

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

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

Abstract

Neural Radiance Fields (NeRF) is a technique for high quality novel view synthesis from a collection of posed input images. Like most view synthesis methods, NeRF uses tonemapped low dynamic range (LDR) as input; these images have been processed by a lossy camera pipeline that smooths detail, clips highlights, and distorts the simple noise distribution of raw sensor data. We modify NeRF to instead train directly on linear raw images, preserving the scene’s full dynamic range. By rendering raw output images from the resulting NeRF, we can perform novel high dynamic range (HDR) view synthesis tasks. In addition to changing the camera viewpoint, we can manipulate focus, exposure, and tonemapping after the fact. Although a single raw image appears significantly more noisy than a postprocessed one, we show that NeRF is highly robust to the zero-mean distribution of raw noise. When optimized over many noisy raw inputs (25-200), NeRF produces a scene representation so accurate that its rendered novel views outperform dedicated single and multi-image deep raw denoisers run on the same wide baseline input images. As a result, our method, which we call RawNeRF, can reconstruct scenes from extremely noisy images captured in near-darkness.

1. Introduction

View synthesis methods, such as neural radiance fields (NeRF) [37], typically use tonemapped low dynamic range (LDR) images as input and directly reconstruct and render new views of a scene in LDR space. This poses no issues for scenes that are well-lit and do not contain large brightness variations, since they can be captured with minimal noise using a single fixed camera exposure setting. However, this precludes many common capture scenarios: images taken at nighttime or in any but the brightest indoor spaces will have poor signal-to-noise ratios, and scenes with regions of both daylight and shadow have extreme contrast ratios that require high dynamic range (HDR) to represent accurately.

Our method, RawNeRF, modifies NeRF to reconstruct the scene in linear HDR color space by supervising directly on noisy raw input images. This bypasses the lossy postpro-

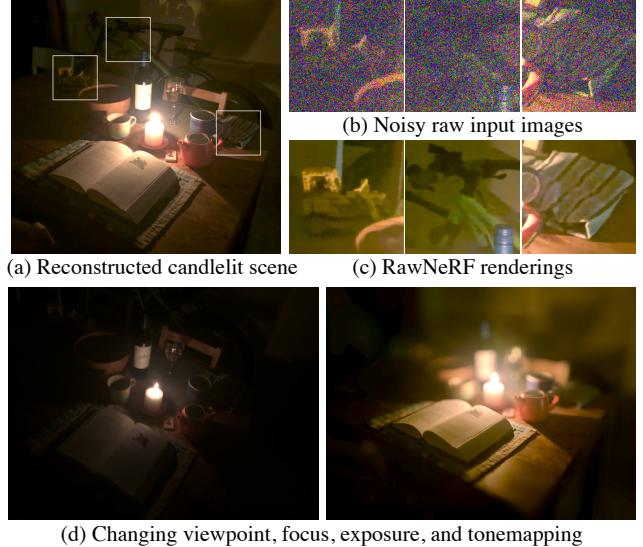


Figure 1. By jointly optimizing a single scene representation over many input images, NeRF is surprisingly robust to high levels of image noise. We exploit this fact to train RawNeRF directly on completely unprocessed HDR linear raw images. In this nighttime scene lit only by a single candle (a), RawNeRF can extract details from the noisy raw data that would have been destroyed by post-processing (b, c). RawNeRF recovers full HDR color information, enabling HDR view synthesis tasks such as changing focus and exposure for rendered novel views. The resulting renderings can be retouched like any raw photograph: here we show (d, left) a dark all-in-focus exposure with a simple global tonemap and (d, right) a brighter, synthetically refocused exposure postprocessed by HDRNet [17]. See our supplementary video for more results.

cessing that cameras apply to compress dynamic range and smooth out noise in order to produce visually palatable 8-bit JPEGs. By preserving the full dynamic range of the raw inputs, RawNeRF enables various novel HDR view synthesis tasks. We can modify the exposure level and tonemapping algorithm applied to rendered outputs and even create synthetically refocused images with accurately rendered bokeh effects around out-of-focus light sources.

Beyond these view synthesis applications, we show that training directly on raw data effectively turns Raw-

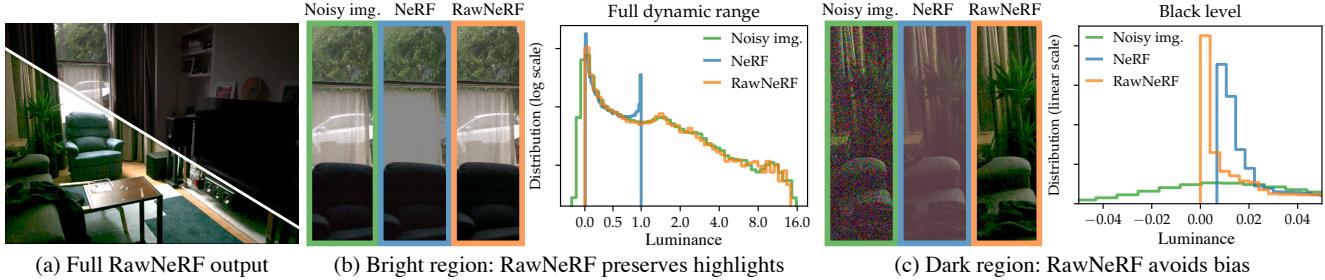


Figure 2. Failure modes of NeRF on a daytime indoor scene. (a) Here we show two exposures ($24\times$ apart) of a full RawNeRF output rendering, both passed through a global tonemapping curve. Training NeRF with postprocessed LDR images, as done in prior work, (b) prevents it from recovering bright highlights clipped above at 1, resulting in the missing car outside the window, and (c) corrupts the per-pixel noise distribution such that NeRF recovers incorrect colors due to the nonlinear tonemap and clipping below at 0, particularly in dark regions around the plant and sofa. In contrast, RawNeRF trains directly on HDR linear raw images and correctly recovers the radiance distribution in both extremely bright and extremely dark parts of the scene.

NeRF into a multi-image denoiser capable of reconstructing scenes captured in near-darkness (Figure 1). The standard camera postprocessing pipeline (e.g., HDR+ [20]) corrupts the simple noise distribution of raw data, introducing significant bias in order to reduce variance and produce an acceptable output image. Feeding these images into NeRF thus produces a biased reconstruction with incorrect colors, particularly in the darkest regions of the scene (see Figure 2 for an example). We instead exploit NeRF’s ability to reduce variance by aggregating information across frames, demonstrating that it is possible for RawNeRF to produce a clean reconstruction from many noisy raw inputs.

Unlike typical video or burst image denoising methods, RawNeRF assumes a static scene and expects camera poses as input. Provided with these extra constraints, RawNeRF is able to make use of 3D multiview consistency to average information across nearly *all* of the input frames at once. Since our captured scenes each contain 25-200 input images, this means RawNeRF can remove more noise than feed-forward single or multi-image denoising networks that only make use of 1-5 input images for each output.

In summary, we make the following contributions:

1. We propose a method for training RawNeRF directly on raw images that can handle high dynamic range scenes as well as noisy inputs captured in the dark.
2. We show that RawNeRF outperforms NeRF on noisy real and synthetic datasets and is a competitive multi-image denoiser for wide-baseline static scenes.
3. We showcase novel view synthesis applications made possible by our linear HDR scene representation (varying exposure, tonemapping, and focus).

2. Related Work

RawNeRF combines concepts from several areas of research. We build upon NeRF as a baseline for high quality view synthesis, bring in ideas from low level image pro-

cessing to optimize NeRF directly on noisy raw data, and take inspiration from uses of HDR in computer graphics and computational photography to showcase new applications made possible by an HDR scene reconstruction. We briefly cover relevant prior work across each of these areas.

2.1. Novel view synthesis

Novel view synthesis is the task of using a set of input images and their camera poses to reconstruct a scene representation capable of rendering novel views. When the input images are densely sampled, it is possible to use direct interpolation in pixel space for view synthesis [19, 31]. A more feasible capture scenario is to capture more widely spaced inputs and use a “proxy” geometry (e.g., a reconstructed triangle mesh) to reproject and combine colors from the input images, using either a heuristic [6] or learned [21, 39, 40] blending function.

