NeRF in the Dark: High Dynamic Range View Synthesis from Noisy Raw Images Supplemental Material

Ben Mildenhall Peter Hedman Ricardo Martin-Brualla Pratul P. Srinivasan Jonathan T. Barron
Google Research

Contents

1. Potential negative impact

Training any NeRF model for scene reconstruction has potential negative environmental impact, as current algorithms are very compute-intensive, requiring hours of training per scene even when run on specialized ML accelerators. This also creates an unfair advantage for research groups with access to more computational resources. Future work will likely address this issue, as it blocks the widespread practical adoption of these models.

Any image restoration model could potentially be applied for illicit surveillance purposes. Multi-image denoisers provide the additional capability of potentially revealing details that are not visible in any single image due to noise. ML-based algorithms further complicate this situation by potentially "hallucinating" details in ambiguous regions, either intentionally (as with generative methods) or unintentionally (in the form of reconstruction artifacts). RawNeRF has a minimal ability to hallucinate, as it largely works by simply averaging the input data, but it does occasionally produce high frequency grid-like patterns due to the bias induced by positional encoding.

2. Additional qualitative results

We include additional qualitative results for both dark (Figure 1) and high contrast scenes (Figure 2). We urge the reader to view our supplemental video as the results are more compelling when animated.

3. Training details

3.1. Full derivation of gradient-weighted loss

We wish to approximate the effect of training with the following loss

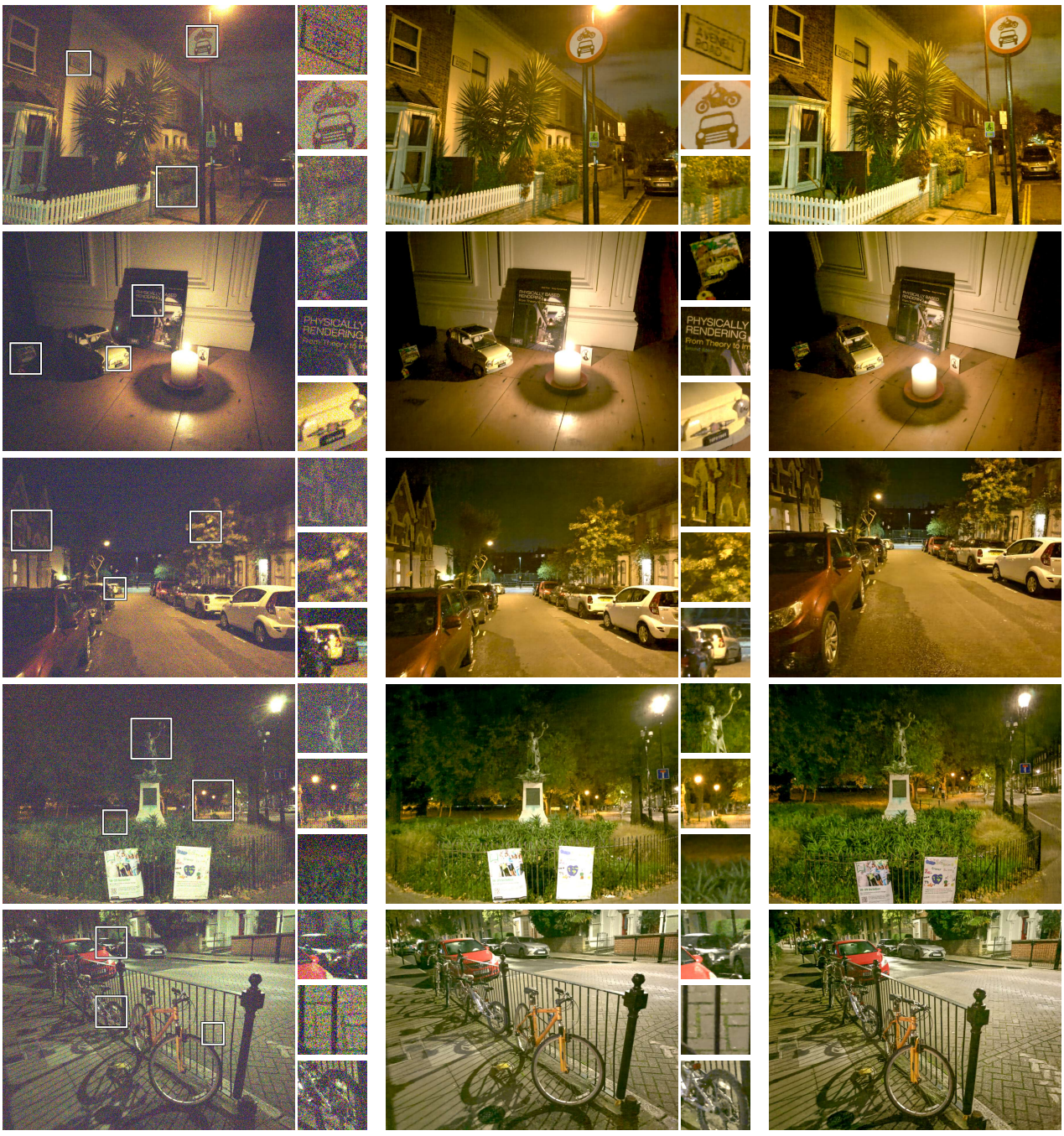
$$L_\psi(\hat{y}, y) = \sum_i (\psi(\hat{y}_i) - \psi(y_i))^2 \tag{1}$$

