments, the final loss is a combination of several terms:

$$\mathcal{L} = \mathcal{L}_{\mathrm{c}} + \mathcal{L}_{\mathrm{prop}} + \lambda_{\mathrm{dist}}\mathcal{L}_{\mathrm{dist}} + \lambda_{\mathrm{mono}}\mathcal{L}_{\mathrm{mono}} + \lambda_{\mathrm{norm}}\mathcal{L}_{\mathrm{norm}}. \quad (21)$$

In this section, we follow the notation that $i = 1, ..., N$ is the sample point index along a ray. We omit the ray index as each loss term has the same form for all rays. The loss term is averaged over all rays within a training batch.

Reconstruction Loss. The reconstruction loss $\mathcal{L}_{\text{c}}$ is the L1 norm between the predicted color $\mathbf{c}$ and the ground-truth color $\mathbf{c}_{\text{gt}}$:

$$\mathcal{L}_{\text{c}} = \|\mathbf{c} - \mathbf{c}_{\text{gt}}\|_1 . \tag{22}$$

For the Eyeful Tower dataset, we compute the reconstruction loss in the Perceptual Quantizer (PQ) color space, as in VR-NeRF [53]. For other public datasets, we use the standard sRGB color space.

Proposal Loss and Distortion Loss. The proposal loss $\mathcal{L}_{\text{prop}}$ supervises the density field of the proposal network to be consistent with that of the main NeRF. The distortion loss $\mathcal{L}_{\text{dist}}$ is a regularization term for the density field of the main NeRF. It consolidates the volumetric blending weights into as small a region as possible. Please refer to Barron et al. [3] for the detailed definitions and explanations of both losses.

Normal Prediction Loss. We encourage the predicted normals from the normal MLP to be consistent with the underlying geometry of NeRF. For this, we use a normal prediction loss $\mathcal{L}_{\text{norm}}$ that supervises the normal $\mathbf{n}'_i$ predicted for every sample point using the normal MLP and NeRF density gradient $\mathbf{g}_i$:

$$\mathcal{L}_{\text{norm}} = \frac{1}{N}\sum_i \left\| \mathbf{n}'_i - \frac{-\mathbf{g}_i}{\|\mathbf{g}_i\|} \right\| . \tag{23}$$

To compute the gradient of the density τ' with respect to the input world coordinate $\mathbf{x} = (x, y, z)$, we could use the analytical gradient, which is natively supported by PyTorch [37]. However, we model the density field using a hash-grid-based representation, which is prone to noisy gradients and has poor optimization performance [28]. Therefore, we adopt a modified version of the numerical gradient from Neuralangelo [28]. To compute the gradient along the x-axis, we use

$$\nabla_x \tau' = \frac{\tau'(\mathbf{x} + \boldsymbol{\epsilon}_x) - \tau'(\mathbf{x} - \boldsymbol{\epsilon}_x)}{2\epsilon}, \tag{24}$$

where $\boldsymbol{\epsilon}_x = (\epsilon, 0, 0)$. The equations for computing the gradient along the y- and z-axes can be derived analogously. Overall, $\nabla_{\mathbf{x}}\tau$ involves sampling six additional points to query the density value. Instead of predefining the schedule of the ϵ value during training, we compute a per-sample ϵ value that is consistent with the cone tracing radius at the sample location: $\epsilon = t \cdot r$. Here, t is the ray-marching distance of the sample point, and r is the base radius of a pixel at unit distance along the ray.

Additional Losses. For the Eyeful Tower dataset, we also deploy a depth supervision loss and an "empty around camera" loss, following VR-NeRF [53]. For the depth loss, we supervise the NeRF depth with the depth from structure-from-motion mesh using L1 distance in the first 500 iterations. For the "empty around camera" loss, we randomly sample 128 points in the unit sphere around training cameras, and regularize the density value to be zero. This reduces the near-plane ambiguity as shown in FreeNeRF [55]. We set the weights of the depth loss and "empty around camera" loss to 0.1 and 10, respectively.

D. Physical Interpretation of 3D Gaussians

Though a 3D Gaussian blob may appear similar to a point light source, we would like to emphasize that the 3D Gaussians do not represent explicit light sources, nor are they specifically designed for modeling direct light alone. Instead, they serve as basis functions for representing the scene's full 5D specular radiance field, including global illumination effects. One example can be seen in Figure 17. This is analogous to how spherical Gaussians (SGs) represent a 2D environment map.