Recent work on applying deep learning to view synthesis has focused on volumetric rather than mesh-based scene representations [15, 34, 53]. NeRF [37] directly optimizes a *neural* volumetric scene representation to match all input images using gradient descent on a rendering loss. Various extensions have improved NeRF’s robustness to varying lighting conditions [35] or added supervision with depth [24, 47, 48], time-of-flight data [1], or semantic segmentation labels [52]. As of yet, no approach has extended NeRF to work with high dynamic range color data. Some previous view synthesis methods trained using LDR data jointly solve for per-image scaling factors to account for inconsistent lighting or miscalibration between cameras [27, 34]. ADOP [41] supervises with LDR images and solves for exposure through a differentiable tonemapping step to approximately recover HDR, but does not focus on robustness to noise or supervision with raw data.

2.2. Denoising

Early neural denoising approaches mostly focused on denoising sRGB images synthetically corrupted with additive white Gaussian noise [50]. In 2017, Plötz and Roth [38] established a real raw image denoising benchmark, which showed that these deep denoisers failed to generalize beyond the synthetic data used during training and were outperformed by standard non-learned methods, such as BM3D [11]. Subsequent work on both single [5, 9] and multi-image [8, 18, 36, 49] denoising demonstrated the benefits of training networks to operate directly on noisy raw input data. Modern cellphone camera pipelines perform a robust averaging of multiple noisy input frames in the raw domain [20], though they typically cannot afford to employ deep networks due to speed and power limitations.

Another line of research investigated whether denoisers could be trained using *only* noisy data when no corresponding clean ground truth exists. Noise2Noise [30] demonstrated this was possible given a dataset of pairs of independent noisy observations of the same image, an insight Ehret *et al.* [13] applied to denoise videos by aligning consecutive noisy frames. Various followups to Noise2Noise proposed modified network architectures allowing supervision with a dataset of single noisy images [4, 28, 29]. Sheth *et al.* [43] showed that this paradigm could be applied to train a denoiser using a *single* noisy video, including an application to raw video data. Similarly, RawNeRF is optimized over a single set of images to both denoise and recover the 3D structure of the captured scene.

2.3. Applications of raw and HDR image data

Computational photography The value of working directly with raw data has long been noted by digital photographers due to the fact that its preservation of dynamic range allows for maximum postprocessing flexibility, letting users modify exposure, white balance, and tonemapping after the fact. Many works have tried to automate this process by using heuristics or machine learning to map directly from raw data to postprocessed LDR images [7, 9, 17, 22].

Another line of work focuses on recovering HDR images from LDR inputs. This concept was pioneered byDebevec and Malik [12], who used a stack of aligned LDR images taken at different exposures to recover and invert the camera’s nonlinear response curve. Current approaches apply machine learning to produce HDR outputs from single [14] or multiple misaligned [23] LDR inputs, either recovering or hallucinating detail in clipped highlights.

Synthetic defocus Many modern cellphones include a postprocessing option to add synthetic defocus blur after capture [45]. Though it is possible to accurately simulate defocus using a thin-lens model [10] or real multi-element

camera lens [26] using ray tracing, most machine learning models use a much faster approximate rendering model, predicting a depth map and applying a depth-varying blur kernel to each discretized depth layer [2, 44]. Performing this blur in HDR space is critical to achieving the correct appearance of defocused bright highlights (known as “bokeh”), as demonstrated by Zhang *et al.* [51].

3. Noisy Raw Input Data

NeRF [37] takes postprocessed low dynamic range (LDR) sRGB color space images as input. This works well when using clean, noise-free images with minimal contrast. However, all real images contain some level of noise, and each step in the camera postprocessing pipeline corrupts this distribution in a certain way. Here we briefly describe the simplified pipeline stages relevant to our method.

Raw camera measurements When capturing an image, the number of photons hitting a pixel on the camera sensor is converted to an electrical charge, which is recorded as a high bit-depth digital signal (typically 10 to 14 bits). These values are offset by a “black level” to allow for negative measurements due to noise. After black level subtraction, the signal is a noisy measurement y_i of a quantity x_i proportional to the expected number of photons arriving while the shutter is open. This noise results from both the physical fact that photon arrivals are a Poisson process (“shot” noise) and noise in the readout circuitry that converts the analog electrical signal to a digital value (“read” noise). The combined shot and read noise distribution can be well modeled as a Gaussian whose variance is an affine function of its mean [16]; importantly, this implies that the distribution of the error $y_i - x_i$ is zero mean.

Color filter demosaicking Color cameras contain a Bayer color filter array in front of the image sensor such that each pixel’s spectral response curve measures either red, green or blue light. The pixel color values are typically arranged in 2×2 squares containing two green pixels, one red, and one blue pixel (known as a Bayer pattern), resulting in “mosaicked” data. To generate a full-resolution color image, the missing color channels are interpolated using a demosaicking algorithm [32]. This interpolation correlates noise spatially, and the checkerboard pattern of the mosaic leads to different noise levels in alternating pixels.

Color correction and white balance The spectral response curves for each color filter element vary between different cameras, and a color correction matrix is used to convert the image from this camera-specific color space to a standardized color space. Additionally, because human perception is robust to the color tint imparted by different

light sources, cameras attempt to account for this tint (*i.e.*, make white surfaces appear RGB-neutral white) by scaling each color channel by an estimated white balance coefficient. These two steps are typically combined into a single linear 3×3 matrix transform, which further correlates the noise between color channels.

Gamma compression and tonemapping Humans are able to discern smaller relative differences in dark regions compared to bright regions of an image. This fact is exploited by sRGB gamma compression, which optimizes the final image encoding by clipping values outside $[0, 1]$ and applying a nonlinear curve to the signal that dedicates more bits to dark regions at the cost of compressing bright highlights. In addition to gamma compression, tonemapping algorithms can be used to better preserve contrast in high dynamic range scenes (where the bright regions are several orders of magnitude brighter than the darkest) when the image is quantized to 8 bits [12, 20].

In a slight abuse of terminology, we will refer both of these steps jointly as “tonemapping” in the rest of the paper, indicating the process by which linear HDR values are mapped to nonlinear LDR space for visualization. We will refer to signals before tonemapping as high dynamic range (HDR) and signals after as low dynamic range (LDR). Of all postprocessing operations, tonemapping has the most drastic effect on the noise distribution: clipping completely discards information in the brightest and darkest regions, and after the non-linear tonemapping curve the noise is no longer guaranteed to be Gaussian or even zero mean.

4. RawNeRF

A neural radiance field (NeRF) [37] is a neural network based scene representation that is optimized to reproduce the appearance of a set of input images with known camera poses. The resulting reconstruction can then be used to render novel views from previously unobserved poses. NeRF’s multilayer perceptron (MLP) network takes 3D position and 2D viewing direction as input and outputs volume density and color. To render each pixel in an output image, NeRF uses volume rendering to combine the colors and densities from many points sampled along the corresponding 3D ray.

Standard NeRF takes clean, low dynamic range (LDR) sRGB color space images with values in the range $[0, 1]$ as input. Converting raw HDR images to LDR images (*e.g.*, using the pipeline described in Section 3) has two significant consequences:

1. Detail in bright areas is lost when values are clipped from above at one, and detail across the image is compressed by the tonemapping curve and subsequent quantization to 8 bits.

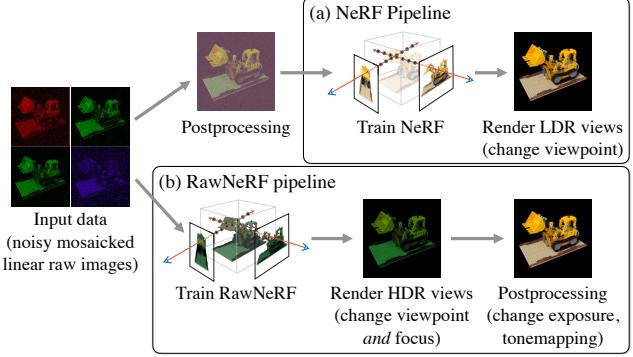


Figure 3. The standard NeRF training pipeline (a) takes in LDR images that have been sent through a camera processing pipeline, reconstructing the scene and rendering new views in LDR color space. As such, its renderings are effectively already postprocessed and cannot be significantly retouched. In contrast, our method RawNeRF (b) modifies NeRF to train directly on linear raw HDR input data. The resulting scene representation produces novel views that can be edited like any raw photograph.

