RobustNeRF: Ignoring Distractors with Robust Losses[4]

Sara Sabour[1,2] Suhani Vora[1] Daniel Duckworth[1] Ivan Krasin[1]
David J. Fleet[1,2] Andrea Tagliasacchi[1,2,3]
Google Research, Brain Team[1] University of Toronto[2] Simon Fraser University[3]

Abstract

Neural radiance fields (NeRF) excel at synthesizing new views given multi-view, calibrated images of a static scene. When scenes include distractors, which are not persistent during image capture (moving objects, lighting variations, shadows), artifacts appear as view-dependent effects or 'floaters'. To cope with distractors, we advocate a form of robust estimation for NeRF training, modeling distractors in training data as outliers of an optimization problem. Our method successfully removes outliers from a scene and improves upon our baselines, on synthetic and real-world scenes. Our technique is simple to incorporate in modern NeRF frameworks, with few hyper-parameters. It does not assume a priori knowledge of the types of distractors, and is instead focused on the optimization problem rather than pre-processing or modeling transient objects. More results on our page `https://robustnerf.github.io/public`.

1. Introduction

The ability to understand the structure of a *static* 3D scene from 2D images alone is a fundamental problem is computer vision [44]. It finds applications in AR/VR for mapping virtual environments [6, 36, 61], in autonomous robotics for action planning [1], and in photogrammetry to create digital copies of real-world objects [34].

Neural fields [55] have recently revolutionized this classical task, by storing 3D representations within the weights of a neural network [39]. These representations are optimized by back-propagating image differences. When the fields store view-dependent *radiance* and volumetric rendering is employed [21], we can capture 3D scenes with photo-realistic accuracy, and we refer to the generated representation as Neural Radiance Fields, or NeRF [25]).

Training of NeRF models generally requires a large collection of images equipped with accurate camera calibration, which can often be recovered via structure-from-motion [37]. Behind its simplicity, NeRF hides several assumptions. As models are typically trained to minimize error in RGB color space, it is of paramount importance that images are photometrically consistent – two photos taken from the same vantage point should be *identical* up to noise. Unless one employs a method explicitly accounting for it [35], one should manually hold a camera's focus, exposure, white-balance, and ISO fixed.

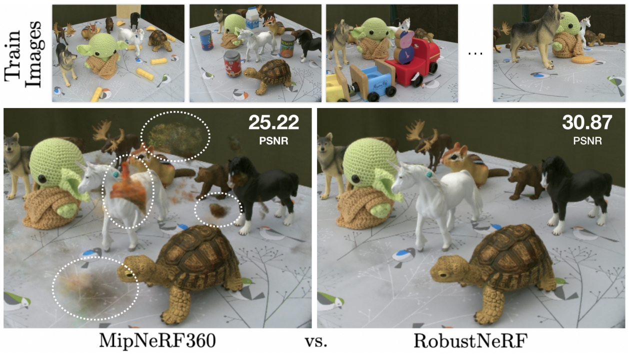

Figure 1. NeRF assumes photometric consistency in the observed images of a scene. Violations of this assumption, as with the images in the top row, yield reconstructed scenes with inconsistent content in the form of "floaters" (highlighted with ellipses). We introduce a *simple* technique that produces *clean* reconstruction by automatically *ignoring distractors without explicit supervision*.

However, properly configuring one's camera is not all that is required to capture high-quality NeRFs – it is also important to avoid *distractors*: anything that isn't persistent throughout the entire capture session. Distractors come in many shapes and forms, from the hard-shadows cast by the operators as they explore the scene to a pet or child casually walking within the camera's field of view. Distractors are tedious to *remove* manually, as this would require pixel-by-pixel labeling. They are also tedious to *detect*, as typical NeRF scenes are trained from hundreds of input images, and the types of distractors are not known a priori. If distractors are *ignored*, the quality of the reconstruction scene suffers significantly; see Figure 1.

In a typical capture session, one does not have the ability to capture multiple images of the same scene from the same vantage point, rendering distractors challenging to model mathematically. More specifically, while view-dependent effects are what give NeRF their realistic look, *how can the*

[4] Work done at Google Research.

model tell the difference between a distractor and a view-dependent effect?

Despite the challenges, the research community has devised several approaches to overcome this issue:

- If distractors are known to belong to a specific class (e.g., people), one can remove them with a pre-trained semantic segmentation model [35, 43] – this process does *not generalize* to "unexpected" distractors such as shadows.
- One can model distractors as per-image *transient* phenomena, and control the balance of transient/persistent modeling [23] – however, it is *difficult to tune* the losses that control this Pareto-optimal objective.
- One can model data in time (i.e., high-framerate video) and decompose the scene into static and dynamic (i.e., distractor) components [53] – but this clearly only applies to *video* rather than photo collection captures.

Conversely, we approach the problem of distractors by modeling them as *outliers* in NeRF optimization.

We analyze the aforementioned techniques through the lens of robust estimation, allowing us to understand their behavior, and to design a method that is not only simpler to implement but also more effective (see Figure 1). As a result, we obtain a method that is straightforward to implement, requires minimal-to-no hyper-parameter tuning, and achieves state-of-the-art performance. We evaluate our method:

- quantitatively, in terms of reconstruction with synthetically, yet photo-realistically, rendered data;
- qualitatively on publicly available datasets (often fine-tuned to work effectively with previous methods);
- on a new collection of natural and synthetic scenes, including those autonomously acquired by a robot, allowing us to demonstrate the sensitivity of previous methods to hyper-parameter tuning.

2. Related Work

We briefly review the basics and notation of Neural Radiance Fields. We then describe recent progress in NeRF research, paying particular attention to techniques for modeling of static/dynamic scenes.

Neural Radiance Fields. A neural radiance field (NeRF) is a continuous volumetric representation of a 3D scene, stored within the parameters of a neural network $\boldsymbol{\theta}$. The representation maps a position $\mathbf{x}$ and view direction $\mathbf{d}$ to a *view-dependent* RGB color and *view-independent* density:

$$\left.\begin{matrix}\mathbf{c}(\mathbf{x},\mathbf{d})\\ \sigma(\mathbf{x})\end{matrix}\right\} f(\mathbf{x},\mathbf{d};\boldsymbol{\theta}) \tag{1}$$

This representation is trained from a collection, $\{(\mathbf{C}_i, \mathbf{T}_i)\}$, of images $\mathbf{C}_i$ with corresponding calibration parameters $\mathbf{T}_i$ (camera extrinsics and intrinsics).

During training the calibration information is employed to convert each pixel of the image into a ray $\mathbf{r}{=}(\mathbf{o},\mathbf{d})$, and rays are drawn randomly from input images to form a training mini-batch ($\mathbf{r}{\sim}\mathbf{C}_i$). The parameters $\boldsymbol{\theta}$ are optimized to correctly predict the colors of the pixels in the batch via the L2 photometric-reconstruction loss:

$$\mathcal{L}_{\text{rgb}}(\boldsymbol{\theta}) = \sum_i \mathbb{E}_{\mathbf{r}\sim\mathbf{C}_i}\left[\mathcal{L}^{\mathbf{r},i}_{\text{rgb}}(\boldsymbol{\theta})\right] \tag{2}$$

$$\mathcal{L}^{\mathbf{r},i}_{\text{rgb}}(\boldsymbol{\theta}) = ||\mathbf{C}(\mathbf{r};\boldsymbol{\theta}) - \mathbf{C}_i(\mathbf{r})||_2^2 \tag{3}$$

Parameterizing the ray as $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, the NeRF model image $\mathbf{C}(\mathbf{r};\boldsymbol{\theta})$ is generated pixel-by-pixel volumetric rendering based on $\sigma(\cdot)$ and $\mathbf{c}(\cdot)$ (e.g., see [25, 42]).

Recent progress on NeRF models. NeRF models have recently been extended in several ways. A major thread has been the speedup of training [15, 27] and inference [6, 13], enabling today's models to be trained in minutes [27], and rendered on mobile in real-time [6]. While initially restricted to forward-facing scenes, researchers quickly found ways to model real-world 360° scenes [4, 59], and to reduce the required number of images, via sensor fusion [35] or hand-designed priors [28]. We can now deal with image artifacts such as motion blur [22], exposure [24], and lens distortion [14]. And the requirement of (precise) camera calibrations is quickly being relaxed with the introduction of techniques for local camera refinement [8, 19], or direct inference [58]. While a NeRF typically represents geometry via volumetric density, there exist models custom-tailored to predict surfaces [29, 51], which can be extended to use predicted normals to significantly improve reconstruction quality [50, 57]. Given high-quality normals [47], inferring the (rendering) structure of a scene becomes a possibility [5]. We also note recent papers about additional applications to generalization [56], semantic understanding [48], generative modeling [33], robotics [1], and text-to-3D [31].

Modeling non-static scenes. For unstructured scenes like those considered here, the community has focused on reconstructing both static and non-static elements from video. The most direct approach, treating time as an auxiliary input, leads to cloudy geometry and a lack of fine detail [11, 54]. Directly optimizing per-frame latent codes as an auxiliary input has proved more effective [17, 30, 53]. The most widely-adopted approach is to fit a time-conditioned deformation field mapping 3D points between pairs of frames [18, 49] or to a canonical coordinate frame [9, 10, 20, 32, 45]. Given how sparsely space-time is sampled, all methods require careful regularization, optimization, or additional training signals to achieve acceptable results.

Relatively little attention has been given to *removing* non-static elements. One common approach is to segment and ignore pixels which are likely to be distractors [35, 43]. While this eliminates larger objects, it fails to account for

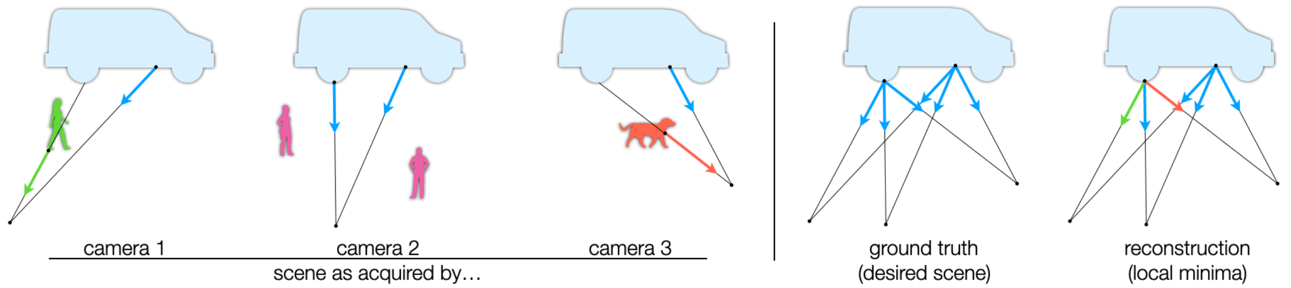

Figure 2. **Ambiguity –** A simple 2D scene where a static object (blue) is captured by three cameras. During the first and third capture the scene is not photo-consistent as a distractor was within the field of view. Not photo-consistent portions of the scene can end up being encoded as view-dependent effects – even when we assume ground truth geometry.

secondary effects like shadows. Prior attempts to model distractors as outliers still leave residual cloudy geometry [23].