while converging to an unbiased result. This can be accomplished by using a locally valid linear approximation for the error term:

$$\begin{aligned}\psi(\hat{y}_i) - \psi(y_i) &\approx \psi(\hat{y}_i) - (\psi(\hat{y}_i) + \psi'(\hat{y}_i)(y_i - \hat{y}_i)) \\ &= \psi'(\hat{y}_i)(\hat{y}_i - y_i)\,. \end{aligned} \tag{2}$$

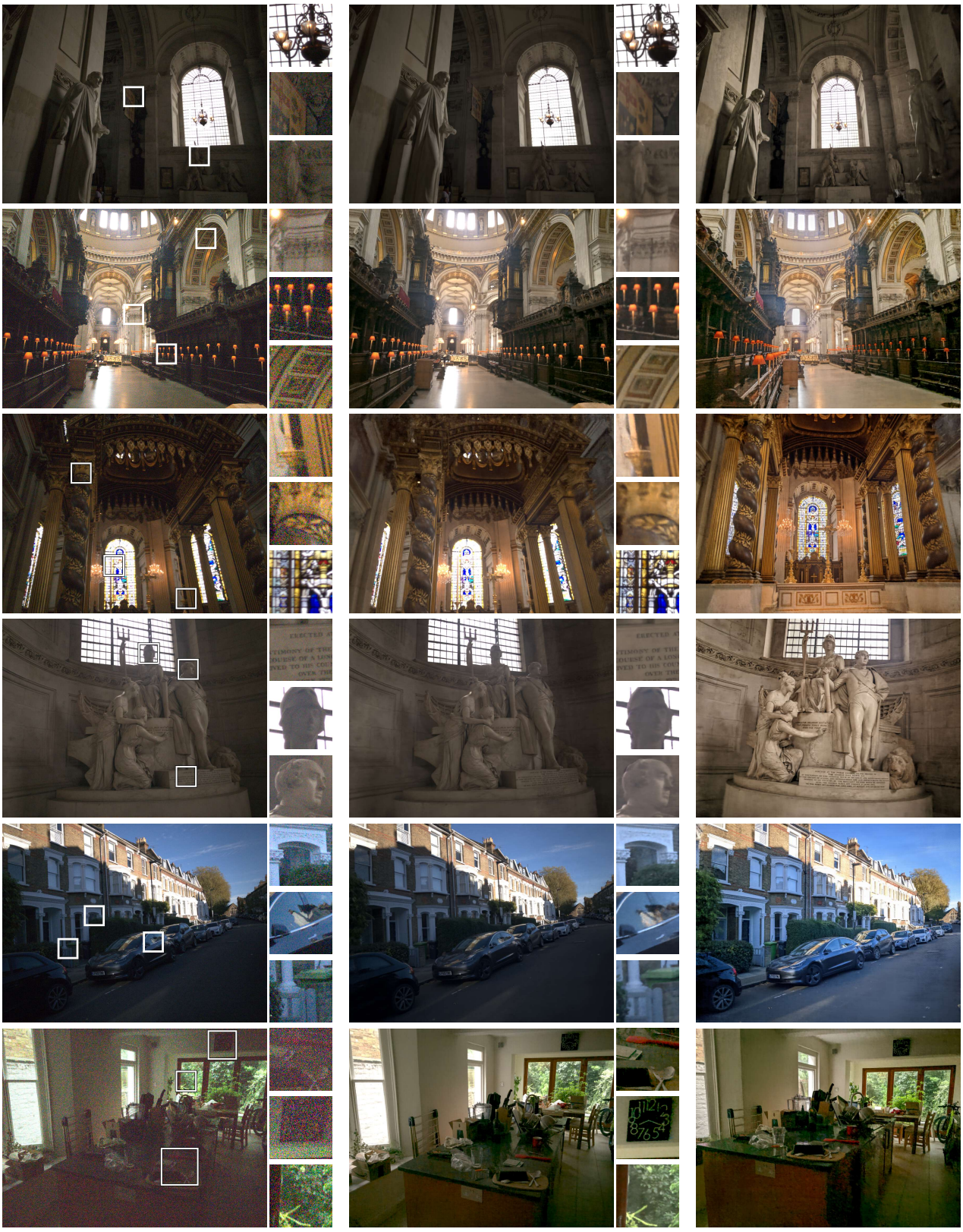
Note that we choose to linearize around $\hat{y}_i$ because, unlike the noisy observation y_i, $\hat{y}_i$ tends towards the true signal value $x_i = \mathrm{E}[y_i]$ over the course of training.