2. The per-pixel noise distribution becomes biased (no longer zero-mean) after passing through a nonlinear tonemapping curve and being clipped from below at zero.

The goal of RawNeRF is to make use of this information rather than discarding it, optimizing NeRF directly on linear raw input data in HDR color space (Figure 3). In Section 5, we will show that reconstructing NeRF in raw space makes it much more robust to noisy inputs and allows for novel HDR view synthesis applications. First, we detail the changes required to make NeRF work with raw data.

4.1. Loss function

Since the color distribution in an HDR image can span many orders of magnitude, a standard L2 loss applied in HDR space will be completely dominated by error in bright areas and produce an image that has muddy dark regions with low contrast when tonemapped (see Figure 4). Instead, we apply a loss that more strongly penalizes errors in dark regions to align with how human perception compresses dynamic range. One way to achieve this is by passing both the rendered estimate \hat{y} and noisy observed intensity y through a tonemapping curve ψ before the loss is applied:

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

However, in low-light raw images the observed signal y is heavily corrupted by zero-mean noise, and a nonlinear tonemap will introduce bias that changes the noisy signal’s expected value ($E[\psi(y)] \neq \psi(E[y])$). In order for the network to converge to an unbiased result [30], we instead use

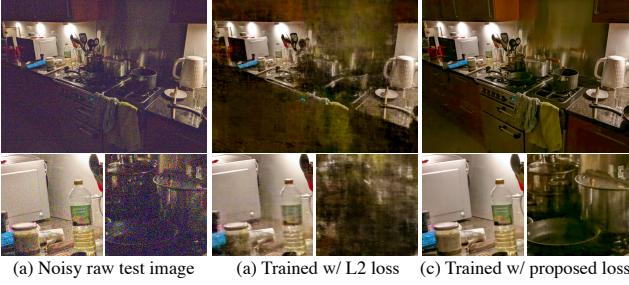


Figure 4. This challenging scene (a) has a $7000\times$ ratio between its 90th and 10th raw color percentiles. (b) When faced with such high-contrast inputs, the standard L2 loss from NeRF manages to recover the bright parts of the scene but produces poor results in darker regions, which becomes particularly apparent after LDR tonemapping. (c) Our proposed loss (4), reweighted according to the gradient of a log tonemap curve, successfully reconstructs all parts of the scene. Both rendered images are tonemapped using HDR+ [20] for visualization.

a weighted L2 loss of the form

$$L(\hat{y}, y) = \sum_i w_i (\hat{y}_i - y_i)^2. \quad (2)$$

We can approximate the tonemapped loss (1) in this form by using a linearization of the tone curve ψ around each \hat{y}_i :

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

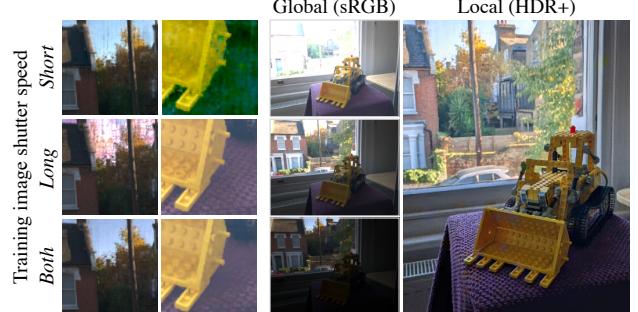
where $\text{sg}(\cdot)$ indicates a stop-gradient that treats its argument as a constant with zero derivative, preventing it from influencing the loss gradient during backpropagation. We find that a “gradient supervision” tone curve $\psi(z) = \log(z + \epsilon)$ with $\epsilon = 10^{-3}$ produces perceptually high quality results with minimal artifacts, implying a loss weighting term of $\psi'(\text{sg}(\hat{y}_i)) = (\text{sg}(\hat{y}_i) + \epsilon)^{-1}$ and final loss

$$\tilde{L}_\psi(\hat{y}, y) = \sum_i \left(\frac{\hat{y}_i - y_i}{\text{sg}(\hat{y}_i) + \epsilon} \right)^2. \quad (4)$$

This corresponds exactly to the relative MSE loss used to achieve unbiased results when training on noisy HDR path-tracing data in Noise2Noise [30]. The curve ψ is proportional to the μ -law function used for range compression in audio processing, and has previously been applied as a tonemapping function when supervising a network to map from a burst of LDR images to an HDR output [23].

4.2. Variable exposure training

In scenes with very high dynamic range, even a 10-14 bit raw image may not be sufficient for capturing both bright and dark regions in a single exposure. This is addressed by the “bracketing” mode included in many digital cameras,



(a) RawNeRF models trained with fixed vs. varying exposure
(b) Global and local tonemapping applied to RawNeRF trained on varying exposures

Figure 5. A fixed shutter speed is not sufficient for capturing the full dynamic range in scenes with extreme brightness variation. (a) For example, this scene requires variable exposure capture to avoid either poor quality in dark indoor regions or blown-out sky highlights. Only a RawNeRF model optimized using both short and long exposures recovers the full dynamic range. (b) This brightness variation is too high to visualize in a single image using a simple global sRGB gamma curve, requiring a more sophisticated local tonemapping algorithm (e.g., HDR+ postprocessing [20]).

where multiple images with varying shutter speeds are captured in a burst, then merged to take advantage of the bright highlights preserved in the shorter exposures and the darker regions captured with more detail in the faster exposures.

We can similarly take advantage of variable exposures in RawNeRF (Figure 5). Given a sequence of images I_i with exposure times t_i (and all other capture parameters held constant), we can “expose” RawNeRF’s linear space color output to match the brightness in image I_i by scaling it by the recorded shutter speed t_i . In practice, we find that varying exposures cannot be precisely aligned using shutter speed alone due to sensor miscalibration (see supplement). To correct for this, we add a learned per-color-channel scaling factor for each unique shutter speed present in the set of captured images, which we jointly optimize along with the NeRF network. The final RawNeRF “exposure” given a output color \hat{y}_i from the network is then $\min(\hat{y}_i^c \cdot t_i \cdot \alpha_{t_i}^c, 1)$, where c indexes color channels, and $\alpha_{t_i}^c$ is the learned scaling factor for shutter speed t_i and channel c (we constrain $\alpha_{t_{\max}}^c = 1$ for the longest exposure). We clip from above at 1 to account for the fact that pixels saturate in overexposed regions. This scaled and clipped value is passed to the previously described loss (Equation 4).

4.3. Implementation details

Our implementation is based on the mip-NeRF [3] codebase, which improves upon the positional encoding used in the original NeRF method. Please see that paper for further details on the MLP scene representation and volumetric rendering algorithm. Our only network architecture change is to modify the activation function for the MLP’s output