3. Method

The classical NeRF training losses (3) are effective for capturing scenes that are photometrically consistent, leading to the photo-realistic novel-view synthesis that we are now accustomed to seeing in recent research. However, "*what happens when there are elements of the scene that are not persistent throughout the entire capture session?*" Simple examples of such scenes include those in which an object is only present in some fraction of the observed images, or may not remain in the same position in all observed images. For example, Figure 2 depicts a 2D scene comprising a persistent object (the truck), along with several transient objects (e.g., people and a dog). While rays in blue from the three cameras intersect the truck, the green and orange rays from cameras 1 and 3 intersect transient objects. For video capture and spatio-temporal NeRF models, the persistent objects comprise the "static" portion of the scene, while the rest would be called the "dynamic".

3.1. Sensitivity to outliers

For Lambertian scenes, photo-consistent structure is view independent, as scene radiance only depends on the incident light [16]. For such scenes, view-dependent NeRF models like (1), trained by minimizing (3), admit local optima in which transient objects are explained by view-dependent terms. Figure 2 depicts this, with the outgoing color corresponding to the memorized color of the outlier – i.e. view-dependent radiance. Such models exploit the view-dependent capacity of the model to over-fit observations, effectively memorizing the transient objects. One can alter the model to remove dependence on $\mathbf{d}$, but the L2 loss remains problematic as least-squares (LS) estimators are sensitive to outliers, or heavy-tailed noise distributions.

Under more natural conditions, dropping the Lambertian assumption, the problem becomes more complex as *both* non-Lambertian reflectance phenomena and outliers can be explained as view-dependent radiance. While we want the models to capture photo-consistent view-dependent radiance, outliers and other transient phenomena should ideally be ignored. And in such cases, optimization with an L2 loss (3) yields significant errors in reconstruction; see Figure 1. Problems like these are pervasive in NeRF model fitting, especially in uncontrolled environments with complex reflectance, non-rigidity, or independently moving objects.

3.2. Robustness to outliers

Robustness via semantic segmentation. One way to reduce outlier contamination during NeRF model optimization is to rely on an *oracle* $\mathbf{S}$ that specifies whether a given pixel $\mathbf{r}$ from image i is an outlier, and should therefore be excluded from the empirical loss, replacing (3) with:

$$\mathcal{L}^{\mathbf{r},i}_{\text{oracle}}(\boldsymbol{\theta}) = \mathbf{S}_i(\mathbf{r}) \cdot ||\mathbf{C}(\mathbf{r};\boldsymbol{\theta}) - \mathbf{C}_i(\mathbf{r})||_2^2 \tag{4}$$

In practice, a *pre-trained* (semantic) segmentation network $\mathcal{S}$ might be used as an oracle, $\mathbf{S}_i{=}\mathcal{S}(\mathbf{C}_i)$. For example, Nerf-in-the-wild [23] employed a semantic segmenter to remove pixels occupied by people, as they represent outliers in the context of photo-tourism. Urban Radiance Fields [35] segmented out sky pixels, while LOL-NeRF [33] ignored pixels not belonging to faces. The obvious problem with this approach is the need for an oracle that detects outliers for arbitrary distractors.

Robust estimators. Another way to reduce sensitivity to outliers is to replace the conventional L2 loss (3) with a *robust loss* (e.g., [2, 41]), so that photometrically-inconsistent observations can be down-weighted during optimization. Given a robust kernel $\kappa(\cdot)$, we rewrite our training loss as:

$$\mathcal{L}^{\mathbf{r},i}_{\text{robust}}(\boldsymbol{\theta}) = \kappa(||\mathbf{C}(\mathbf{r};\boldsymbol{\theta}) - \mathbf{C}_i(\mathbf{r})||_2) \tag{5}$$

where $\kappa(\cdot)$ is positive and monotonically increasing. Mip-NeRF [3], for example, employs an L1 loss $\kappa(\epsilon){=}|\epsilon|$, which provides some degree of robustness to outliers during NeRF training. Given our analysis, a valid question is whether we can straightforwardly employ a robust kernel to approach our problem, and if so, given the large variety of robust kernels [2], which is the kernel of choice.

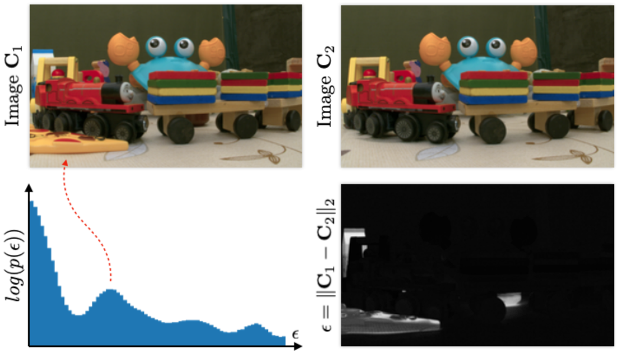

Figure 3. **Histograms** – Robust estimators perform well when the distribution of residuals agrees with the one implied by the estimator (e.g., Gaussian for L2, Laplacian for L1). Here we visualize the ground-truth distribution of residuals (bottom-left), which is hardly a good match with any simple parametric distribution.

Unfortunately, as discussed above, outliers and non-Lambertian effects can *both* be modelled as view-dependent effects (see Figure 3). As a consequence, with simple application of robust estimators it can be difficult to separate signal from noise. Figure 4 shows examples in which outliers are removed, but fine-grained texture and view-dependent details are also lost, or conversely, fine-grained details are preserved, but outliers cause artifacts in the reconstructed scene. One can also observe mixtures of these cases in which details are not captured well, nor are outliers fully removed. We find that this behaviour occurs consistently for many different robust estimators and parameter settings.

Training time can also be problematic. The robust estimator gradient w.r.t. model parameters can be expressed using the chain rule as

$$\left.\frac{\partial\kappa(\epsilon(\boldsymbol{\theta}))}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}^{(t)}} = \left.\frac{\partial\kappa(\epsilon)}{\partial\epsilon}\right|_{\epsilon(\boldsymbol{\theta}^{(t)})} \cdot \left.\frac{\partial\epsilon(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}^{(t)}} \tag{6}$$

The second factor is the classical NeRF gradient. The first factor is the kernel gradient evaluated at the *current* error residual $\epsilon(\boldsymbol{\theta}^{(t)})$. During training, large residuals can *equivalently* come from high-frequency details that have not yet been learnt, or they may arise from outliers (see Figure 4 (bottom)). This explain why robust optimization, implemented as (5), should not be expected to decouple high-frequency details from outliers. Further, when *strongly* robust kernels are employed, like redescending estimators, this also explains the loss of visual fidelity. That is, because the gradient of (large) residuals get down-weighted by the (small) gradients of the kernel, *slowing down* the learning of these fine-grained details (see Figure 4 (top)).

3.3. Robustness via Trimmed Least Squares

In what follows we advocate a form of iteratively reweighted least-squares (IRLS) with a Trimmed least squares (LS) loss for NeRF model fitting.

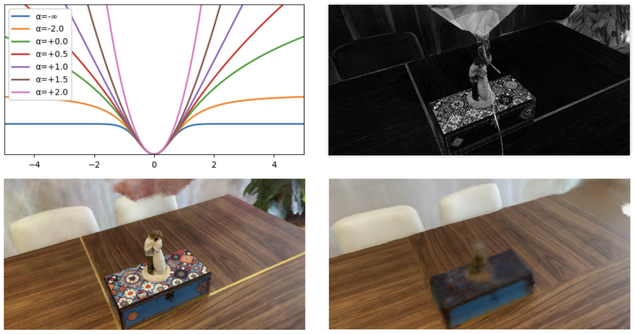

Figure 4. **Kernels –** (top-left) The family of robust kernels from [2], including L2 (α=2), Charbonnier (α=1) and Geman-McClure (α=−2). (top-right) Mid-training, residual magnitudes are similar for distractors and fine-grained details, and pixels with large residuals are learned more slowly, as the gradient of redescending kernels flattens out. (bottom-right) Kernels that are too aggressive in down-weighting large residuals remove both outliers and high-frequency detail. (bottom-left) Less aggressive kernels do not effectively remove outliers.

Iteratively Reweighted least Squares. IRLS is a widely used method for robust estimation that involves solving a sequence of weighted LS problems, the weights of which are adapted to reduce the influence of outliers. To that end, at iteration t, one can write the loss as

$$\begin{aligned} \mathcal{L}^{\mathbf{r},i}_{\text{robust}}(\boldsymbol{\theta}^{(t)}) &= \omega(\boldsymbol{\epsilon}^{(t-1)}(\mathbf{r})) \cdot ||\mathbf{C}(\mathbf{r};\boldsymbol{\theta}^{(t)}) - \mathbf{C}_i(\mathbf{r})||_2^2 \\ \boldsymbol{\epsilon}^{(t-1)}(\mathbf{r}) &= ||\mathbf{C}(\mathbf{r};\boldsymbol{\theta}^{(t-1)}) - \mathbf{C}_i(\mathbf{r})||_2 \end{aligned} \tag{7}$$

For weight functions given by $\omega(\epsilon){=}\epsilon^{-1}\cdot\partial\kappa(\epsilon)/\partial\epsilon$ one can show that, under suitable conditions, the iteration converges to a local minima of (5) (see [41, Sec. 3]).