If we use a weighted L2 loss, then as we train the network we will have $\hat{y}_i \to \mathrm{E}[y_i] = x_i$ in expectation (where



Noisy test image | RawNeRF rendering | New viewpoint, HDR tonemapping

Figure 1. RawNeRF in the dark.



Noisy test image | RawNeRF rendering | New viewpoint, HDR tonemapping

Figure 2. Examples of scenes with very high dynamic range.

x_i is the true signal value). This means that the terms summed in our gradient-weighted loss

$$\tilde{L}_\psi(\hat{y}, y) = \sum_i \left[\psi'(\text{sg}(\hat{y}_i))(\hat{y}_i - y_i)\right]^2 \tag{3}$$

will tend towards $\psi'(x_i)(\hat{y}_i - y_i)$ over the course of training. Additionally, we note that the gradient of our reweighted loss 3 is a linear approximation of the gradient of the tonemapped loss 1:

$$\begin{aligned}
\nabla_\theta L_\psi(\hat{y}, y) &= \sum_i \nabla_\theta (\psi(\hat{y}_i) - \psi(y_i))^2 && (4)\\
&= \sum_i 2(\psi(\hat{y}_i) - \psi(y_i))\psi'(\hat{y}_i)\nabla_\theta y_i && (5)\\
&\approx \sum_i 2(\psi'(\hat{y}_i)(\hat{y}_i - y_i))\psi'(\hat{y}_i)\nabla_\theta y_i && (6)\\
&= \sum_i 2(\psi'(\text{sg}(\hat{y}_i))(\hat{y}_i - y_i))\psi'(\text{sg}(\hat{y}_i))\nabla_\theta y_i && (7)\\
&= \nabla_\theta \tilde{L}_\psi(\hat{y}, y)\,. && (8)
\end{aligned}$$

In line 6 we substitute the linearization from 2, and in line 7 we exploit the fact that a stop-gradient has no effect for expressions that will not be further differentiated.

3.2. Weight variance regularizer

Our weight variance regularizer is a function of the compositing weights used to calculate the final color for each ray. Given MLP outputs c_i, σ_i for respective ray segments $[t_{i-1}, t_i)$ with lengths Δ_i (see [2]), these weights are

$$w_i = (1 - \exp(-\Delta_i \sigma_i)) \exp\left(-\sum_{j<i} \Delta_j \sigma_j\right)\,. \tag{9}$$

If we define a piecewise-constant probability distribution p_w over the ray segments using these weights, then our variance regularizer is equal to

$$\mathcal{L}_w = \text{Var}_{X\sim p_w}(X) = \text{E}_{X\sim p_W}\left[(X - \text{E}[X])^2\right] \tag{10}$$

Calculating the mean (expected depth):

$$\begin{aligned}
E_{X\sim p_W}[X] &= \sum_i \int_{t_{i-1}}^{t_i} \frac{w_i}{\Delta_i} t\, dt && (11)\\
&= \sum_i \frac{w_i}{\Delta_i} \frac{t_i^2 - t_{i-1}^2}{2} && (12)\\
&= \sum_i w_i \frac{t_i + t_{i-1}}{2}\,. && (13)
\end{aligned}$$

We will denote this value as $\bar{t}$. Calculating the regularizer:

$$\begin{aligned}
\text{Var}_{X\sim p_w}(X) &= \text{E}_{X\sim p_W}\left[(X - \text{E}[X])^2\right] && (14)\\
&= \sum_i \int_{t_{i-1}}^{t_i} \frac{w_i}{\Delta_i} \left(t - \bar{t}\right)^2 dt && (15)\\
&= \sum_i \frac{w_i}{\Delta_i} \frac{\left(t_i - \bar{t}\right)^3 - \left(t_{i-1} - \bar{t}\right)^3}{3} && (16)\\
&= \sum_i w_i \frac{(t_i - \bar{t})^2 + \left(t_i - \bar{t}\right)\left(t_{i-1} - \bar{t}\right) + \left(t_{i-1} - \bar{t}\right)^2}{3} && (17)
\end{aligned}$$

We apply a weight between 1×10^{-2} and 1×10^{-1} to $\mathcal{L}_w$ (relative to the rendering loss), typically using higher weights in noisier or darker scenes that are more prone to "floater" artifacts. Applying this regularizer with a high weight can result in a minor loss of sharpness, which can be ameliorated by annealing its weight from 0 to 1 over the course of training.