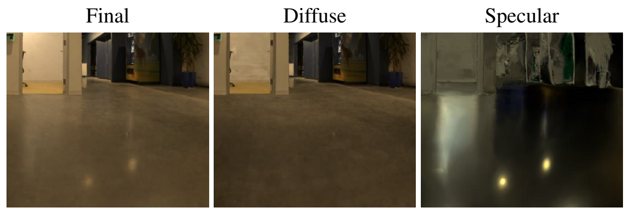

Figure 17. Our 3D Gaussians can model global illumination effects. This is evident on the floor, where the indirect light from the room is captured and represented through the specular component.

E. Additional Experiments

E.1. Number of Gaussians

We test the GPU memory usage of our Gaussian directional encoding and the specular MLP under a series of Gaussians,

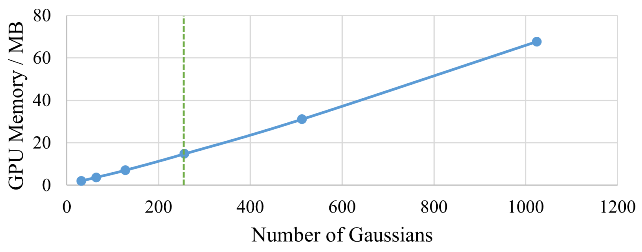

Figure 18. The GPU memory consumption of the Gaussian directional encoding and the Specular MLP with various number of Gaussians. We test GPU memory with a batch size of 12,800 rays. The green dashed line is the configuration used in our experiments.

Table 4. Quantitative comparisons on the Shiny Blender dataset [47]. Our approach demonstrates comparable performance to Ref-NeRF since the dataset assumes perfect 2D lighting conditions.

Methods	PSNR ↑	SSIM ↑	LPIPS ↓
Ours	34.65	0.9615	0.0515
Ref-NeRF [47]	34.69	0.9619	0.0508

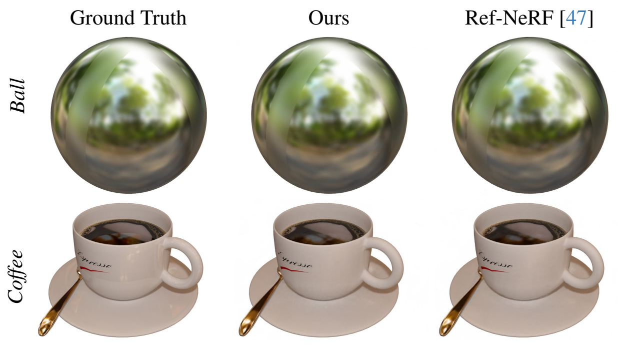

Figure 19. Qualitative comparisons of two example test views from Shiny Blender dataset [47].

and visualize the results in Figure 18. We can see that our reflection model adds very little GPU memory overhead compared to the approximately 8 GB of overall memory used for training the whole pipeline.

E.2. Shiny Blender Dataset

We evaluate our method and the Ref-NeRF baseline on the Shiny Blender dataset [47]. We utilize a re-implementation of Ref-NeRF in NeRF-Factory [17]. To ensure a fair comparison, we adopt the same MLP backbone as used in NeRF-Factory. We train both methods for 80,000 iterations for each scene. The visual results are shown in Figure 19 and the quantitative results are depicted in Table 4. Our method achieves comparable performance to Ref-NeRF, which is expected because all scenes in the dataset are lit by perfect 2D (far-field) environment light. Our method outperforms Ref-NeRF under near-field lighting scenes as shown in the paper.

E.3. Synthetic Dataset

We compare our method with several baselines on the FIPT synthetic dataset [49]. In addition to the baselines described in the main paper, we also compare with FIPT [49], a state-of-the-art path-tracing-based inverse rendering approach. We report the average PSNR, SSIM and LPIPS metrics for novel-view synthesis. Since we have the ground-truth mesh for the synthetic dataset, we also report the mean angular error (MAE) used in Ref-NeRF [47] for evaluating the estimated normal accuracy. The results in Table 5 show that our method achieves the best novel-view synthesis quality and geometry accuracy. Interestingly, despite the use of ground-truth geometry for the physically based inverse rendering approach, the novel-view synthesis is worse than any NeRF-based baseline by a large margin. This suggests that introducing a fully physically based rendering model may be a disadvantage when it comes to novel-view synthesis quality, at least compared to NeRF-like approaches that are tailored specifically for the view synthesis task.