This framework admits a broad family of losses, including maximum likelihood estimators for heavy-tailed noise processes. Examples in Figure 4 include the Charbonnier loss (smoothed L1), and more aggressive redescending estimators such as the Lorentzian or Geman-McClure [2]. The objective in (4) can also be viewed as a weighted LS objective, the binary weights of which are provided by an oracle. And, as discussed at length below, one can also view several recent methods like NeRFW [23] and D^{2}NeRF [53] through the lens of IRLS and weighted LS.

Nevertheless, choosing a suitable weight function $\omega(\epsilon)$ for NeRF optimization is non-trivial, due in large part to the intrinsic ambiguity between view-dependent radiance phenomena and outliers. One might try to solve this problem by learning a neural weight function [40], although generating enough annotated training data might be prohibitive. Instead, the approach taken below is to exploit inductive biases in the structure of outliers, combined with the simplicity of a robust, trimmed LS estimator.

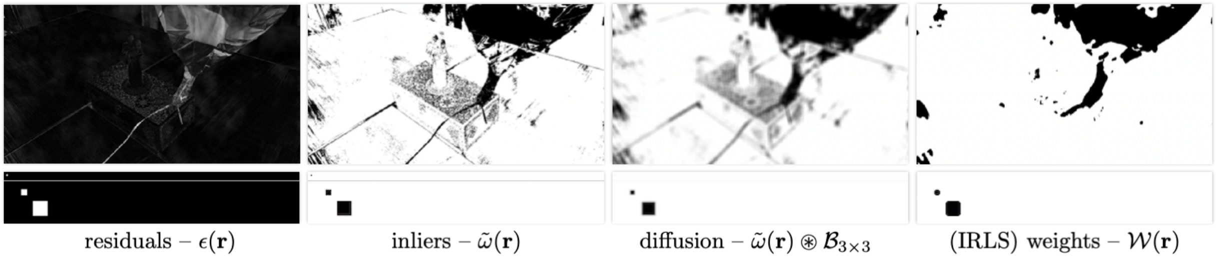
residuals – $\epsilon(\mathbf{r})$
inliers – $\tilde{\omega}(\mathbf{r})$
diffusion – $\tilde{\omega}(\mathbf{r}) \circledast \mathcal{B}_{3\times3}$
(IRLS) weights – $\mathcal{W}(\mathbf{r})$

Figure 5. **Algorithm** – We visualize our weight function computed by residuals on two examples: (top) the residuals of a (mid-training) NeRF rendered from a *training* viewpoint, (bottom) a toy residual image containing residual of small spatial extent (dot, line) and residuals of large spatial extent (squares). Notice residuals with large magnitude but small spatial extent (texture of the box, dot, line) are included in the optimization, while weaker residuals with larger spatial extent are excluded. Note that while we operate on patches, we visualize the weight function on the whole image to facilitate visualization.

Trimmed Robust Kernels. Our goal is to develop a weight function for use in iterative weighted LS optimization that is simple and captures useful inductive biases for NeRF optimization. For simplicity we opt for a binary weight function with intuitive parameters that adapts naturally through model fitting so that fine-grained image details that are not outliers can be learned quickly. It is also important to capture the structured nature of typical outliers, contrary to the typical i.i.d. assumption in most robust estimator formulations. To this end the weight function should capture spatial smoothness of the outlier process, recognizing that objects typically have continuous local support, and hence outliers are expected to occupy large and connected regions of an image (e.g., the silhouette of a person to be segmented out from a photo-tourism dataset).

Surprisingly, a relatively simple weight function embodies these properties and performs extremely well in practice. The weight function is based on so-called *trimmed estimators* that are used in trimmed least-squares, like that used in trimmed ICP [7]. We first *sort* residuals, and assume that residuals below a certain percentile are inliers. Picking the 50% percentile for convenience (i.e., median), we define

$$\tilde{\omega}(\mathbf{r}) = \epsilon(\mathbf{r}) \leq \mathcal{T}_\epsilon \ , \quad \mathcal{T}_\epsilon = \text{Median}_{\mathbf{r}}\{\epsilon(\mathbf{r})\} \ . \qquad (8)$$

To capture spatial smoothness of outliers we further spatially diffuse inlier/outlier labels ω with a 3×3 box kernel $\mathcal{B}_{3\times3}$. Formally, we define

$$\mathcal{W}(\mathbf{r}) = (\tilde{\omega}(\mathbf{r}) \circledast \mathcal{B}_{3\times3}) \geq \mathcal{T}_\circledast \ , \quad \mathcal{T}_\circledast = 0.5 \ . \qquad (9)$$

This tends to remove high-frequency details from being classified as outliers, allow them to be captured by the NeRF model during optimization (see Figure 5).

While the trimmed weight function (9) improves the robustness of model fitting, it also misclassifies fine-grained texture details early in training where the NeRF model first captures coarse-grained structure. These localized texture elements may emerge but only after very long training times. We find that stronger inductive bias to spatially coherence allows fine-grained details to be learned more quickly. To that end, we aggregate the detection of outliers on 16×16 neighborhoods; i.e., we label entire 8×8 patches as outliers or inliers based on the behavior of $\mathcal{W}$ in the 16×16 neighborhood of the patch . Formally, denoting the $N\times N$ neighborhood of pixels around $\mathbf{r}$ as $\mathcal{R}_N(\mathbf{r})$, we define

$$\omega(\mathcal{R}_8(\mathbf{r})) = \mathbb{E}_{\mathbf{s}\sim\mathcal{R}_{16}(\mathbf{r})}\left[\mathcal{W}(\mathbf{s})\right] \geq \mathcal{T}_\mathcal{R} \ , \quad \mathcal{T}_\mathcal{R} = 0.6 \ . \quad (10)$$

Note that this robust weight function evolves during optimization, as one expects with IRLS where the weights are a function of the residuals at the previous iteration. That is, the labeling of pixels as inliers/outliers *changes* during training, and settles around masks similar to the one an oracle would provide as training converges (see Figure 6).

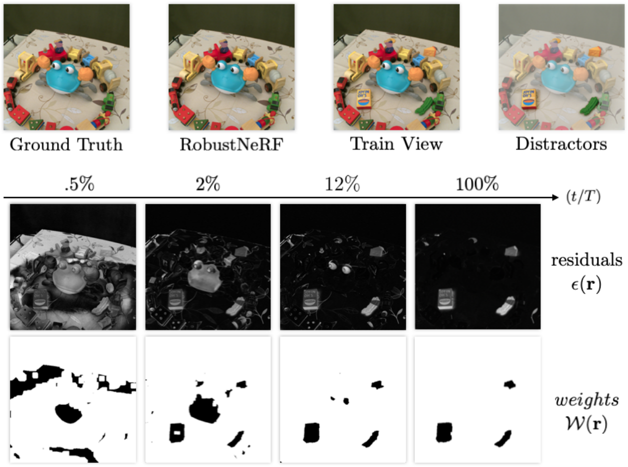
Ground Truth
RobustNeRF
Train View
Distractors
.5%
2%
12%
100%
(t/T)
residuals $\epsilon(\mathbf{r})$
weights $\mathcal{W}(\mathbf{r})$

Figure 6. **Residuals** – For the dataset shown in the top row, we visualize the dynamics of the RobustNeRF training residuals, which show how over time the estimated distractor weights go from being random ((t/T)=0.5%) to identify distractor pixels ((t/T)=100%) without any explicit supervision.

4. Experiments

We implement our robust loss function in the MultiNeRF codebase [26] and apply it to mip-NeRF 360 [4]. We dub this method "RobustNeRF". To evaluate RobustNeRF,

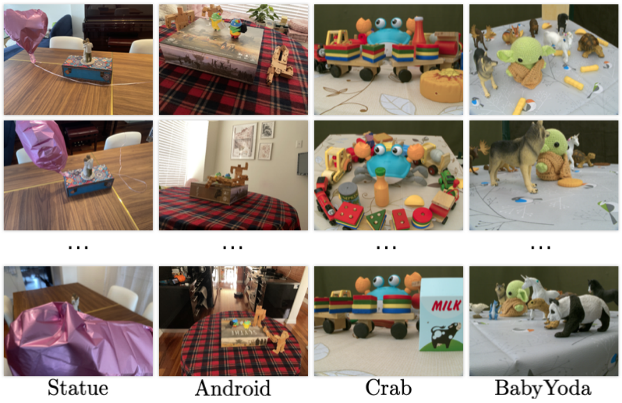

Figure 7. **Dataset –** Sample training images showing the distractors in each scene. Statue and Android were acquired manually, and the others with a robotic arm. In the robotic setting we have pixel-perfect alignment of distractor vs. distractor-free images.

we compare against baselines on several scenes containing different types of distractors. Where possible, we quantitatively compare reconstructions to held-out, distraction-free images; we report three metrics, averaged across held-out frames, namely, PSNR, SSIM [52], and LPIPS [60].

We compare different methods on two collections of scenes, i.e., those provided by the authors of D^{2}NeRF, and *novel* datasets described below. We also present a series of illustrative experiments on synthetic scenes, shedding light on RobustNeRF's efficacy and inner workings.

4.1. Baselines

We compare RobustNeRF to variants of mip-NeRF 360 optimized with different loss functions (L_2, L_1, and Charbonnier). These variants serve as natural baselines for models with limited or no robustness to outliers.We also compare to D^{2}NeRF, a recent method for reconstructing dynamic scenes from monocular *video*. Unlike our method, D^{2}NeRF is designed to *reconstruct* distractors rather than discard them. While D^{2}NeRF is presented as a method for monocular video, it does not presuppose temporal continuity, and can be directly applied to unordered images. We omit additional comparisons to NeRF-W as its performance falls short of D^{2}NeRF [53]. For more details on model training, see the supplementary material (Section 6.2).

4.2. Datasets – Figure 7

In addition to scenes from D^{2}NeRF, we introduce a set of natural and synthetic scenes. They facilitate the evaluation of RobustNeRF's effectiveness on illustrative use cases, and they enable empirical analysis under controlled conditions.

Natural scenes. We capture *four* natural scenes exemplifying different types of distractors. Scenes are captured in two settings, an apartment and a robotics lab. Distractor objects are moved, or are allowed to move, between frames to simulate capture over extended periods of time. We vary the number of unique distractors from 1 (Statue) to 150 (BabyYoda), and their movements. Unlike prior work on monocular video, frames are captured without a clear temporal ordering (see Figure 7). We capture additional frames *without distractors* to enable quantitative evaluations. Camera poses are estimated using COLMAP [38]. We provide a full description of each scene in the supplementary material (subsubsection 6.1.1).