3.3. Findings with alternate loss functions

In practice, we directly scale our loss by the derivative of the desired tone curve:

$$\psi'(\text{sg}(\hat{y}_i)) = \frac{1}{\text{sg}(\hat{y}_i) + \epsilon} \tag{18}$$

We performed a hyperparameter sweep over loss weightings of the form $(\text{sg}(\hat{y}_i) + \epsilon)^{-p}$ for ϵ and p and found that $\epsilon = 1 \times 10^{-3}$ and $p = 1$ produced the best qualitative results.

We also experimented with using a reweighted L1 loss or the negative log-likelihood function of the actual camera noise model (using shot/read noise parameters from the EXIF data) but found that this performed worse than reweighted L2. RawNeRF models supervised with a standard unweighted L2 or L1 loss tended to diverge early in training, particularly in very noisy scenes.

We tried using the unclipped sRGB gamma curve (extended as a linear function below zero and as an exponential function above 1) in our loss, but found that it caused many color artifacts in dark regions. Directly applying our log tone curve (rather than reweighting by its gradient) before the L2 loss caused training to diverge.

3.4. Quality limitations

As briefly mentioned in the main text, our method cannot scale to arbitrary amounts of noise in real world scenes. For our darkest nighttime scenes, we often must run COLMAP [5] multiple times (varying the random seed) or tune its parameters to obtain camera poses. Even when COLMAP reports a successful reconstruction, the results are sometimes poorly aligned at image corners, where the distortion model used for camera intrinsics may not fit well.

RawNeRF itself is prone to reconstruction artifacts in very noisy scenes or scenes captured with few images (under 30), typically in the form of positional encoding grid-like artifacts. These artifacts are often more evident in videos than in still frames. In regions that are essentially pure noise and no signal, RawNeRF sometimes produces a foggy "cloud", since no multiview information exists to guide its recovery of geometry.

The near and far plane bounds calculated using the point cloud from COLMAP are sometimes wider than the true bounds of the scene. Using these bounds wastes many samples at the front of each ray, which reduces sharpness and can cause additional "floater" artifacts. We therefore sometimes retrain RawNeRF models using tighter depth bounds than those reported by COLMAP.

We found it necessary to use gradient clipping due to the high level of noise in the data we use for supervision. Certain losses (such as standard L2) are prone to producing NaN gradient values and require careful tuning of the clipping values. We found our reweighted loss to be more stable.

4. Data capture and postprocessing details

4.1. Data capture

We captured all images using a 2017 iPhone X with the Halide app[1] and a 2020 iPhone SE with the Adobe Lightroom app. We used manual modes in both apps with focus and ISO level fixed for each capture, manually adjusting shutter speed to achieve an exposure with no clipped highlights (except in scenes with varying exposure) and minimal motion blur (at least $1/100$s when possible). At night, it was usually necessary to use the maximum ISO level (approximately 2000 on the iPhones) to achieve minimal motion blur. Each capture took around 10-200 seconds, except for the denoising test scenes. All raw images are stored as Adobe DNG[2] files.

We extract the following parameters from the EXIF metadata using `exiftool`:

Variable	EXIF field name	# values
w	`WhiteLevel`	1
b	`BlackLevel`	1
g_{wb}	`AsShotNeutral`	3
C_{ccm}	`ColorMatrix2`	3×3
t	`ShutterSpeed`	1

The color correction matrix C_{ccm} is an XYZ-to-camera-RGB transform under the D65 illuminant, so we use the corresponding RGB-to-XYZ matrix[3]:

$$C_{\text{rgb-xyz}} = \begin{bmatrix} 0.4124564 & 0.3575761 & 0.1804375 \\ 0.2126729 & 0.7151522 & 0.0721750 \\ 0.0193339 & 0.1191920 & 0.9503041 \end{bmatrix} \quad (19)$$

We use these to create a single color transform C_{all} mapping from camera RGB directly to standard linear RGB space:

$$C_{\text{all}} = \text{rownorm}((C_{\text{rgb-xyz}} C_{\text{ccm}})^{-1}) \quad (20)$$

where rownorm normalizes each to sum to 1.