Table 5. Quantitative comparisons of novel-view synthesis and geometry quality on the FIPT synthetic dataset. Our method achieves the best view synthesis quality, and is most accurate in terms of geometry. We highlight the best numbers in **bold**.

Methods	PSNR↑	SSIM↑	LPIPS↓	MAE°↓
Ours	**32.043**	**0.8657**	0.1266	**16.09**
NeRF [34]	31.621	0.8586	0.1325	34.16
Ref-NeRF [47]	31.952	0.8650	**0.1250**	18.76
MS-NeRF [57]	31.441	0.8534	0.1345	42.19
FIPT [49]	28.322	0.6922	0.1379	$0^{\dagger}$

†Note that FIPT uses the ground-truth geometry.

E.4. Additional Results

We show additional comparisons and decomposition results in Figure 20 and Figure 21. Our method achieves the best visual quality as well as the predicted normal quality for specular reflections.

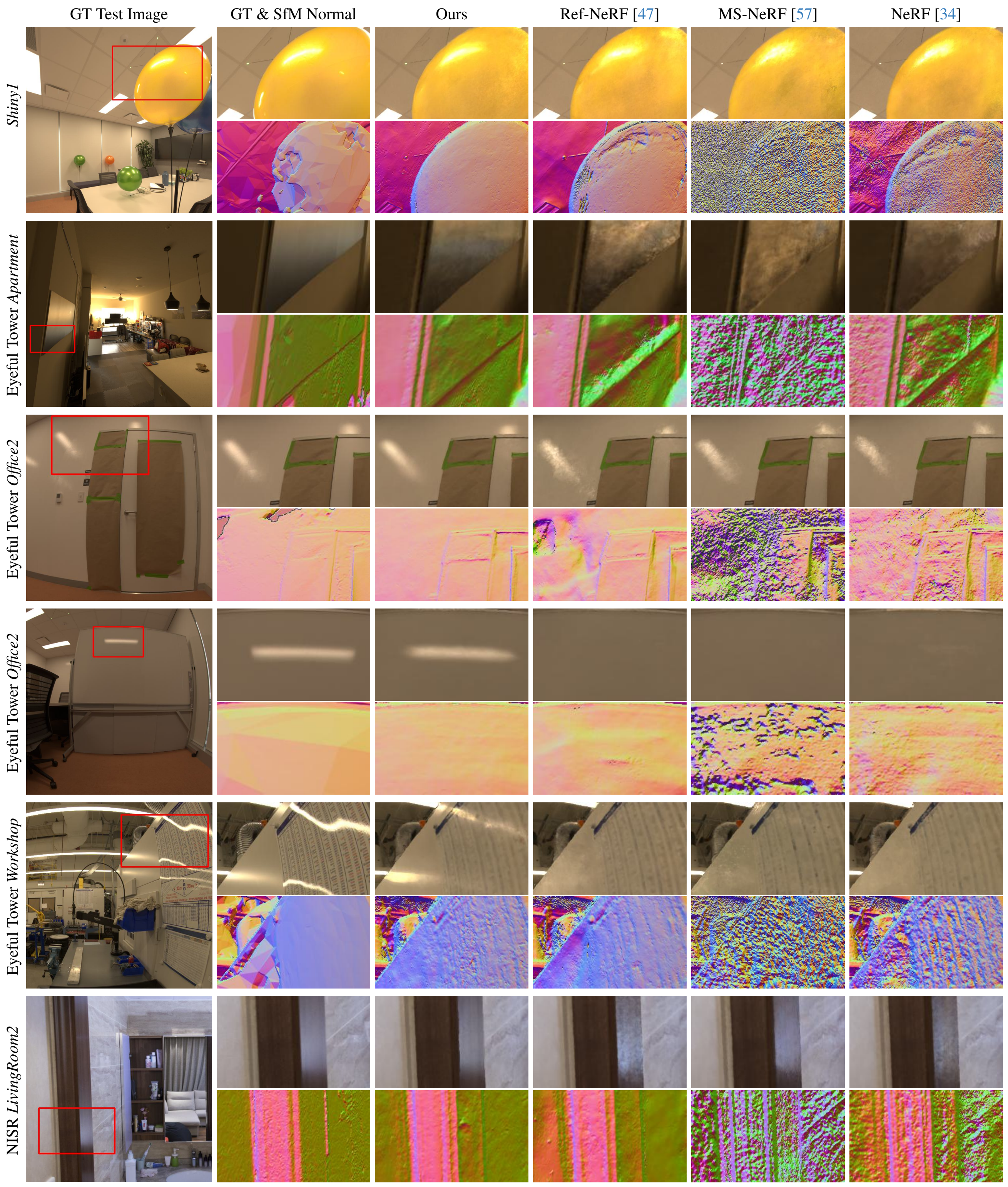

Figure 20. Comparisons of novel-view synthesis quality and normal map visualizations on the Eyeful Tower [53] and NISR datasets [50].