Synthetic scenes. To further evaluate RobustNeRF, we generate synthetic scenes using the Kubric dataset generator [12]. Each scene is constructed by placing a set of simple geometries in an empty, texture-less room. In each scene, a subset of objects remain fixed while the other objects (i.e., distractors) change position from frame to frame. By varying the number of objects, their size, and the way they move, we control the level of distraction in each scene. We use these scenes to examine RobustNeRF's sensitivity to its hyperparameters, see supplementary material (subsubsection 6.3.3).

4.3. Evaluation

We evaluate RobustNeRF on its ability to *ignore* distractors while accurately reconstructing the static elements of a scene. We train RobustNeRF, D^{2}NeRF, and variants of mip-NeRF 360 on scenes where distraction-free frames are available. Models are *trained* on frames with distractors and *evaluated* on distractor-free frames.

Comparison to mip-NeRF 360 – Figure 8. On natural scenes, RobustNeRF generally outperforms variants of mip-NeRF 360 by 1.3 to 4.7 dB in PSNR. As L_2, L_1, and Charbonnier losses weigh all pixels equally, the model is forced to represent, rather than ignore, distractors as "clouds" with view-dependent appearance. We find clouds to be most apparent when distractors remain stationary for multiple frames. In contrast, RobustNeRF's loss isolates distractor pixels and assigns them a weight of zero (see Figure 6). To establish an upper bound on reconstruction accuracy, we train mip-NeRF 360 with Charbonnier loss on distraction-free versions of each scene, the images for which are taken from (approximately) the same viewpoints. Reassuringly, RobustNeRF when trained on distraction-free frames, achieves nearly identical accuracy; see Figure 11.

While RobustNeRF consistently outperforms mip-NeRF 360, the gap is smaller in the Apartment scenes (Statue, Android) than the Robotics Lab scenes (Crab, BabyYoda). This can be explained by challenging background geometry, errors in camera parameter estimation, and imperceptible changes to scene appearance. For further discussion, see the supplementary material (subsubsection 6.3.1).

Comparison to D^{2}NeRF – Figure 9. Quantitatively, RobustNeRF matches or outperforms D^{2}NeRF by as much

	Statue			Android			Crab			BabyYoda		
	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑
mip-NeRF 360 (L_2)	0.36	0.66	19.09	0.40	0.65	19.35	0.27	0.77	25.73	0.31	0.75	22.97
mip-NeRF 360 (L_1)	0.30	0.72	19.55	0.40	0.66	19.38	0.22	0.79	26.69	0.22	0.80	26.15
mip-NeRF 360 (Ch.)	0.30	0.73	19.64	0.40	0.66	19.53	0.21	0.80	27.72	0.23	0.80	25.22
D^{2}NeRF	0.48	0.49	19.09	0.43	0.57	20.61	0.42	0.68	21.18	0.44	0.65	17.32
RobustNeRF	**0.28**	**0.75**	**20.89**	**0.31**	**0.65**	**21.72**	**0.21**	**0.81**	**30.75**	**0.20**	**0.83**	**30.87**
mip-NeRF 360 (clean)	0.19	0.80	23.57	0.31	0.71	23.10	0.16	0.84	32.55	0.16	0.84	32.63

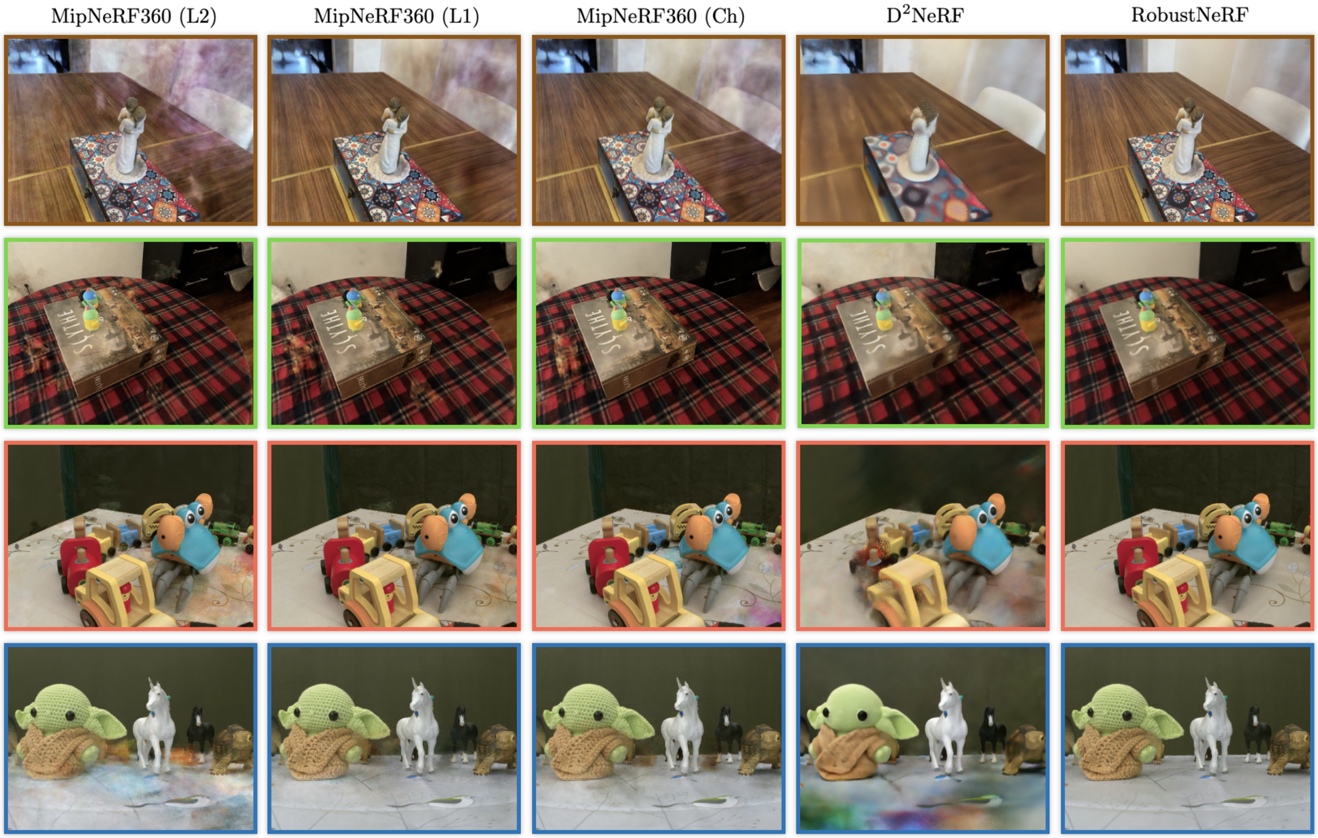

Figure 8. **Evaluation on Natural Scenes –** RobustNeRF outperforms baselines and D^{2}NeRF [53] on novel view synthesis with real-world captures; see supplementary material (subsubsection 6.3.4) for more qualitative results. D^{2}NeRF underperforms on robotic scenes with multiple, varied distractors. On manually acquired scenes, Statute and Android, RobustNeRF yields accurate, detailed models without floaters; the corresponding evaluation ground truth images are captured in the wild, with imperfections that yield lower PSNR for all methods.

as 12 dB PSNR depending on the number of unique outlier objects in the capture. In Statue and Android, one and three non-rigid objects are moved around the scene, respectively. D^{2}NeRF is able to model these objects and thus separate them from the scenes' static content. In the remaining scenes, a much larger pool of 100 to 150 unique, non-static objects are used – too many for D^{2}NeRF to model effectively. As a result, "cloud" artifacts appear in its static representation, similar to those produced by mip-NeRF 360. In contrast, RobustNeRF identifies non-static content as outliers and omits it during reconstruction. Although both methods use a similar number of parameters, D^{2}NeRF's peak memory usage is 2.3x higher than RobustNeRF and 37x higher when normalizing for batch size. This is a direct consequence of model architecture: D^{2}NeRF is tailored to simultaneously modeling static and dynamic content and thus merits higher complexity. To remain comparable, we limit image resolution to 0.2 megapixels for all experiments.

Ablations – Figure 10. We ablate each element of the RobustNeRF loss on the crab scene, and compare to an upper bound on reconstruction accuracy provided by mip-NeRF 360 trained on distractor-free (clean) images taken from the same viewpoints. Our trimmed robust estimator (8) successfully eliminates distractors at the expense of high fre-

	Car			Cars			**Bag**			Chairs			Pillow		
	LPIPS↓	MS-SSIM↑	PSNR↑	LPIPS↓	MS-SSIM↑	PSNR↑	LPIPS↓	MS-SSIM↑	PSNR↑	LPIPS↓	MS-SSIM↑	PSNR↑	LPIPS↓	MS-SSIM↑	PSNR↑
NeRF-W [23]	.218	.814	24.23	.243	.873	24.51	.139	.791	20.65	.150	.681	23.77	.088	.935	28.24
NSFF [18]	.200	.806	24.90	.620	.376	10.29	.108	.892	25.62	.682	.284	12.82	.782	.343	4.55
NeuralDiff [46]	.065	.952	31.89	.098	.921	25.93	.117	.910	29.02	.112	.722	24.42	.565	.652	20.09
D^{2}NeRF [53]	.062	.975	34.27	.090	.953	26.27	.076	.979	34.14	.095	.707	24.63	.076	.979	36.58
RobustNeRF	**.013**	**.988**	**37.73**	**.063**	**.957**	**26.31**	**.006**	**.995**	**41.82**	**.007**	**.992**	**41.23**	**.018**	**.990**	**38.95**

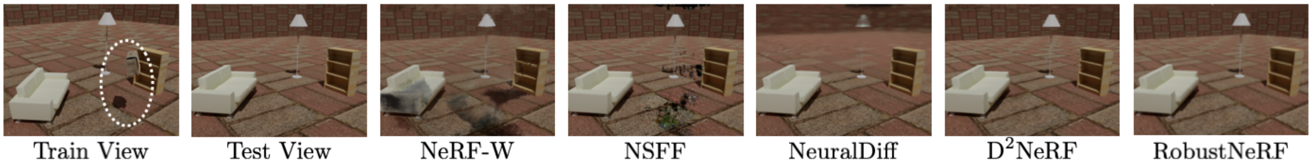
Train View Test View NeRF-W NSFF NeuralDiff D^{2}NeRF RobustNeRF

Figure 9. **Evaluations on D^{2}NeRF Synthetic Scenes –** Quantitative and qualitative evaluations on the Kubric synthetic dataset introduced by D^{2}NeRF, consisting of 200 training frames (with distractor) and 100 novel views for evaluation (without distractor).