We use the standard sRGB gamma curve as a basic tonemap for linear RGB space data:

$$\gamma_{\text{sRGB}}(z) = \begin{cases} 12.92z & z \leq 0.0031308 \\ 1.055z^{1/2.4} - 0.055 & z > 0.0031308 \end{cases} \quad (21)$$

4.2. Recovering camera pose

For each scene, we run COLMAP [5] to recover per-image extrinsic poses and a single shared set of intrinsic parameters. We use the post-processed JPEGs produced by the default pipeline in the Halide app (or Lightroom app for the iPhone SE) by using the "RAW+JPEG" capture mode. Given that these camera apps are commercial software, we do not know exactly what postprocessing is applied. However, unlike our own minimal postprocessing pipeline, these apps apply some amount of denoising or smoothing to the images which likely aids COLMAP's feature extraction and matching steps. For particularly noisy scenes, we sometimes find it necessary to run COLMAP multiple times before the bundle adjustment succeeds (this varies the random seed used to initialize the matches for the sparse 3D model). In the case of variable exposure, we directly use the JPEGs as they come out of the app, as is typically done when running structure from motion on datasets captured with auto-exposure, since the extracted image features are robust to changes in brightness.

4.3. Postprocessing pipeline

Our exact postprocessing pipeline for converting raw images to postprocessed sRGB space is detailed below.

1. Load 12-bit raw data using `rawpy`.
2. Cast to 32-bit floating point.
3. Rescale so that the black level is 0 and the white level is 1, preserving values below zero. (The result here is used to train RawNeRF.)

$$z \leftarrow \frac{z - b}{w - b} \quad (22)$$

[1] https://halide.cam/

[2] https://www.adobe.com/content/dam/acom/en/products/photoshop/pdfs/dng_spec_1.4.0.0.pdf

[3] http://www.brucelindbloom.com/index.html?Eqn_RGB_XYZ_Matrix.html

4. Apply bilinear demosaicking (when necessary).

5. Apply elementwise white balance gains.

$$z \leftarrow \frac{z}{g_{\text{wb}}} \tag{23}$$

6. Apply a color correction matrix (from camera RGB to canonical XYZ) and XYZ-to-RGB matrix, combined into a 3×3 transformation.

$$z \leftarrow C_{\text{all}} z \tag{24}$$

7. Adjust the exposure to set the white level to the p-th percentile ($p = 97$ by default).

$$z \leftarrow \frac{z}{\text{percentile}(z, p)} \tag{25}$$

8. Clip to $[0, 1]$.

$$z \leftarrow \text{clip}(z, 0, 1) \tag{26}$$

9. Apply the sRGB gamma curve to each color channel.

$$z \leftarrow \gamma_{\text{sRGB}}(z) \tag{27}$$

When applying a different tonemapping algorithm, we take the color corrected output from step 6 and pass it through the alternate method, while tuning exposure and other tonemapping parameters manually per scene.

4.4. Camera shutter speed miscalibration

In Section 4.2 of the main text, we discuss our implementation of a learned per-color-channel scaling to account for miscalibration when using variable exposure inputs. Here, we document this miscalibration effect for completeness.

Figure 3 plots data taken from a "sweep" over many shutter speeds. The 2017 iPhone X (used for most data capture in the paper) is held fixed on a tripod, all other parameters (focus, ISO, white balance, etc.) are held fixed, and shutter speeds are sampled roughly logarithmically from 1/100 to 1/10000 seconds. We ensure that no pixels are saturated. To minimize the effect of image noise, we study the average color value $y^c_{t_i}$ for each Bayer filter channel (R, G1, G2, B) over the entire 12MP sensor. Specifically, we plot:

$$\frac{y^c_{t_i}}{t_i} \cdot \frac{t_{\max}}{y^c_{t_{\max}}} \tag{28}$$

which is the ratio of normalized brightness at speed t_i to normalized brightness at the longest shutter speed $t_{\max}$. In the case of perfect calibration, this should be equal to 1 everywhere since dividing out by shutter speed should perfectly normalize the brightness value. However, from Figure 3 we see that not only does this quantity decay for faster shutter speeds, it decays at *different rates* per color channel. To preempt concerns that this problem is due to black level miscalibration, we include the plot based on the correct black level 528, as well as the surrounding values, which shows that this problem is only worsened by shifting the black level higher or lower. Note that black level is an integer on the scale of 0 to 4095 (since this is a 12-bit sensor).

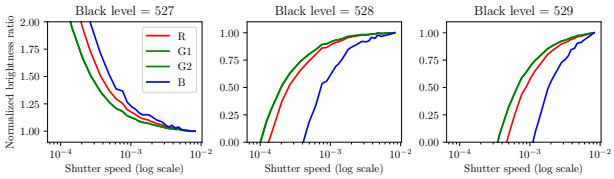


Figure 3. Camera shutter speed miscalibration. We plot normalized brightness for each Bayer color channel, relative to its value at the longest shutter speed. For a perfectly calibrated sensor, these lines would all be at a constant height of 1. We show plots using both the true black level (528) and surrounding values.

(a) Fast shutter (b) Fast shutter, corrected (c) Slow shutter

Figure 4. (a) Fast and (c) slow captures of the *testyucca* scene, with brightness normalized by shutter speed (heavily downscaled to minimize noise). These two images should match perfectly, but have a perceptible color difference due to the miscalibration documented in Section 4.4 and Figure 3. (b) In the center, we show a version of (a) with per-channel rescaling in the raw domain to match the global color balance of (c).