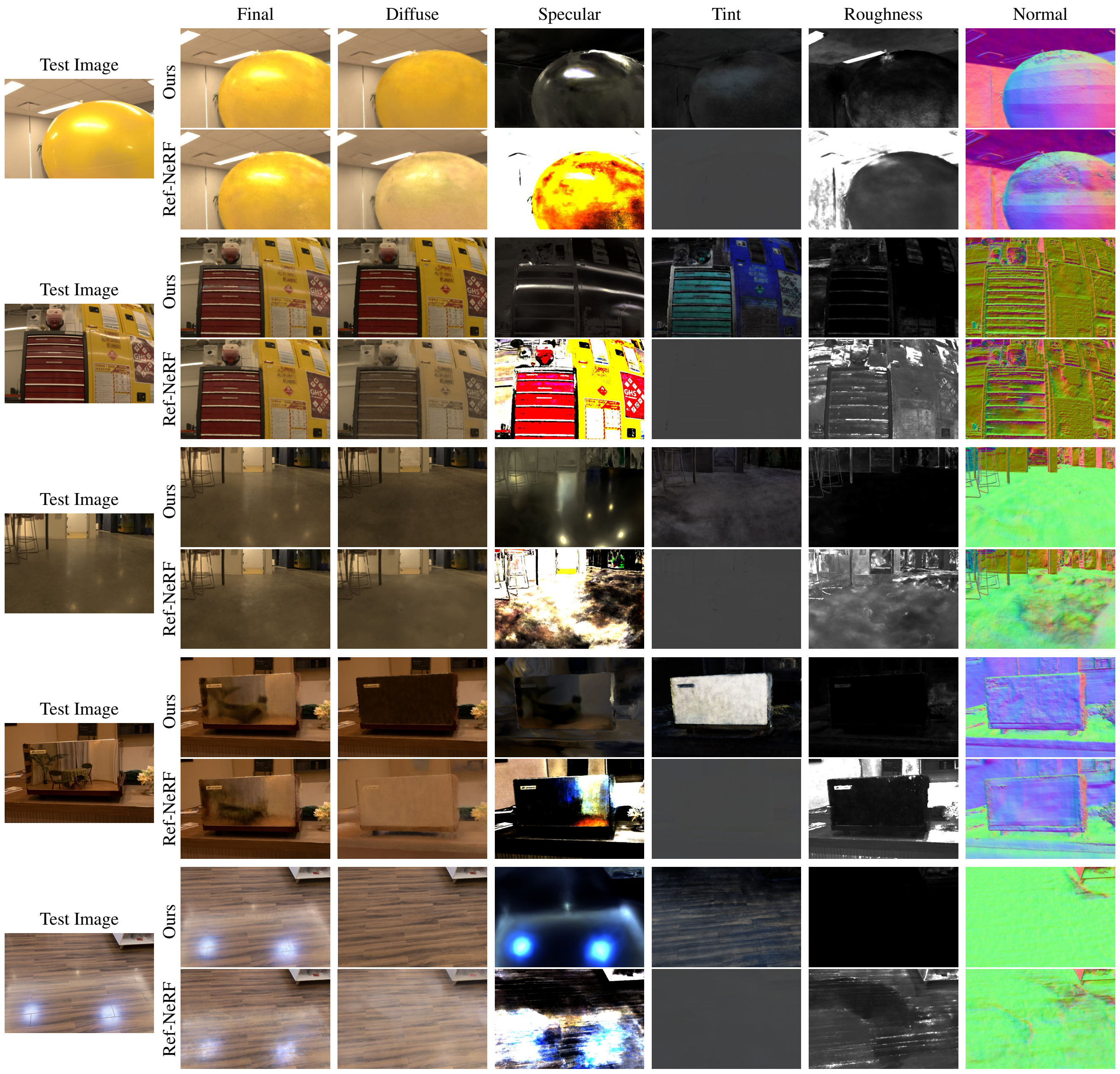

Figure 21. Additional results for intermediate component visualizations of our approach compared to Ref-NeRF [47] on the Eyeful Tower [53] and NISR datasets [50]. Our approach produces more accurate decompositions and normal maps.