	LPIPS↓	SSIM↑	PSNR↑	Updates to PSNR=30
mip-NeRF 360 (L_2)	0.31	0.75	22.97	–
+ robust (8)	0.39	0.60	18.21	–
+ smoothing (9)	0.22	0.81	30.01	250K
+ patching (10)	**0.21**	**0.81**	**30.75**	70K
oracle (clean)	0.16	0.84	32.55	25K

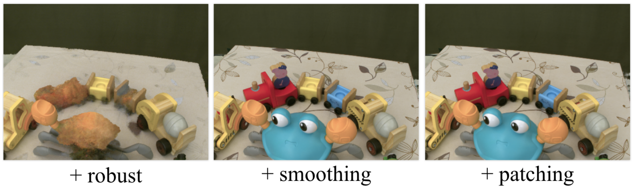
+ robust + smoothing + patching

Figure 10. **Ablations –** Blindly trimming the loss causes details to be lost. Smoothing recovers fine-grained detail, while patch-based evaluation speeds up training and adds more detail. Patching enables the model to reach PSNR of 30, almost 4× faster.

quency texture, yielding lower PSNR. Adding smoothing (9), high frequency detail is recovered, at the cost of longer training times. With the spatial window (10), RobustNeRF training time is on-par with mip-NeRF 360.

Sensitivity – Figure 11. We find that RobustNeRF is remarkably robust to the amount of clutter in a dataset. We define an image as "cluttered" if it contains some number of distractor pixels. The figure shows how the reconstruction accuracy of RobustNeRF and mip-NeRF 360 depends on the fraction of training images with distractors, keeping the training set size constant. As the fraction increases, mip-NeRF 360's accuracy steadily drops from 33 to 25 dB, while RobustNeRF's remains steadily above 31 dB throughout. In the distraction-free regime, we find that RobustNeRF mildly under-performs mip-NeRF 360, both in reconstruction quality and the time needed for training. This follows from the statistical inefficiency induced by the trimmed estimator (8), for which a percentage of pixels will be discarded even if they do not correspond to distractors.

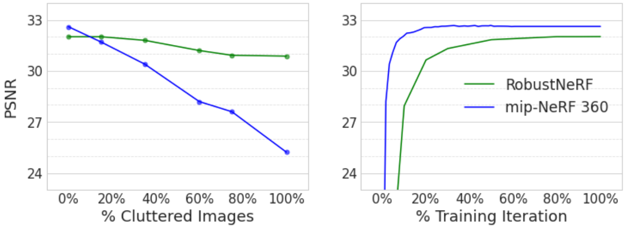

Figure 11. **Sensitivity and Limitations –** (left) Reconstruction accuracy for BabyYoda as we increase the fraction of train images with distractors. (right) Accuracy vs training time on *clean* BabyYoda images (distractor-free).

5. Conclusions

We address a central problem in training NeRF models, namely, optimization in the presence of distractors, such as transient or moving objects and photometric phenomena that are not persistent throughout the capture session.

Viewed through the lens of robust estimation, we formulate training as a form of iteratively re-weighted least squares, with a variant of trimmed LS, and an inductive bias on the smoothness of the outlier process. RobustNeRF is surprisingly simple, yet effective on a wide range of datasets. RobustNeRF is shown to outperform recent state-of-the-art methods [4, 53], qualitatively and quantitatively, on a suite of synthetic datasets, common benchmark datasets, and new datasets captured by a robot, allowing fine-grained control over distractors for comparison with previous methods. While our experiments explore robust estimation in the context of mip-NeRF 360, the RobustNeRF loss can be incorporated within other NeRF models.

Limitations. While RobustNeRF performs well on scenes with distractors, the loss entails some statistical inefficiency. On clean data, this yields somewhat poorer reconstructions, often taking longer to train (see Figure 11). Future work will consider very small distractors, which may require adaptation of the spatial support used for outlier/inlier decisions. It would also be interesting to learn a neural weight function, further improving RobustNeRF; active learning may be useful in this context. Finally, it would be interesting to include our robust loss in other NeRF frameworks.

References

[1] Michal Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager. Vision-only robot navigation in a neural radiance world. *IEEE Robotics and Automation Letters*, 2022. 1, 2

[2] Jonathan T. Barron. A general and adaptive robust loss function. *Proc. CVPR*, 2019. 3, 4

[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. 3

[4] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proc. CVPR*, 2022. 2, 5, 8, 14

[5] Mark Boss, Andreas Engelhardt, Abhishek Kar, Yuanzhen Li, Deqing Sun, Jonathan T. Barron, Hendrik P.A. Lensch, and Varun Jampani. SAMURAI: Shape And Material from Unconstrained Real-world Arbitrary Image collections. In *Proc. NeurIPS*, 2022. 2

[6] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. *arXiv preprint arXiv:2208.00277*, 2022. 1, 2

[7] Dmitry Chetverikov, Dmitry Svirko, Dmitry Stepanov, and Pavel Krsek. The trimmed iterative closest point algorithm. In *International Conference on Pattern Recognition*, 2002. 5

[8] Shin-Fang Chng, Sameera Ramasinghe, Jamie Sherrah, and Simon Lucey. Garf: Gaussian activated radiance fields for high fidelity reconstruction and pose estimation. *arXiv e-prints*, 2022. 2

[9] Yilun Du, Yinan Zhang, Hong-Xing Yu, Joshua B Tenenbaum, and Jiajun Wu. Neural radiance flow for 4d view synthesis and video processing. In *Proc. ICCV*. IEEE Computer Society, 2021. 2

[10] Jiemin Fang, Taoran Yi, Xinggang Wang, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Matthias Nießner, and Qi Tian. Fast dynamic radiance fields with time-aware neural voxels. *arXiv preprint arXiv:2205.15285*, 2022. 2

[11] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proc. ICCV*, 2021. 2

[12] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam Laradji, Hsueh-Ti (Derek) Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, Cengiz Oztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, and Andrea Tagliasacchi. Kubric: a scalable dataset generator. In *Proc. CVPR*, 2022. 6, 14

[13] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *Proc. ICCV*, 2021. 2

[14] Yoonwoo Jeong, Seokjun Ahn, Christopher Choy, Anima Anandkumar, Minsu Cho, and Jaesik Park. Self-calibrating neural radiance fields. In *Proc. ICCV*, 2021. 2

[15] Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. Relu fields: The little non-linearity that could. *TOG (Proc. SIGGRAPH)*, 2022. 2

[16] KN Kutulakos and SM Seitz. A theory of shape by space carving. *IJCV*, 2000. 3

[17] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, et al. Neural 3d video synthesis from multi-view video. In *Proc. CVPR*, 2022. 2

[18] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proc. CVPR*, 2021. 2, 8

[19] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. In *Proc. CVPR*, 2021. 2

[20] Jia-Wei Liu, Yan-Pei Cao, Weijia Mao, Wenqiao Zhang, David Junhao Zhang, Jussi Keppo, Ying Shan, Xiaohu Qie, and Mike Zheng Shou. Devrf: Fast deformable voxel radiance fields for dynamic scenes. *arXiv preprint arXiv:2205.15723*, 2022. 2

[21] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM TOG*, 2019. 1

[22] Li Ma, Xiaoyu Li, Jing Liao, Qi Zhang, Xuan Wang, Jue Wang, and Pedro V. Sander. Deblur-nerf: Neural radiance fields from blurry images. In *Proc. CVPR*, 2022. 2

[23] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proc. CVPR*, 2021. 2, 3, 4, 8

[24] Ben Mildenhall, Peter Hedman, Ricardo Martin-Brualla, Pratul P. Srinivasan, and Jonathan T. Barron. NeRF in the dark: High dynamic range view synthesis from noisy raw images. In *Proc. CVPR*, 2021. 2, 15

[25] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Proc. ECCV*, 2020. 1, 2

[26] Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, Peter Hedman, Ricardo Martin-Brualla, and Jonathan T. Barron. Multinerf: a code release for Mip-NeRF 360, Ref-NeRF, and RawNeRF, 2022. 5, 14

[27] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *TOG (Proc. SIGGRAPH)*, 2022. 2

[28] Michael Niemeyer, Jonathan T. Barron, Ben Mildenhall, Mehdi S. M. Sajjadi, Andreas Geiger, and Noha Radwan. Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs. In *Proc. CVPR*, 2022. 2

[29] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *Proc. ICCV*, 2021. 2

[30] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M Seitz. Hypernerf: a higher-dimensional representation for topologically varying neural radiance fields. *ACM Transactions on Graphics (TOG)*, 2021. 2, 15

[31] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint 2209.14988*, 2022. 2

[32] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proc. CVPR*, 2021. 2

[33] Daniel Rebain, Mark Matthews, Kwang Moo Yi, Dmitry Lagun, and Andrea Tagliasacchi. LOLNeRF: Learn from One Look. In *Proc. CVPR*, 2022. 2, 3

[34] Jeremy Reizenstein, Roman Shapovalov, Philipp Henzler, Luca Sbordone, Patrick Labatut, and David Novotny. Common objects in 3d: Large-scale learning and evaluation of real-life 3d category reconstruction. In *Proc. ICCV*, 2021. 1

[35] Konstantinos Rematas, Andrew Liu, Pratul P. Srinivasan, Jonathan T. Barron, Andrea Tagliasacchi, Tom Funkhouser, and Vittorio Ferrari. Urban radiance fields. *Proc. CVPR*, 2022. 1, 2, 3

[36] Antoni Rosinol, John J Leonard, and Luca Carlone. Nerf-slam: Real-time dense monocular slam with neural radiance fields. *arXiv preprint arXiv:2210.13641*, 2022. 1

[37] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proc. CVPR*, 2016. 1, 13

[38] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *Proc. ECCV*, 2016. 6

[39] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Proc. NeurIPs*, 2019. 1

[40] Weiwei Sun, Wei Jiang, Andrea Tagliasacchi, Eduard Trulls, and Kwang Moo Yi. ACNe: Attentive Context Normalization for Robust Permutation-Equivariant Learning. *Proc. CVPR*, 2020. 4