We show an example of the resulting qualitative color shift in Figure 4 using images from one of our three real test scenes. Here the two shutter speeds are 1/1104 and 1/181 seconds, and the relative color shift from the slow to the fast channel is calculated to be $(0.89, 0.93, 0.75)$ for red, green, and blue in the raw domain. The effect of undoing this shift before postprocessing is shown in Figure 4b. This miscalibration is another reason for primarily reporting affine-aligned metrics on our real test set, since we cannot rely on perfect color alignment between the input noisy image and the clean ground truth frame.

We do not fully understand the cause of this issue. We speculate that it could be due to the sensor temperature changing over the course of capture, imprecise shutter speed timing for very fast exposures, or any number of other factors related to low level sensor hardware. Given that the effect exists and affects our captures in an unmeasurable manner, it must be accounted for. Using a DSLR or mirror-

	Unproc.	SID	RViDeNet	UDVD
Time (sec)	1.8	4.0	65.1	229.2

Table 1. Time required for compared methods to denoise a single frame. RawNeRF is run on different hardware and requires about 25 seconds of render time per frame (see details in Section 5.1).

less camera with a better sensor may avoid this issue.

5. Comparison and ablation details

5.1. Runtimes

Each scene requires about 12 hours of training for RawNeRF (same as mip-NeRF) and rendering a 12MP output image takes about 25 seconds, running on 16 TPU v2 accelerators. In Table 1, we list timings for the compared methods to denoise one 12MP frame, running on a single NVIDIA GTX 1080 GPU. We use the publicly available model weights for each compared method; these presumably require hours or days to train, but this step must only be performed once (in comparison to NeRF, which is optimized from scratch for every scene).

5.2. Real test dataset details

Affine alignment As mentioned in the main text, we solve for an affine color alignment between each output and the ground truth clean image. For all methods but SID and LDR NeRF, this is done directly in raw Bayer space for each RGGB plane separately. For SID and LDR NeRF (which output images in tonemapped sRGB space), this is done for each RGB plane against the tonemapped sRGB clean image. If the ground truth channel is x and the channel to be matched is y, we specifically compute

$$a = \frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2} = \frac{\mathrm{Cov}(x, y)}{\mathrm{Var}(x)}, \tag{29}$$

$$b = \overline{y} - a\overline{x} \tag{30}$$

to get the least-squares fit of an affine transform $ax + b \approx y$ (here $\overline{z}$ indicates the mean over all elements of z). We then apply the inverse transform as $(y - b)/a$ to match the estimated y to x. In the case where matching happens in the raw domain, we postprocess $(y - b)/a$ through our standard pipeline (Section 4.3) before calculating sRGB-space metrics.

Compared baselines We provide an overview of each baseline and the pre- and post-processing pipelines used in the main text. Unprocessing [3] is the only method that is a "non-blind" denoiser, and therefore requires a per-pixel noise level as input. We calculate this by using the empirical per-pixel variance from our tripod-aligned fast and clean images to estimate shot and read noise parameters as a best-fit 1D affine transform mapping from clean signal values to empirical variances. Each method required its own relative input rescaling and clipping convention, which we set based on each authors' source code.

	Simulated shutter speed (seconds)						
Method	∞	1/7	1/15	1/30	1/60	1/120	1/240
Noisy input	-	20.16	16.90	13.81	10.83	8.06	5.95
LDR NeRF	**38.06**	24.66	21.39	18.27	15.31	12.47	10.13
RawNeRF	36.85	**36.82**	**36.65**	**36.27**	**35.62**	**34.33**	**32.37**

Table 2. Unmasked LDR sRGB PSNRs for the ablation study on our synthetic *Lego* scene data.

5.3. Synthetic Lego dataset details

In the synthetic Lego dataset, we did *not* include the effects of remosaicking/demosaicking or quantization when unprocessing/reprocessing the data. We wanted the "infinite" shutter speed case to be perfectly clean, with no degradation resulting from unprocessing and reprocessing in the absence of noise, thus providing an upper bound on possible performance. This example does not particularly test the ability of RawNeRF to encode high dynamic range since the object is diffusely lit, resulting in fairly dim highlights and negligible clipping; instead, it focuses on robustness to noise.

We rendered new randomly sampled images of the scene using the Blender file[4] provided by the NeRF authors [4], saving the resulting linear space color data in EXR format. There are 120 images in the training set and 40 images in the test set. Note that metric values on this data are not comparable to metrics on the original scene, since it uses images from different random poses generated using a different postprocessing pipeline.