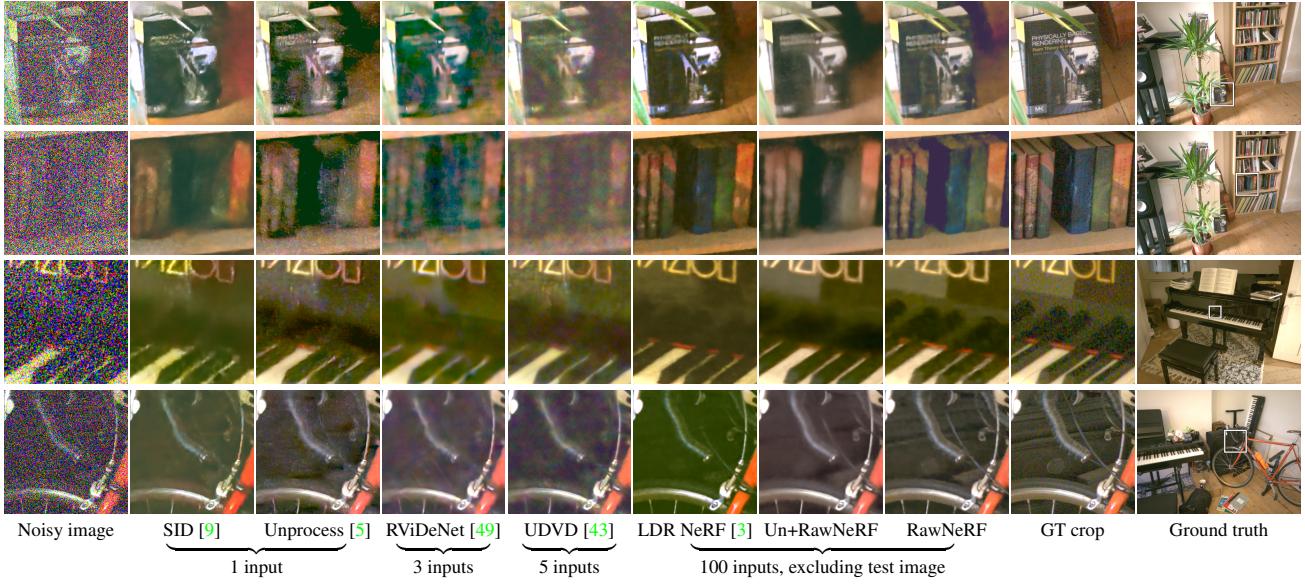


Figure 6. Example postprocessed and color-aligned patches from our real denoising dataset. RawNeRF produces the most detailed output in each case. All deep denoising methods (columns 2–5) receive the noisy test image as input, whereas NeRF variants (columns 6–8) perform both novel view synthesis and denoising.

Method	Num. inputs	Raw PSNR↑	Affine-aligned sRGB		
			PSNR↑	SSIM↑	LPIPS↓
Noisy input	-	54.38	10.24	0.035	0.733
SID [9]	1	-	21.62	0.525	0.547
Unprocess [5]	1	70.80	23.02	0.491	0.489
RViDeNet [49]	3	68.29	22.20	0.516	0.545
UDVD [43]	5	70.68	22.75	0.514	0.507
LDR NeRF [3]	$N - 1$	-	19.43	0.518	0.544
Un+RawNeRF	$N - 1$	67.99	23.35	0.531	0.507
RawNeRF	$N - 1$	67.20	23.53	0.536	0.501

Table 1. We compare RawNeRF’s denoising performance to various single and multi-image denoisers and NeRF ablations. Despite only being optimized on a single scene and never having seen even a noisy version of the test view, RawNeRF achieves results competitive with deep denoising methods trained on large image datasets. RawNeRF also outperforms NeRF trained on LDR sRGB images (LDR NeRF) and an ablation where RawNeRF’s inputs have been denoised using “Unprocess” (Un+RawNeRF).

color from a sigmoid to an exponential function to better parameterize linear radiance values. We use the Adam optimizer [25] with batches of 16k random rays sampled across all training images and a learning rate decaying from 10^{-3} to 10^{-5} over 500k steps of optimization.

We find that extremely noisy scenes benefit from a regularization loss on volume density to prevent partially transparent “floater” artifacts. We apply a loss on the variance of the weight distribution used to accumulate color values along the ray during volume rendering; please see the supplement for details.

As our raw input data is mosaicked, it only contains one color value per pixel. We only apply the loss to the active color channel for each pixel, such that optimizing NeRF effectively demosaics the input images. Since any resampling steps will effect the raw noise distribution, we do not undistort or downsample the inputs, and instead train using the full resolution mosaicked images (usually 12MP for our scenes). To achieve this, we use camera intrinsics to account for radial distortion when generating rays. We use full resolution postprocessed JPEG images to calculate camera poses as COLMAP [42] does not support raw images.

5. Results

We present results exploring two consequences of supervising NeRF with raw HDR data. First, we show that RawNeRF is surprisingly robust to high levels of noise, to the extent that it can act as a competitive multi-image denoiser when applied to wide-baseline images of a static scene. Second, we demonstrate the HDR view synthesis applications enabled by recovering a scene representation that preserves high dynamic range color values.

5.1. Denoising

Recent years have seen an increasing focus on developing deep learning methods for denoising images directly in the raw linear domain [5, 9]. This effort has expanded to include multi-image denoisers that can be applied to burst images or video frames [8, 43, 49]. These multi-image denoisers typically assume that there is a relatively small

amount of motion between frames, but that there may be large amounts of object motion within the scene. When nearby frames can be well aligned, these methods merge information from similar image patches (typically across 2-8 neighboring images) to outperform single image denoisers.

By comparison, NeRF (and by extension, RawNeRF) optimizes for a single scene reconstruction that is consistent with *all* input images. By specializing to wide-baseline static scenes and taking advantage of 3D multiview information, RawNeRF can aggregate observations from much more widely spaced input images than a typical multi-image denoising method.

Real dataset We collect a real world denoising dataset with 3 different scenes, each consisting of 101 noisy images and a clean reference image merged from stabilized long exposures. The first 100 images are taken handheld across a wide baseline (a standard forward-facing NeRF capture), using a fast shutter speed to accentuate noise. We then capture a stabilized burst of 50-100 longer exposures on a tripod and robustly merge them using HDR+ [20] to create a clean ground truth frame. One additional tripod image taken at the original fast shutter speed serves as a noisy input “base frame” for the deep denoising methods. All images are taken with an iPhone X at 12MP resolution using the wide-angle lens and saved as 12-bit raw DNG files.

Comparisons In Table 1 and Figure 6, we compare RawNeRF’s joint view synthesis and denoising performance to several recent deep single and multi-image denoising methods. Note that all denoisers require the noisy version of the test image as input, whereas RawNeRF and its ablations only require its camera pose.

We focus our comparison on methods explicitly designed to handle raw input images. Chen *et al.* [9] (SID) present a single image denoiser that maps from raw inputs to postprocessed LDR images and is trained on a large dataset of noisy raw and clean postprocessed image pairs collected by the authors. Brooks *et al.* [5] (Unprocess) is a method for training a raw single image denoiser on simulated raw data created from internet image datasets that transfers well to real raw images. RViDeNet [49] trains a raw video denoiser on a combination of Unprocessing-style synthetic data and a new real raw video dataset. Sheth *et al.* [43] (UDVD) present a “self-supervised” method for training a video denoiser only using noisy data, building on ideas from Noise2Noise [30] and blind-spot networks [29]. UDVD provides network weights specifically trained on the raw video dataset from RViDeNet. For all methods, we use publicly available code and pretrained model weights.

We also compare to two ablations of our method. LDR NeRF represents mip-NeRF [3] trained (as usual) in LDR sRGB space on images postprocessed by a minimal sRGB

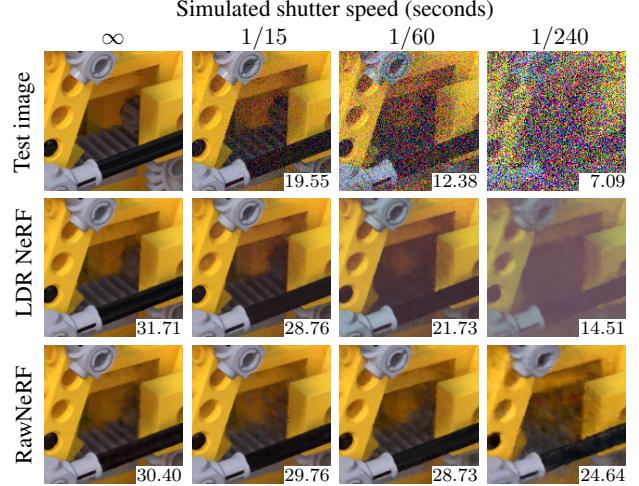


Figure 7. Example patches from the synthetic scene used in Table 2, annotated with sRGB PSNR for each inset. With perfectly clean inputs, training on LDR images is superior, but with any nonzero amount of noise, it is more beneficial to optimize NeRF in raw space, where the noise distribution remains unbiased.

Method	Simulated shutter speed (seconds)						
	∞	1/7	1/15	1/30	1/60	1/120	1/240
Noisy input	-	23.33	19.65	16.03	12.51	9.40	7.18
LDR NeRF	33.16	31.25	29.14	26.10	22.31	18.27	14.87
RawNeRF	32.15	32.11	31.94	31.59	30.94	29.69	27.73

Table 2. We perform an ablation study on a synthetically rendered raw dataset with 120 training images, simulating shot and read noise for 8 different shutter speeds. Here we report PSNR values in LDR sRGB space.

tonemapping pipeline. “Un+RawNeRF” preprocesses the training images using the single image raw denoiser from Brooks *et al.* [5] (“Unprocess”) before training RawNeRF.