[41] Andrea Tagliasacchi and Hao Li. Modern techniques and applications for real-time non-rigid registration. In *Proc. SIGGRAPH Asia (Technical Course Notes)*, 2016. 3, 4

[42] Andrea Tagliasacchi and Ben Mildenhall. Volume Rendering Digest (for NeRF), 2022. 2

[43] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *Proc. CVPR*, 2022. 2

[44] Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul Srinivasan, Edgar Tretschk, W Yifan, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, et al. Advances in neural rendering. In *Computer Graphics Forum*, 2022. 1

[45] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *Proc. ICCV*, 2021. 2

[46] Vadim Tschernezki, Diane Larlus, and Andrea Vedaldi. NeuralDiff: Segmenting 3D objects that move in egocentric videos. In *Proc. 3DV*, 2021. 8

[47] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. *Proc. CVPR*, 2022. 2

[48] Suhani Vora, Noha Radwan, Klaus Greff, Henning Meyer, Kyle Genova, Mehdi SM Sajjadi, Etienne Pot, Andrea Tagliasacchi, and Daniel Duckworth. Nesf: Neural semantic fields for generalizable semantic segmentation of 3d scenes. *TMLR*, 2021. 2

[49] Chaoyang Wang, Ben Eckart, Simon Lucey, and Orazio Gallo. Neural trajectory fields for dynamic novel view synthesis. *arXiv preprint arXiv:2105.05994*, 2021. 2

[50] Jiepeng Wang, Peng Wang, Xiaoxiao Long, Christian Theobalt, Taku Komura, Lingjie Liu, and Wenping Wang. Neuris: Neural reconstruction of indoor scenes using normal priors. *arXiv preprint*, 2022. 2

[51] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *Proc. NeurIPS*, 2021. 2

[52] Z. Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 2004. 6

[53] Tianhao Wu, Fangcheng Zhong, Forrester Cole, Andrea Tagliasacchi, and Cengiz Oztireli. D2nerf: Self-supervised decoupling of dynamic and static objects from a monocular video. In *Proc. NeurIPS*, 2022. 2, 4, 6, 7, 8, 14, 16

[54] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *Proc. CVPR*, 2021. 2

[55] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *Comput. Graph. Forum*, 2022. 1

[56] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proc. CVPR*, 2021. 2

[57] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. In *Proc. NeurIPS*, 2022. 2

[58] Jason Y Zhang, Deva Ramanan, and Shubham Tulsiani. Relpose: Predicting probabilistic relative rotation for single objects in the wild. In *Proc. ECCV*, 2022. 2

[59] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 2

[60] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. CVPR*, 2018. 6

[61] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R Oswald, and Marc Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. In *Proc. CVPR*, 2022. 1

RobustNeRF: Ignoring Distractors with Robust Losses[4] (Supplementary Material)

6.1. Dataset Description

To investigate RobustNeRF and its baselines, we capture and generate a collection of natural and synthetic scenes. With the goal of reconstructing the *static* elements of a scene, we capture frames both with and without distractors present. We describe the details of the capture below.

	# Clut.	# Clean	# Extra	Paired?	Res.	Setting
Android	122	122	10	No	4032x3024	Apartment
Statue	255	132	19	No	4032x3024	Apartment
Crab	109	109	194	Yes	3456x3456	Robotics Lab
BabyYoda	109	109	202	Yes	3456x3456	Robotics Lab

Figure 12. **Natural Scenes** – Key facts about natural scenes introduced in this work. Includes number of paired photos with (# Clut.) and without (# Clean) distractors. Extra photos (# Extra) do not contain distractors and are taken from unpaired camera poses.

6.1.1 Natural Scenes

We introduce four natural scenes, two captured in an apartment setting, and two in a robotics lab. See Figure 12 for key details.

Apartment (Statue & Android). To mimic a casual home scenario, we capture two tabletop scenes in an apartment using a commodity smartphone. Both captures focus on one or more objects on a table top, with photos taken from different viewpoints from a hemisphere of directions around the objects of interest. A subset of objects on the table move from photo to photo as described below. The photos within each scene do not have a clear temporal order.

The capture setup is as follows. We employ an iPhone 12 mini and use ProCamera v15 to control camera exposure settings. We use a fixed shutter speed of 1/60, 0.0 exposure bias, and a fixed ISO of 80 or 200 for the Statue and Android scenes, respectively. We use the iPhone's standard wide lens with an aperture of f/1.6 and resolution of 4032x3024. A tripod is used to reduce the effects of the rolling shutter.

The Android dataset comprises 122 cluttered photos and 10 clean photos (i.e., with no distractors). This scene depicts two Android robot figures standing on a board game box, which in turn is sitting on a table with a patterned table cloth. We pose three small wooden robots atop the table in various ways in each cluttered photo to serve as distractors.

For the Statue scene, we capture 255 cluttered photos and 19 clean photos. The scene depicts a small statue on top of a highly-detailed decorative box on a wooden kitchen table. To simulate a somewhat persistent distractor, we float a balloon over the table which, throughout the capture, naturally changes its position slightly with each photo. Unlike the Android scene, where distractors move to entirely new poses in each frame, the balloon frequently inhabits the same volume of space for multiple photos. The decorative box and kitchen table both exhibit fine grained texture details.

We run COLMAP's [37] Structure-from-Motion pipeline using the `SIMPLE_RADIAL` camera model. While COLMAP's camera parameter estimates are only approximate, we find that they are sufficient for training NeRF models with remarkable detail.

The apartment scenes are considerably more challenging to reconstruct than the robotics lab scenes (described below). An accurate NeRF reconstruction must model not only the static, foreground content but also the scene's background. Unlike the foreground, each object in the background is partially over- or underexposed and appears in a limited number of photos. We further found it challenging to maintain a controlled, static scene during capture. As a result, some objects in the background move by a small, unintended amount between photos (e.g., see Figure 14).

Robotics Lab (Crab & BabyYoda). In an effort to control confounding factors in data acquisition, we capture two scenes in a Robotics Lab setting. In these scenes, we employ a robotic arm to randomly position a camera within 1/4 of the hemisphere over a table. The table is placed in a closed booth with constant, indoor lighting. A series of toys are placed on the table, a subset of which are glued to the table's surface to prevent them from moving. Between photos, distractor toys on the table are removed and/or new distractor toys are introduced.

For capture, we use a Blackfly S GigE camera with a TECHSPEC 8.5mm C Series fixed length lens. Photos are center-cropped from their original resolution of 5472x3648 to 3456x3456 to eliminate lens distortion. We capture 12-bit raw photos with an aperture of f/8 and exposure time of 650 ms. Raw photos are automatically color-calibrated afterwards according to a reference color palette.

In each scene, we capture 109 pairs of photos from identical camera poses, one with distractors present and another without. This results in a large number of unique distractors which are challenging to model directly. This further allows us to investigate the counterfactual: What if distractors were *not* present? We further capture an additional ∼200 photos from random viewpoints, not aligned with those for training and without distractors, for the purposes of evaluation. In total, because the placement of objects is done manually, one capture session often takes several hours.

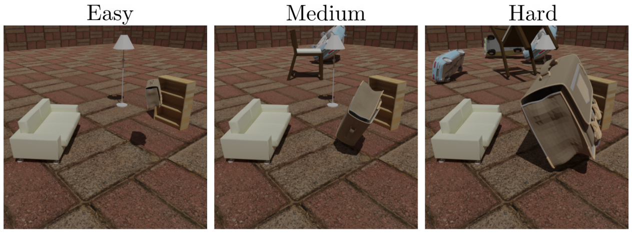

Figure 13. **Synthetic Kubric Scenes** – Example Kubric synthetic images for three datasets with different ratio of outlier pixels. The sofa, lamp, and bookcase are static objects in all three setups. The easy setup has 1 small distractor, the medium setup has 3 medium distractors, and the hard setup has 6 large distractors.

6.1.2 Synthetic Scenes

We generate three Kubric [12] scenes similar to the D^{2}NeRF synthetic scenes with different difficulty levels: easy, medium, and hard. These datasets are used to ablate our method with control on the proportion of outlier occupancy (see Sec. 6.3.3).

Each dataset contains 200 cluttered images for training and 100 clean images for evaluation. In all three scenes the static objects include a sofa, a lamp and a bookshelf. Figure 13 shows one example image from the training set for each dataset. The easy scene contains only one small distractor object (a bag). This dataset is similar to Kubric Bag dataset of D^{2}NeRF. The medium scene has three distractors (a bag, a chair, and a car) which are larger in size and hence the outlier occupancy is 4× the outlier occupancy of the easy scene. The hard scene has six large distractors (a bag, a chair, and four cars). They occupy on average 10× more pixels than the easy setup, covering roughly half of each image.

6.2. Training Details

While camera parameters are estimated on the full-resolution imagery, we downsample images by 8x for each natural scene dataset. While mip-NeRF 360 and RobustNeRF are capable of training on high resolution photos, we limit the resolution to accommodate D^{2}NeRF. Unless otherwise stated, we train on all available cluttered images, and evaluate on a holdout set; i.e., 10 images for Android; 19 for the Statue dataset; 194 for Crab; and 202 for the BabyYoda dataset (see Figure 12).

RobustNeRF. We implement RobustNeRF by incorporating our proposed loss function into the MultiNeRF codebase [26], replacing mip-NeRF 360's [4] reconstruction loss. All other terms in the loss function, such as regularizers, are included as originally published in mip-NeRF 360.

We train RobustNeRF for 250,000 steps with the Adam optimizer, using a batch size of 64 image patches randomly sampled from training images. Each pixel within a 16x16 patch contributes to the loss function, except those identified as outliers (see Figure 6 for a visualization). The learning rate is exponentially decayed from 0.002 to 0.00002 over the course of training with a warmup period of 512 steps.

Our model architecture comprises a proposal Multilayer Perceptron (MLP) with 4 hidden layers and 256 units per layer, and a NeRF MLP with 8 hidden layers, each with 1024 units. We assign each training image a 4-dimensional GLO vector to account for unintended appearance variation. Unless otherwise stated, we use the robust loss hyperparameters given in the main body of the paper. All models are trained on 16 TPUv3 chips over the course of 9 hours.

mip-NeRF 360 [4]. We use the reference implementation of mip-NeRF 360 from the MultiNeRF codebase. Similar to RobustNeRF, we train each variant of mip-NeRF 360 with the Adam optimizer, using the same number of steps, batch size, and learning rate schedule. mip-NeRF 360 uses a random sample of 16384 rays per minibatch. Proposal and NeRF MLP depth and width are identical to those for RobustNeRF. Training hardware and duration are also the same as RobustNeRF.