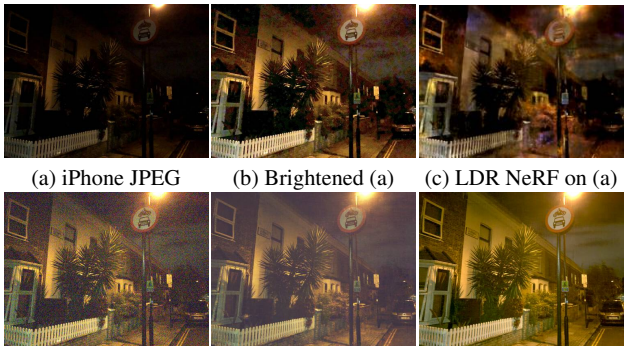
For completeness, we report the unmasked PSNR values for this experiment in Table 2 (Table 2 in the main text reports masked PSNR), which is heavily skewed by the LDR NeRF's color bias in the black background regions.

6. Further qualitative ablations

6.1. Training with iPhone JPEG inputs

In all LDR NeRF comparisons in the main paper, we use our own simple postprocessing pipeline to generate LDR sRGB inputs from the raw data. However, a standard NeRF implementation would instead use JPEG images directly from the camera, which have a more sophisticated postprocessing pipeline that likely includes noise reduction and a more sophisticated nonlinear tonemap to better compress dynamic range. To satisfy the reader's potential curiosity, in

[4] https://drive.google.com/file/d/1RjwxZCUoPlUgEWIUiuCmMmG0AhuV8A2Q/view?usp=sharing



(a) iPhone JPEG (b) Brightened (a) (c) LDR NeRF on (a)

(d) Our LDR input (e) LDR NeRF on (d) (f) RawNeRF

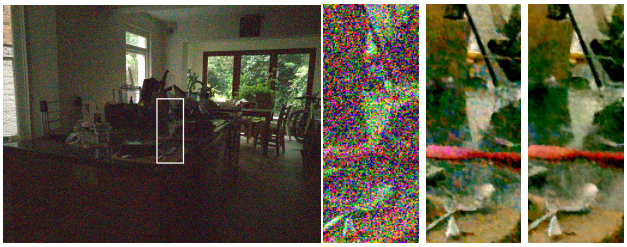
Figure 5. Comparison of training LDR NeRF using sRGB images either directly from the iPhone camera or from our simplified pipeline. (a) The JPEG image from the phone is extremely dark, so we brighten it for visualization (b). We also brighten the resulting LDR NeRF rendering (c), thereby revealing its pervasive color noise artifacts. When trained on the images from our LDR processing pipeline (d), LDR NeRF produces a more reasonable result (e), though the input images' biased noise distribution still results in muddy, low contrast dark regions and incorrectly muted colors. (f) Only RawNeRF accurately recovers the correct colors and details throughout the scene.

Figure 5 we provide an example of LDR NeRF trained on iPhone JPEGs versus our LDR images, as well as a RawNeRF result on the same scene.

6.2. Bayer mosaic mask and sensor artifacts

In the main text, we note that we only apply our loss function to the color channel measured by the Bayer filter for each ray. (In practice, we render all three colors for every training ray, then apply a one-hot mask to select the desired output color.) In Figure 6, we show an example of the color noise that emerges when supervising all 3 color channels using bilinearly demosaicked raw images instead of masking the loss. Perhaps surprisingly, we noted that relatively clean regions of the scene seemed to benefit from using all 3 channels of a bilinear demosaicked image as supervision. However, we concluded that the distracting color artifacts induced by demosaicking outweighed this occasional benefit, and opted to use Bayer masking in all scenes.

These artifacts may potentially be caused by broken "hot" pixels that are always fully saturated, in violation of our assumed noise distribution. Bilinear demosaicking would disperse the influence of a hot pixel to many neighboring pixels, potentially increasing its effect on the final trained NeRF. In preliminary experiments, we did not notice any benefit to additionally masking hot pixels when applying a Bayer mask. We did apply a second mask to remove a 4 pixel border from all training images, since many iPhone raw images contained 1 or 2 entire rows or columns of saturated pixels on one side, particularly in bright scenes.



(a) Full noisy image (b) Noisy (c) No mask (d) Ours

Figure 6. Comparison of training with bilinear demosaicking and no Bayer masking (c), or with a Bayer mask that uses only the measured raw pixels (d). In image areas with extremely high noise, we observed unpleasant bright color noise emerge when training with bilinear demosaicked images in the raw domain.

7. Synthetic defocus rendering model

To render defocused images, we use a similar rendering model as prior work that has addressed this task [1, 6, 7]. To avoid prohibitively expensive rendering speeds, we first precompute a multiplane image [8] representation from the trained RawNeRF model. This MPI consists of a series of fronto-parallel RGBA planes (with colors still in linear HDR space), sampled linearly in disparity within a camera frustum at a central camera pose. Given this MPI representation, our rendering algorithm for synthetic defocus (including lateral camera translation) is described in Algorithm 1.