All compared methods take mosaicked raw images as input. Every deep denoiser [5, 9, 43, 49] uses the noisy “base frame” as input, and the two multi-image denoising networks [43, 49] also receive the nearest images from the wide-baseline capture (based on camera position). We convert the 12-bit raw input to floating point by normalizing with the white and black levels. Since each method was trained on raw data from a different source, they impart different color tints to the output. So this not affect metrics, we calculate a per-color-channel affine transform that best matches each method’s raw output to the ground truth raw image. (The exceptions are SID and LDR NeRF, whose sRGB output we match to the postprocessed sRGB ground truth.) Our basic postprocessing pipeline for visualization and computing sRGB metrics is to apply a bilinear demosaic (when necessary), perform white balance/color correction, rescale white level, clip to [0, 1], and apply the sRGB gamma curve. Please see the supplement for details.

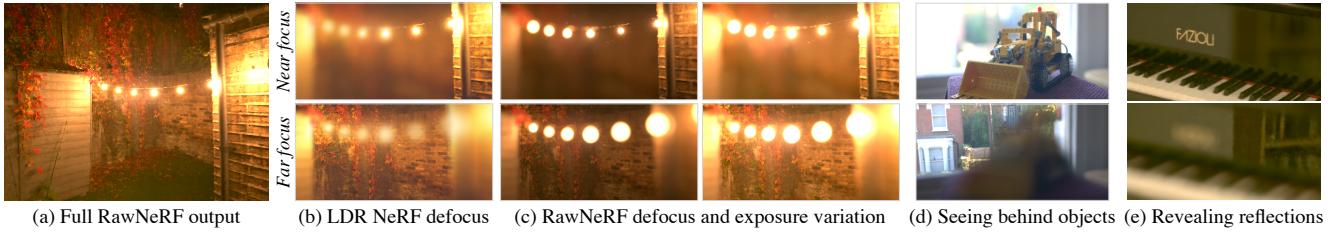


Figure 8. Synthetic defocus examples. In this nighttime garden scene (a), LDR NeRF cannot accurately render defocused bright highlights since it is trained on images that have already been tonemapped and clipped (b). RawNeRF recovers the linear intensity of the light sources such that applying defocus blur produces correctly oversaturated “bokeh balls” (c). Since RawNeRF is optimized for view synthesis from wide-baseline inputs, it can achieve 3D defocus effects not possible with a single image and depth map, such as revealing occluded parts of the background by focusing behind the foreground bulldozer (d) or focusing on the bookshelves reflected above the piano keys (e).

Analysis Despite simultaneously performing denoising and novel view synthesis, our method is competitive with all compared deep denoisers (Table 1, Figure 6). We suspect that the multi-image denoisers struggle to make use of the additional frames provided from the wide-baseline capture, as the camera movement is larger than in a typical sub-second burst or video clip. By comparison, RawNeRF, despite lacking any explicitly learned image priors, clean training data, or even a “base frame” input image, produces high quality outputs by combining information from across all input images in its reconstruction. Despite the fact that LDR NeRF is directly trained to minimize mean-squared error in sRGB space, RawNeRF achieves significantly better sRGB metrics. We also find that applying a single image denoiser to the inputs before training RawNeRF results in oversmoothed renderings (Un+RawNeRF).

Synthetic noise ablation In Table 2 and Figure 7, we demonstrate the impact of noise level on RawNeRF image quality. For training, we render 120 linear HDR images using the *Lego* scene from NeRF [37], borrowing color correction, white balance, and noise parameters from our iPhone captures’ EXIF metadata to “unprocess” this data into raw space [5]. Since the renderings have a large amount of empty space, we report sRGB PSNR on the object only, by using the provided alpha masks (otherwise error from the background pixels heavily penalizes LDR NeRF). Even in this synthetic setting free from camera miscalibration issues, we can clearly observe the color bias and loss of detail caused by training LDR NeRF on postprocessed noisy data.

5.2. HDR view synthesis applications

Modifying exposure and tonemapping Figures 1, 2, 4, 5, and 8 include examples of varying the exposure level and tonemapping algorithm for images output by RawNeRF, which exist in linear HDR space and can thus be postprocessed like a raw photo from a digital camera. Please see our supplement and video for many more examples.

Synthetic defocus Given a full 3D model of a scene, physically-based renderers accurately simulate camera lens defocus effects by tracing rays refracted through each lens element [26], but this process is extremely computationally expensive. A reasonably convincing and much cheaper solution is to apply a varying blur kernel to different depth layers of the scene and composite them together [2, 45]. In Figure 8, we apply this synthetic defocus rendering model to sets of RGBA depth layers precomputed from trained RawNeRF models (similar to a multiplane image [53]). As shown by Zhang *et al.* [51], recovering linear HDR color is critical for achieving the characteristic oversaturated “bokeh balls” around defocused bright light sources.

6. Discussion

We have demonstrated the benefits of training NeRF directly on linear raw camera images. However, this modification is not without tradeoffs. Most digital cameras can only save raw images at full resolution with minimal compression, resulting in huge storage requirements when capturing tens or hundreds of images per scene. Our method is also dependent on COLMAP’s [42] robustness for computing camera poses, preventing us from capturing scenes below a certain light level. This could potentially be addressed by jointly optimizing RawNeRF and the input camera poses [33, 46]. Finally, despite its robustness to noise, RawNeRF cannot be considered a general purpose denoiser as it cannot handle scene motion and requires orders of magnitude more computation than a feed-forward network.

Despite these shortcomings, we believe that RawNeRF represents a step toward robust, high quality capture of real world environments. Training on raw images with variable exposure allows us to capture scenes with a much wider dynamic range, and robustness to noise makes reconstructing dark nighttime captures possible. Lifting these constraints greatly increases the fraction of the world that can be reconstructed and explored with photorealistic view synthesis.