D^{2}NeRF [53]. We use the reference implementation of D^{2}NeRF [53] provided by the authors. Model architecture, hierarchical volume sampling density, and learning rate are the same as published in [53]. As in the original work, we train the model for 100,000 iterations with a batch size of 1024 rays, though over the course of 3 hours. Due to hardware availability, we employ four NVIDIA V100 GPUs in place of the A100 GPUs used in the original work.

Images are kept in the order of provided by the file system (i.e., ordered by position information alphanumerically). However, this image order is not guaranteed to represent a continuous path in space since the images were not captured along a continuous path, but rather at random locations. Below we discuss the effects of random ordering versus ordering the views along a heuristically identified path.

D^{2}NeRF training is controlled by five key hyperparameters, namely, skewness (k), which encourages a binarization loss to favor static explanations, and four regularization weights that scale the skewed binarization loss (λ_s), ray regularization loss (λ_r), static regularization loss (λ_{σ^s}), and the view-correlated shadow field loss (λ_ρ). A hyperparameter search is performed in D^{2}NeRF for 16 real world scenes to identify combinations best suited for each scene, and four primary configurations of these parameters are identified as optimal. In particular, the first configuration (i.e., $k = 1.75$, $\lambda_s = 1e^{-4} \rightarrow 1e^{-2}$, $\lambda_r = 1e^{-3}$, $\lambda_{\sigma^s} = 0$, and $\lambda_\rho = 1e^{-1}$) was reported to be most effective across the largest number of scenes real world (10 of 16). We additionally conduct a tuning experiment (see Figure 16) and confirm the first configuration as best suited. We apply this configuration in all additional D^{2}NeRF experiments.

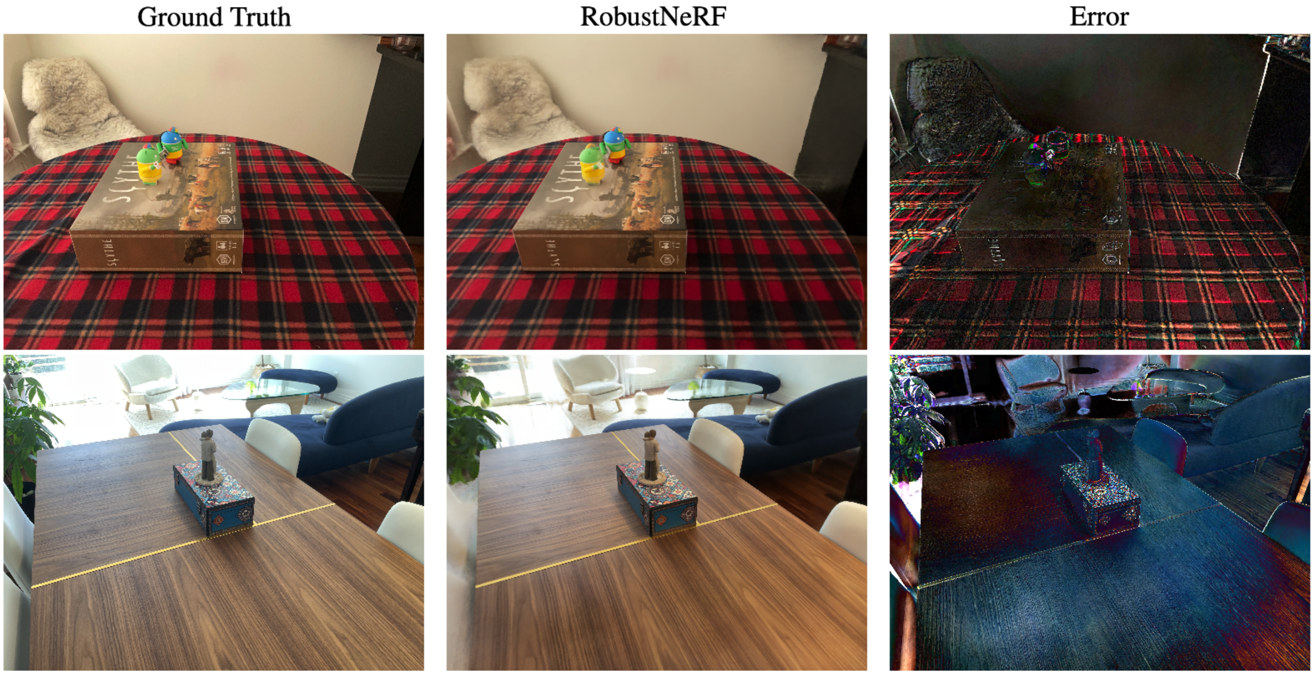

Figure 14. **Challenges in Apartment Scenes –** Each row, from left to right, shows a ground truth photo, a RobustNeRF render, and the difference between the two. Best viewed in PDF. (Top) Note the fold in the table cloth in ground truth image and the lack of fine-grained detail on the covered chair in the background. The table cloth moved during capture, and the background was not captured thoroughly enough for a high-fidelity reconstruction. (Bottom) The ground truth image for the Statue dataset exhibits overexposure and color calibration issues, and hence do not exactly match the RobustNeRF render.

6.3. Experiments

6.3.1 Comparison to mip-NeRF 360

In experiments on natural scenes, as reported in Figure 8, the performance gap between mip-NeRF 360 (Ch.) and RobustNeRF is markedly higher for the two scenes captured in the robotics lab (i.e., Crab, BabyYoda), compared to those in the apartment (i.e., Statue, Android). We attribute this to the difficulty in reconstructing the apartment scenes, regardless of the presence of distractors. This statement is supported by metrics for reconstruction quality of a mip-NeRF 360 model trained on clean, distractor-free photos. In particular, while mip-NeRF 360 achieves over 32 dB PSNR on Crab and BabyYoda scenes, its PSNR is nearly 10 dB lower on Statue and Android.

Upon closer inspection of the photos and our reconstructions, we identified several reasons for this. First, the apartment scenes contain non-trivial background content with 3D structure. As the background was not the focus of these captures, background content is poorly reconstructed by all models considered. Second, background content illuminated by sunlight is overexposed in some test images (see Figure 14). While this challenge has already been addressed by RawNeRF [24], we do not address it here as it is not a focus of this work. Lastly, we find that some static objects were unintentionally moved during our capture. The most challenging form of this is the movement of a table cloth prominently featured in the Android scene which lead to perturbed camera parameter estimates (e.g., see Figure 14).

	Crab			BabyYoda		
	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑
Order 1	0.43	0.66	20.19	0.44	0.66	18.17
Order 2	0.42	0.68	20.95	0.44	0.66	17.13

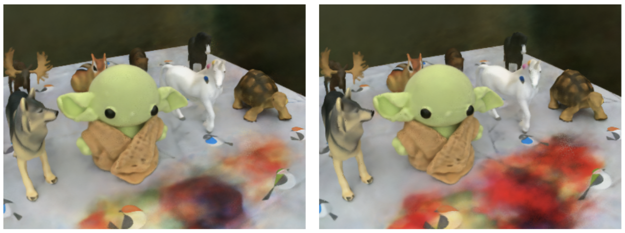

Figure 15. **Effect of Image Order on D^{2}NeRF** – As this model is based on space-time NeRFs [30], to make it compatible with our setting we create a 'temporal' indexing of the photos. Here, we visualize: (left) with our heuristic ordering; (right) with another random order. We observe similar distractor-related artifacts in both cases.

6.3.2 Comparison to D^{2}NeRF

Unlike RobustNeRF, D^{2}NeRF makes use of a time signal in the form of provided appearance and warp IDs to gen-

	Statue			Crab		
	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑
Config 1	0.48	0.49	19.09	0.42	0.68	21.18
Config 2	0.49	0.48	18.20	0.51	0.59	17.02
Config 3	0.51	0.47	18.28	0.46	0.63	19.01
Config 4	0.49	0.48	18.18	0.49	0.58	16.77

Config #	k	λ_s	λ_r	λ_{σ^s}	λ_ρ
Config 1	1.75	1e-4 → 1e-2	1e-3	0	1e-1
Config 2	3	1e-4 ⇒ 1	1e-3	0	1e-1
Config 3	2.75	1e-5 ⇒ 1	1e-3	0	-
Config 4	2.875	5e-4 ⇒ 1	0	0	-

Figure 16. **D^{2}NeRF HParam Tuning** – The performance of D^{2}NeRF is heavily influenced by the choice of hyperparameters. In particular, optimal choices of hyperparameters are noted to be strongly influenced by the amount of object and camera motion, as well as video length. We tune by applying four recommended configurations, and identify the first as optimal across the Statue and Crab datasets. Please note that → indicates linear increase in value and ⇒ indicates exponential increase in value.

erate a code as additional input to the HyperNeRF model. This explicitly models dynamic content alongside the static component of the scene. Two assumptions of D^{2}NeRF are broken in our datasets: 1) the objects sporadically appear (by design); and 2) the views are not necessarily captured in a video-like order. Sporadic object appearance is central to our task, so we do not ablate this property. However, we do evaluate the effect of heuristically reordering camera views according to z position and radial angle of the robotic arm, thereby producing an image order for an imagined ”continuous” path. As a control, we pseudorandomly scramble the view order, and train D^{2}NeRF in both settings. The results for BabyYoda and Crab can be seen in Figure 15. We observe no consistent discernable improvement in performance as a result of view reordering and hypothesize that the major hurdle for D^{2}NeRF is rather the modeling of sporadic artifacts.

We also evaluate the effect of applying the four hyperparameter configurations provided by D^{2}NeRF [53]. We observe, as expected, that the first configuration performs best across our datasets. Due to limited access to appropriate compute architecture for D^{2}NeRF, we were not able to tune every scene, but selected configuration 1 for all experiments as it performed best in 10/16 real world scenes for D^{2}NeRF as well as tuning experiments on two of our example datasets as see in in Figure 16.