Algorithm 1 Synthetic defocus rendering

procedure DEFOCUS($c_{\text{mpi}}, \alpha_{\text{mpi}}, i_{\text{focus}}, \Delta_r, \Delta_d$)
  $C \leftarrow 0$
  for $i = 0, \ldots, , N-1$ **do**
    $r \leftarrow \Delta_r \cdot |i - i_{\text{focus}}|$
    $k_{\text{blur}} \leftarrow \text{blurkernel}(r)$
    $c_{\text{blur}} \leftarrow \text{convolve}(c_{\text{mpi}}^{(i)} \cdot \alpha_{\text{mpi}}^{(i)}, k_{\text{blur}})$
    $\alpha_{\text{blur}} \leftarrow \text{convolve}(\alpha_{\text{mpi}}^{(i)}, k_{\text{blur}})$
    $d \leftarrow \Delta_d \cdot i$
    $c_{\text{trans}} \leftarrow \text{translate}(c_{\text{blur}}, d)$
    $\alpha_{\text{trans}} \leftarrow \text{translate}(\alpha_{\text{blur}}, d)$
    $C \leftarrow c_{\text{trans}} + (1 - \alpha_{\text{trans}})C$
  end for
  return C
end procedure

Here the input MPI planes are indexed from back to front. i_{focus} controls the focal plane, Δ_r controls the simulated aperture size (defocus strength), and Δ_d (a 2D vector) controls the camera translation parallel to the image plane. $\text{blurkernel}(r)$ returns a circular mask at the origin with radius r pixels. blurkernel is implemented as a 2D Fourier space convolution, and translate is a continuous 2D image

translation (using bilinear resampling). Note that the color is "premultiplied" by alpha before blurring, which is why alpha is not applied to c_{trans} in the accumulation step for C.

8. Scene index

We provide various details about each scene shown in the paper and video in Table 3.

References

[1] Jonathan T. Barron, Andrew Adams, YiChang Shih, and Carlos Hernández. Fast bilateral-space stereo for synthetic defocus. *CVPR*, 2015. 8

[2] 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. 4

[3] Tim Brooks, Ben Mildenhall, Tianfan Xue, Jiawen Chen, Dillon Sharlet, and Jonathan T. Barron. Unprocessing images for learned raw denoising. *CVPR*, 2019. 7

[4] 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. 7

[5] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. *CVPR*, 2016. 4, 5

[6] Neal Wadhwa, Rahul Garg, David E. Jacobs, Bryan E. Feldman, Nori Kanazawa, Robert Carroll, Yair Movshovitz-Attias, Jonathan T. Barron, Yael Pritch, and Marc Levoy. Synthetic depth-of-field with a single-camera mobile phone. *SIGGRAPH*, 2018. 8

[7] Xuaner Zhang, Kevin Matzen, Vivien Nguyen, Dillon Yao, You Zhang, and Ren Ng. Synthetic defocus and look-ahead autofocus for casual videography. *SIGGRAPH*, 2019. 8

[8] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *SIGGRAPH*, 2018. 8

	Scene	Figures	Video	Images	Shutter speed (s^{-1})	ISO	Time of day
Main text	*candle*	1	0:00, 1:45	173	45, 119	2000	20:21
	livingroom	2		50	1429	800	15:14
	stove	4	3:33	106	139, 258, 1621	2000	20:17
	windowlegovary	5, 8d	4:28	104	432, 16129, 16393	500	10:29
	gardenlights	8a-c	5:39	91	50	1600	23:53
Test set	*pianotest*	6, 8e	5:25	103	145, 207	2000	22:08
	officetest	6		113	110, 249	2000	17:43
	yuccatest	6, A4		102	181, 1104	800	13:09
Supplement and video	*streetcorner*	A1a, A5	2:35	57	123	2000	22:19
	candlefiat	A1b	4:15	52	97	2000	00:33
	nightstreet	A1c		49	82	2000	23:04
	parkstatue	A1d	5:13	51	124	2000	23:14
	bikes	A1e	3:21	45	62	2000	22:22
	twostatue	A2a	4:52	86	239	20	11:32
	choir	A2b	4:03	28	112	50	11:50
	stainedglass	A2c	5:02	43	155	32	11:46
	onestatue	A2d	4:42	40	112	25	11:51
	sharpshadow	A2e		36	8130	32	13:35
	morningkitchen	A2f, A6		53	110	2000	08:18
	scooter		3:08	54	107	2000	19:27
	notchbush		3:44	63	95	2000	22:15

Table 3. A summary of image metadata for our scenes. Figure from the supplement are indicated using the prefix "A".