References

- [1] Benjamin Attal, Eliot Laidlaw, Aaron Gokaslan, Changil Kim, Christian Richardt, James Tompkin, and Matthew O’Toole. Törf: Time-of-flight radiance fields for dynamic scene view synthesis, 2021. 2
- [2] Jonathan T. Barron, Andrew Adams, YiChang Shih, and Carlos Hernández. Fast bilateral-space stereo for synthetic defocus. *CVPR*, 2015. 3, 8, 17
- [3] 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. 5, 6, 7, 13
- [4] Joshua Batson and Loic Royer. Noise2self: Blind denoising by self-supervision. *ICML*, 2019. 3
- [5] Tim Brooks, Ben Mildenhall, Tianfan Xue, Jiawen Chen, Dillon Sharlet, and Jonathan T. Barron. Unprocessing images for learned raw denoising. *CVPR*, 2019. 3, 6, 7, 8, 16
- [6] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. *SIGGRAPH*, 2001. 2
- [7] Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédéric Durand. Learning photographic global tonal adjustment with a database of input / output image pairs. *CVPR*, 2011. 3
- [8] Chen Chen, Qifeng Chen, Minh Do, and Vladlen Koltun. Seeing motion in the dark. *ICCV*, 2019. 3, 6
- [9] Chen Chen, Qifeng Chen, Jia Xu, and Vladlen Koltun. Learning to see in the dark. *CVPR*, 2018. 3, 6, 7
- [10] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 1984. 3
- [11] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *TIP*, 2007. 3
- [12] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. *SIGGRAPH*, 1997. 3, 4
- [13] Thibaud Ehret, Axel Davy, Jean-Michel Morel, Gabriele Facciolo, and Pablo Arias. Model-blind video denoising via frame-to-frame training. *CVPR*, 2019. 3
- [14] Gabriel Eilertsen, Joel Kronander, Gyorgy Denes, Rafał Mantiuk, and Jonas Unger. Hdr image reconstruction from a single exposure using deep cnns. *ACM Transactions on Graphics (TOG)*, 2017. 3
- [15] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world’s imagery. *CVPR*, 2016. 2
- [16] Alessandro Foi, Mejdi Trimeche, Vladimir Katkovnik, and Karen O. Egiazarian. Practical poissonian-gaussian noise modeling and fitting for single-image raw-data. *IEEE Transactions on Image Processing*, 2008. 3
- [17] Michaël Gharbi, Jiawen Chen, Jonathan T Barron, Samuel W Hasinoff, and Frédéric Durand. Deep bilateral learning for real-time image enhancement. *ACM Transactions on Graphics (TOG)*, 2017. 1, 3
- [18] Clement Godard, Kevin Matzen, and Matt Uyttendaele. Deep burst denoising. *ECCV*, 2018. 3
- [19] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. *SIGGRAPH*, pages 43–54, 1996. 2
- [20] Samuel W. Hasinoff, Dillon Sharlet, Ryan Geiss, Andrew Adams, Jonathan T. Barron, Florian Kainz, Jiawen Chen, and Marc Levoy. Burst photography for high dynamic range and low-light imaging on mobile cameras. *SIGGRAPH Asia*, 2016. 2, 3, 4, 5, 7
- [21] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *SIGGRAPH Asia*, 2018. 2
- [22] Yuanming Hu, Hao He, Chenxi Xu, Baoyuan Wang, and Stephen Lin. Exposure: A white-box photo post-processing framework. *ACM Transactions on Graphics (TOG)*, 2018. 3
- [23] Nima Khademi Kalantari and Ravi Ramamoorthi. Deep high dynamic range imaging of dynamic scenes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2017)*, 36(4), 2017. 3, 5
- [24] Jun-Yan Zhu, Kangle Deng, Andrew Liu, and Deva Ramanan. Depth-supervised nerf: Fewer views and faster training for free. *arXiv:2107.02791*, 2021. 2
- [25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 6
- [26] Craig Kolb, Don Mitchell, and Pat Hanrahan. A realistic camera model for computer graphics. *SIGGRAPH*, 1995. 3, 8
- [27] Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-based neural rendering with per-view optimization. *Eurographics*, 2021. 2
- [28] Alexander Krull, Tim-Oliver Buchholz, and Florian Jug. Noise2void-learning denoising from single noisy images. *CVPR*, 2019. 3
- [29] Samuli Laine, Tero Karras, Jaakko Lehtinen, and Timo Aila. High-quality self-supervised deep image denoising. *NeurIPS*, 2019. 3, 7
- [30] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. *ICML*, 2018. 3, 4, 5, 7
- [31] Marc Levoy and Pat Hanrahan. Light field rendering. *SIGGRAPH*, 1996. 2
- [32] Xin Li, Bahadir Gundurk, and Lei Zhang. Image demosaicing: A systematic survey. *Visual Communications and Image Processing 2008*, 2008. 3
- [33] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. BARF: Bundle-adjusting neural radiance fields. *ICCV*, 2021. 8
- [34] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *SIGGRAPH*, 2019. 2
- [35] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. *CVPR*, 2021. 2

- [36] Ben Mildenhall, Jonathan T. Barron, Jiawen Chen, Dillon Sharlet, Ren Ng, and Robert Carroll. Burst denoising with kernel prediction networks. *CVPR*, 2018. 3
- [37] 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. 1, 2, 3, 4, 8, 16
- [38] Tobias Plötz and Stefan Roth. Benchmarking denoising algorithms with real photographs. *CVPR*, 2017. 3
- [39] Gernot Riegler and Vladlen Koltun. Free view synthesis. *ECCV*, 2020. 2
- [40] Gernot Riegler and Vladlen Koltun. Stable view synthesis. *CVPR*, 2021. 2
- [41] Darius Rückert, Linus Franke, and Marc Stamminger. Adop: Approximate differentiable one-pixel point rendering, 2021. 2
- [42] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. *CVPR*, 2016. 6, 8, 13
- [43] Dev Yashpal Sheth, Sreyas Mohan, Joshua Vincent, Ramon Manzorro, Peter A. Crozier, Mitesh M. Khapra, Eero P. Simoncelli, and Carlos Fernandez-Granda. Unsupervised deep video denoising. *ICCV*, 2021. 3, 6, 7
- [44] Pratul P. Srinivasan, Rahul Garg, Neal Wadhwa, Ren Ng, and Jonathan T. Barron. Aperture supervision for monocular depth estimation. *CVPR*, 2018. 3
- [45] 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. *ACM Trans. Graph.*, 2018. 3, 8, 17
- [46] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. NeRF—: Neural radiance fields without known camera parameters. *arXiv:2102.07064*, 2021. 8
- [47] Yi Wei, Shaohui Liu, Yongming Rao, Wang Zhao, Jiwen Lu, and Jie Zhou. Nerfingmvs: Guided optimization of neural radiance fields for indoor multi-view stereo. *ICCV*, 2021. 2
- [48] Christopehr Choy Animashree Anandkumar Minsu Cho Yoonwoo Jeong, Seokjun Ahn and Jaesik Park. Self-calibrating neural radiance fields. *ICCV*, 2021. 2
- [49] Huanjing Yue, Cong Cao, Lei Liao, Ronghe Chu, and Jingyu Yang. Supervised raw video denoising with a benchmark dataset on dynamic scenes. *CVPR*, 2020. 3, 6, 7
- [50] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising. *IEEE Transactions on Image Processing*, 2017. 3
- [51] Xuaner Zhang, Kevin Matzen, Vivien Nguyen, Dillon Yao, You Zhang, and Ren Ng. Synthetic defocus and look-ahead autofocus for casual videography. *SIGGRAPH*, 2019. 3, 8, 17
- [52] Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew Davison. In-place scene labelling and understanding with implicit scene representation. *ICCV*, 2021. 2
- [53] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *SIGGRAPH*, 2018. 2, 8, 17

A. 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.

B. Additional qualitative results

We include additional qualitative results for both dark (Figure 9) and high contrast scenes (Figure 10). We urge the reader to view our supplemental video as the results are more compelling when animated.

C. Training details

C.1. Full derivation of gradient-weighted loss

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

$$L_\psi(\hat{y}, y) = \sum_i (\psi(\hat{y}_i) - \psi(y_i))^2 \quad (5)$$

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

$$\begin{aligned} \psi(\hat{y}_i) - \psi(y_i) &\approx \psi(\hat{y}_i) - (\psi(\hat{y}_i) + \psi'(\hat{y}_i)(y_i - \hat{y}_i)) \\ &= \psi'(\hat{y}_i)(\hat{y}_i - y_i). \end{aligned} \quad (6)$$

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 $x_i = \text{E}[y_i]$ over the course of training.

If we use a weighted L2 loss, then as we train the network we will have $\hat{y}_i \rightarrow \text{E}[y_i] = x_i$ in expectation (where x_i is the true signal value). This means that the terms summed in our gradient-weighted loss

$$\tilde{L}_\psi(\hat{y}, y) = \sum_i [\psi'(\text{sg}(\hat{y}_i))(\hat{y}_i - y_i)]^2 \quad (7)$$

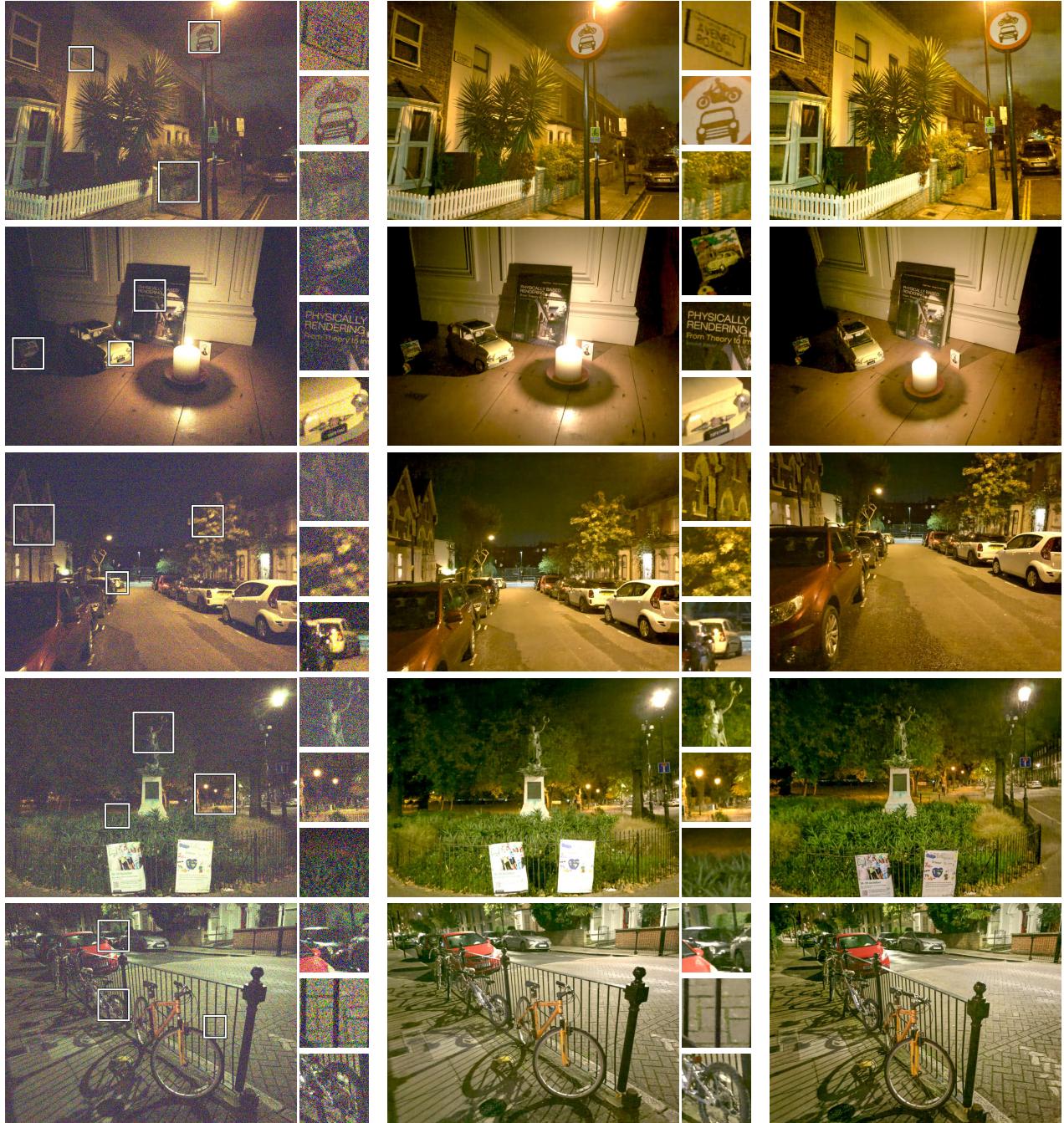
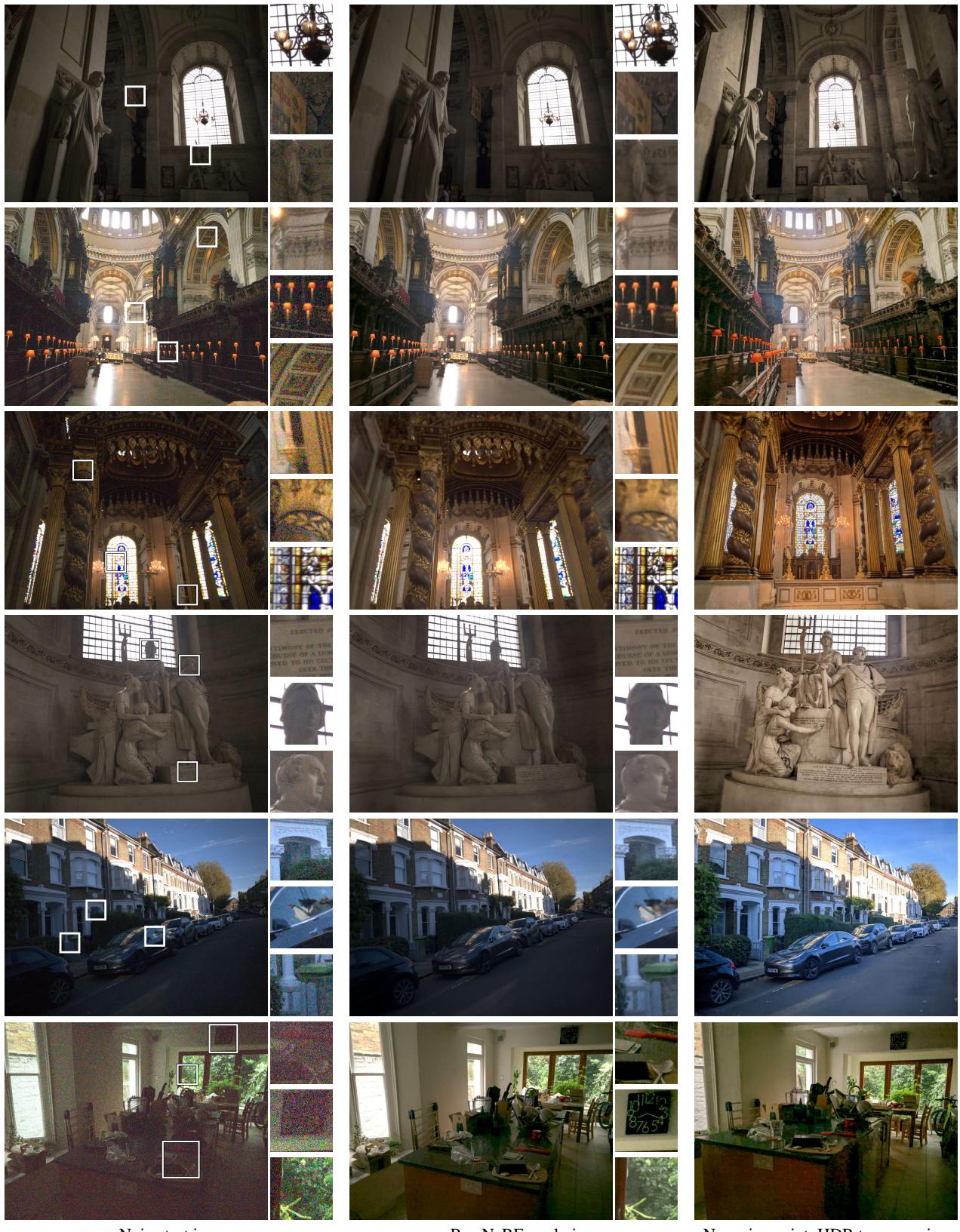


Figure 9. RawNeRF in the dark.



Noisy test image

RawNeRF rendering

New viewpoint, HDR tonemapping

Figure 10. Examples of scenes with very high dynamic range.

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 7 is a linear approximation of the gradient of the tonemapped loss 5:

$$\nabla_{\theta} L_{\psi}(\hat{y}, y) = \sum_i \nabla_{\theta} (\psi(\hat{y}_i) - \psi(y_i))^2 \quad (8)$$

$$= \sum_i 2(\psi(\hat{y}_i) - \psi(y_i))\psi'(\hat{y}_i)\nabla_{\theta} y_i \quad (9)$$

$$\approx \sum_i 2(\psi'(\hat{y}_i)(\hat{y}_i - y_i))\psi'(\hat{y}_i)\nabla_{\theta} y_i \quad (10)$$

$$= \sum_i 2(\psi'(\text{sg}(\hat{y}_i))(\hat{y}_i - y_i))\psi'(\text{sg}(\hat{y}_i))\nabla_{\theta} y_i \quad (11)$$

$$= \nabla_{\theta} \tilde{L}_{\psi}(\hat{y}, y). \quad (12)$$

In line 10 we substitute the linearization from 6, and in line 11 we exploit the fact that a stop-gradient has no effect for expressions that will not be further differentiated.

C.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 [3]), these weights are

$$w_i = (1 - \exp(-\Delta_i \sigma_i)) \exp\left(-\sum_{j < i} \Delta_j \sigma_j\right). \quad (13)$$

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

$$\mathcal{L}_w = \text{Var}_{X \sim p_w}(X) = \mathbb{E}_{X \sim p_w} [(X - \mathbb{E}[X])^2] \quad (14)$$

Calculating the mean (expected depth):

$$\mathbb{E}_{X \sim p_w}[X] = \sum_i \int_{t_{i-1}}^{t_i} \frac{w_i}{\Delta_i} t dt \quad (15)$$

$$= \sum_i \frac{w_i}{\Delta_i} \frac{t_i^2 - t_{i-1}^2}{2} \quad (16)$$

$$= \sum_i w_i \frac{t_i + t_{i-1}}{2}. \quad (17)$$

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

$$\text{Var}_{X \sim p_w}(X) = \mathbb{E}_{X \sim p_w} [(X - \mathbb{E}[X])^2] \quad (18)$$

$$= \sum_i \int_{t_{i-1}}^{t_i} \frac{w_i}{\Delta_i} (t - \bar{t})^2 dt \quad (19)$$

$$= \sum_i \frac{w_i}{\Delta_i} \frac{(t_i - \bar{t})^3 - (t_{i-1} - \bar{t})^3}{3} \quad (20)$$

$$= \sum_i w_i \frac{(t_i - \bar{t})^2 + (t_i - \bar{t})(t_{i-1} - \bar{t}) + (t_{i-1} - \bar{t})^2}{3} \quad (21)$$

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.

C.3. Findings with alternate loss functions

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

$$\psi'(\text{sg}(\hat{y}_i)) = \frac{1}{\text{sg}(\hat{y}_i) + \epsilon} \quad (22)$$

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.

C.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 [42] 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.

D. Data capture and postprocessing details

D.1. Data capture

We captured all images using a 2017 iPhone X with the Halide app¹ 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 1/100s 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 DNG² 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 matrix³:

$$C_{\text{rgb-xyz}} = \begin{bmatrix} 0.4124564 & 0.3575761 & 0.1804375 \\ 0.2126729 & 0.7151522 & 0.0721750 \\ 0.0193339 & 0.1191920 & 0.9503041 \end{bmatrix} \quad (23)$$

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

$$C_{\text{all}} = \text{rownorm}((C_{\text{rgb-xyz}} C_{\text{ccm}})^{-1}) \quad (24)$$

where `rownorm` normalizes each to sum to 1.

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

$$\gamma_{\text{sRGB}}(z) = \begin{cases} 12.92z & z \leq 0.0031308 \\ 1.055z^{1/2.4} - 0.055 & z > 0.0031308 \end{cases} \quad (25)$$

D.2. 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.)

$$z \leftarrow \frac{z - b}{w - b} \quad (26)$$

4. Apply bilinear demosaicking (when necessary).
5. Apply elementwise white balance gains.

$$z \leftarrow \frac{z}{g_{\text{wb}}} \quad (27)$$

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

$$z \leftarrow C_{\text{all}} z \quad (28)$$

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

$$z \leftarrow \frac{z}{\text{percentile}(z, p)} \quad (29)$$

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

$$z \leftarrow \text{clip}(z, 0, 1) \quad (30)$$

¹<https://halide.cam/>
²http://www.adobe.com/content/dam/acom/en/products/photoshop/pdfs/dng_spec_1.4.0.0.pdf

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

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

$$z \leftarrow \gamma_{\text{sRGB}}(z) \quad (31)$$

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.

D.3. 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 11 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 $y_{t_i}^c$ for each Bayer filter channel (R, G1, G2, B) over the entire 12MP sensor. Specifically, we plot:

$$\frac{y_{t_i}^c}{t_i} \cdot \frac{t_{\max}}{y_{t_{\max}}^c} \quad (32)$$

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 11 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).

We show an example of the resulting qualitative color shift in Figure 12 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 12b. 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

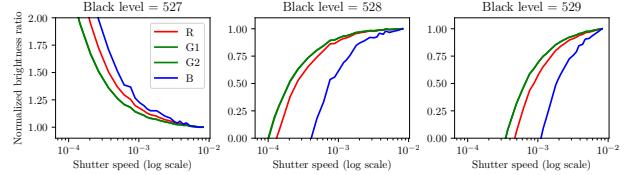


Figure 11. Camera shutter speed miscalibration. We plot normalized brightness for each Bayer color channel, relative to its value at the longest shutter speed. For a perfectly calibrated sensor, these lines would all be at a constant height of 1. We show plots using both the true black level (528) and surrounding values.



Figure 12. (a) Fast and (c) slow captures of the *testyuca* scene, with brightness normalized by shutter speed (heavily downsampled to minimize noise). These two images should match perfectly, but have a perceptible color difference due to the miscalibration documented in Section D.3 and Figure 11. (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).

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 mirrorless camera with a better sensor may avoid this issue.

E. Comparison and ablation details

E.1. 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

$$a = \frac{\bar{xy} - \bar{x}\bar{y}}{\bar{x}^2 - \bar{x}^2} = \frac{\text{Cov}(x, y)}{\text{Var}(x)}, \quad (33)$$

$$b = \bar{y} - a\bar{x} \quad (34)$$

to get the least-squares fit of an affine transform $ax + b \approx y$ (here \bar{z} indicates the mean over all elements of z). We then apply the inverse transform as $(y - b)/a$ to match the es-

Method	∞	Simulated shutter speed (seconds)					
		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 3. Unmasked LDR sRGB PSNRs for the ablation study on our synthetic *Lego* scene data.

timated y to x . In the case where matching happens in the raw domain, we postprocess $(y - b)/a$ through our standard pipeline (Section D.2) 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 [5] 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.

E.2. 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 file⁴ provided by the NeRF authors [37], 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 3 (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.

⁴<https://drive.google.com/file/d/1RjwxZCUoPlUgEWIUiuCmMmG0AhvV8A2Q/view?usp=sharing>

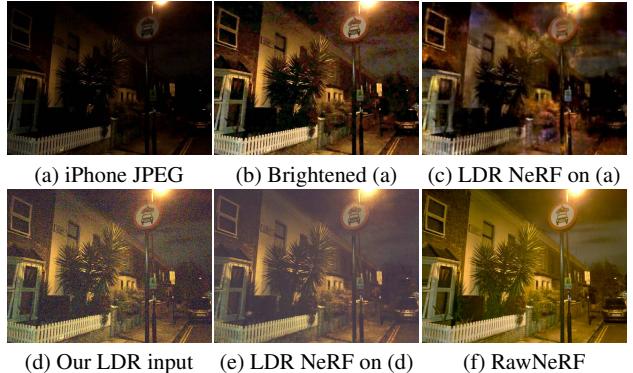


Figure 13. Comparison of training LDR NeRF using sRGB images either directly from the iPhone camera or from our simplified pipeline. (a) The JPEG image from the phone is extremely dark, so we brighten it for visualization (b). We also brighten the resulting LDR NeRF rendering (c), thereby revealing its 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.

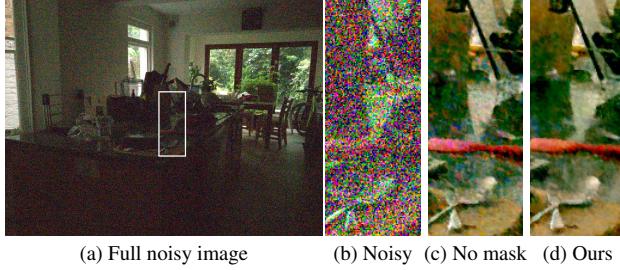
F. Further qualitative ablations

F.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 Figure 13 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.

F.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 14, 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.



(a) Full noisy image

(b) Noisy

(c) No mask

(d) Ours

Figure 14. 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.

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.

G. Synthetic defocus rendering model

To render defocused images, we use a similar rendering model as prior work that has addressed this task [2, 45, 51]. To avoid prohibitively expensive rendering speeds, we first precompute a multiplane image [53] 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.

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 .

Algorithm 1 Synthetic defocus rendering

```

procedure DEFOCUS( $c_{\text{mpi}}$ ,  $\alpha_{\text{mpi}}$ ,  $i_{\text{focus}}$ ,  $\Delta_r$ ,  $\Delta_d$ )
     $C \leftarrow 0$ 
    for  $i = 0, \dots, N - 1$  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

```

H. Scene index

We provide various details about each scene shown in the paper and video in Table 4.

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

Table 4. A summary of image metadata for our scenes. Figure from the supplement are indicated using the prefix “A”.