6.3.3 Sensitivity to Hyperparameters

We find that the choice of thresholds and filter sizes, described in Section 3, suffices for a wide range of datasets. As long as the threshold $\mathcal{T}_\epsilon$ is greater than the proportion of outlier pixels in a dataset, RobustNeRF will reliably identify and ignore outlier pixels; see Figure 17. Easy has less than 10% outlier pixels so any $\mathcal{T}_\epsilon$ less than 80% works. In the medium case at least a $\mathcal{T}_\epsilon$ of 60% is required to remove the outliers. In the hard case 44% of pixels are on average occupied so any $\mathcal{T}_\epsilon$ above 50% has worse results. Training with less than 50% of the loss slows down training significantly. Therefore, we observe that after the 250k iterations the model has not converged yet. On average training with 30% of loss requires twice the number of training iterations to catch up. In contrast, D^{2}NeRF requires careful, manual hyperparameter tuning for each scene (e.g., see Figure 16) for several hyperparameters.

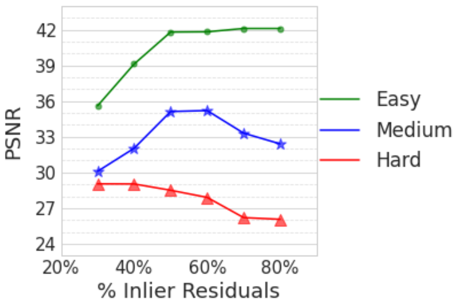

Figure 17. **Sensitivity to $\mathcal{T}_\epsilon$ –** RobustNeRF's reconstruction quality as a function of $\mathcal{T}_\epsilon$ on scenes with different inlier/outlier proportions. Overestimating $\mathcal{T}_\epsilon$ increases training time without affecting final reconstruction accuracy.

6.3.4 More Qualitative Results

We render images from different NerF models from more viewpoints from each of our datasets to further expand the comparison with baselines, D^{2}NeRF, and RobustNeRF. Looking at Figures 18 through 21 one can see that D^{2}NeRF is only able to remove the outliers when there is a single distractor object (Statue dataset) and it fails on the other three datasets. The Android dataset has three wooden robots with articulated joints as distractors, and even in this setup where the texture of the distractor objects are similar to one another, D^{2}NeRF fails to fully remove the outliers. In comparison, RobustNeRF is able to remove the outliers irrespective of their number and diversity.

For all four datasets mip-NeRF 360, with either L1, L2, or Charbonnier loss, fails to detect the outliers; one can see ’clouds’ or even distinct floaters for these methods. The worst performing loss is L2, as expected. L1 and Charbonnier behave similarly in terms of outlier removal. Changing the loss to RobustNeRF eliminates the floaters and artifacts in all datasets. Video renderings for these scenes are also included in the zipfile with the supplementary material. The floaters in mip-NeRF 360 are easier to resolve in the videos.

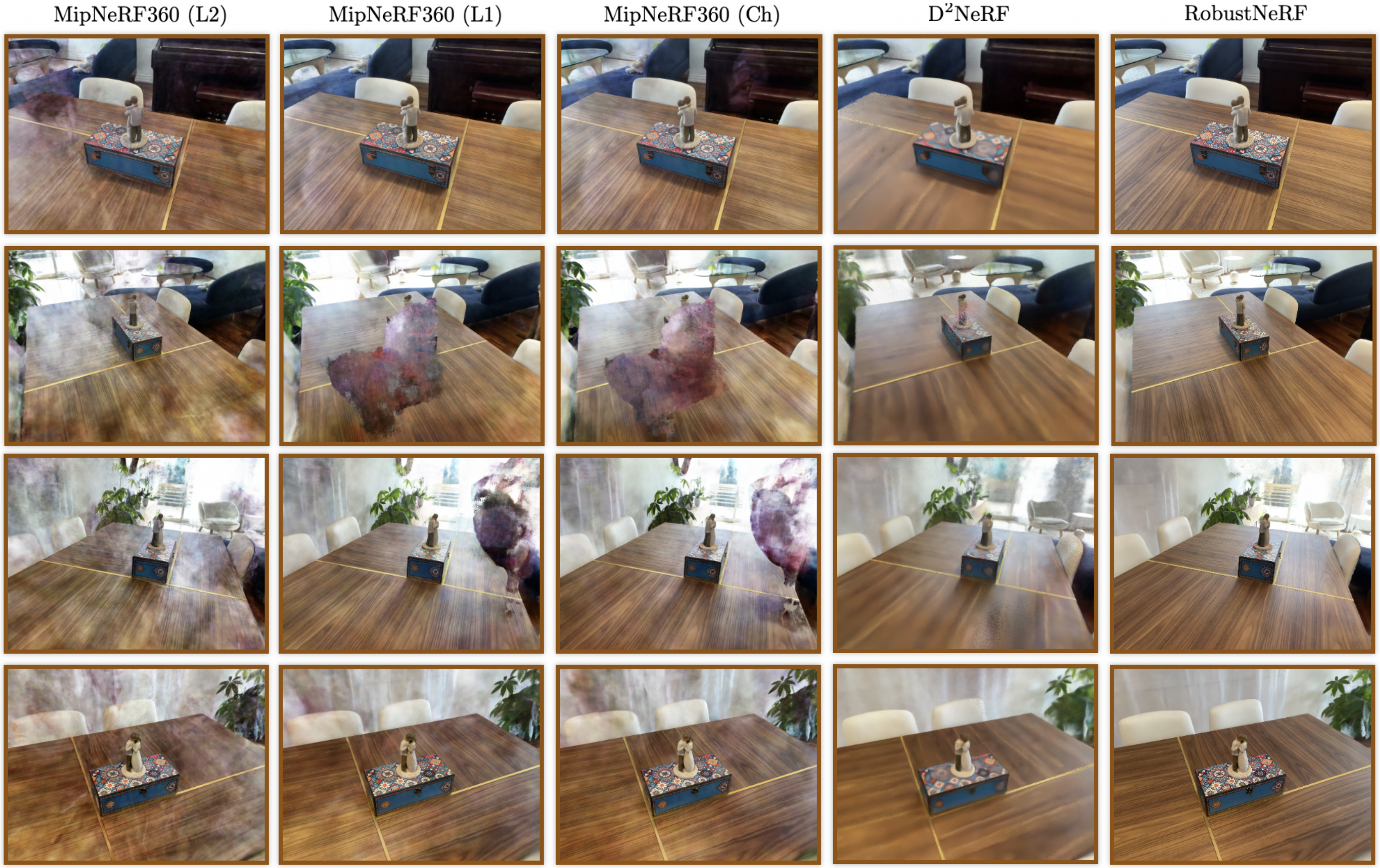

Figure 18. **Statue –** Qualitative results on Statue.

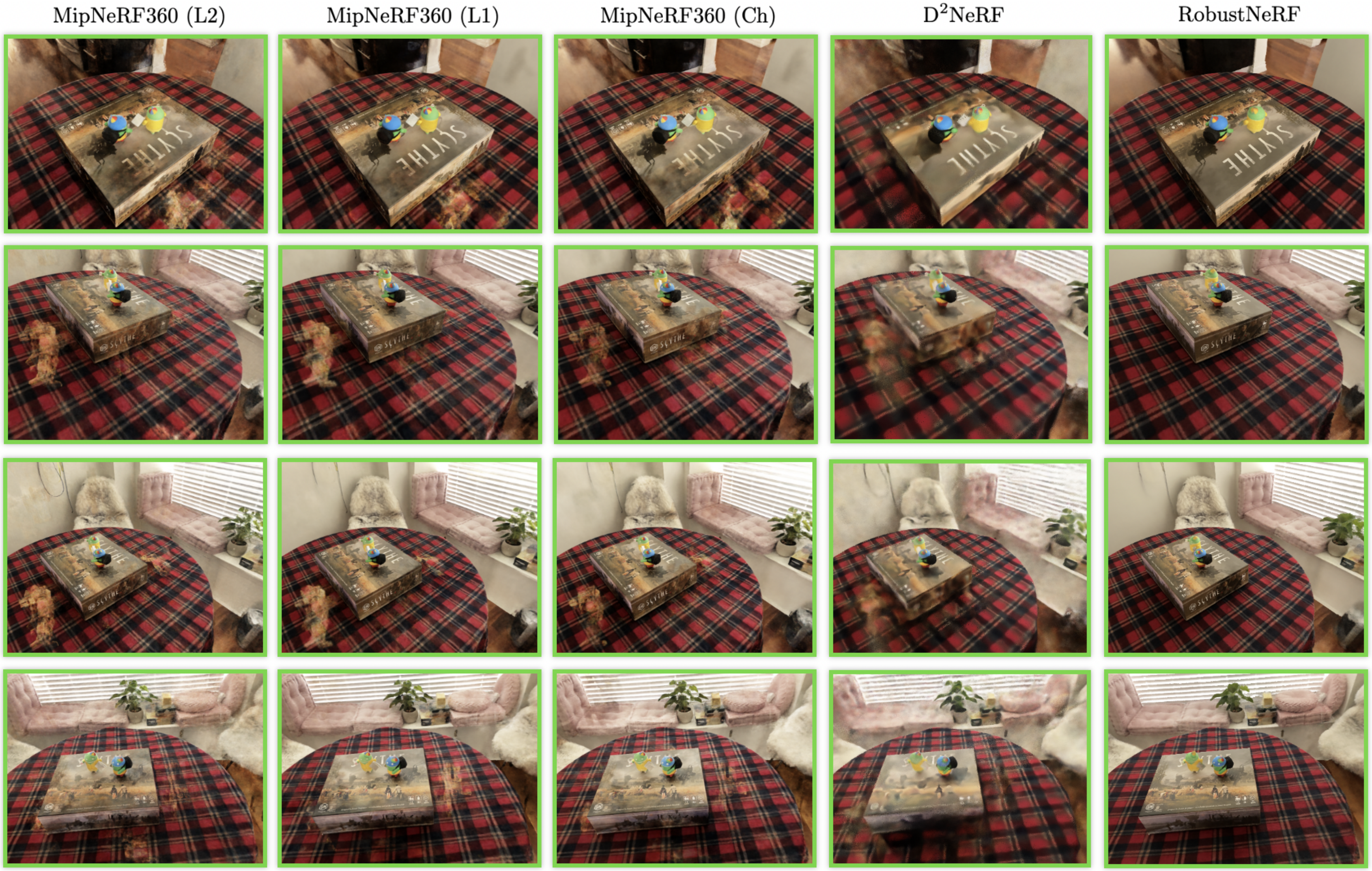

Figure 19. **Android –** Qualitative results on Android.

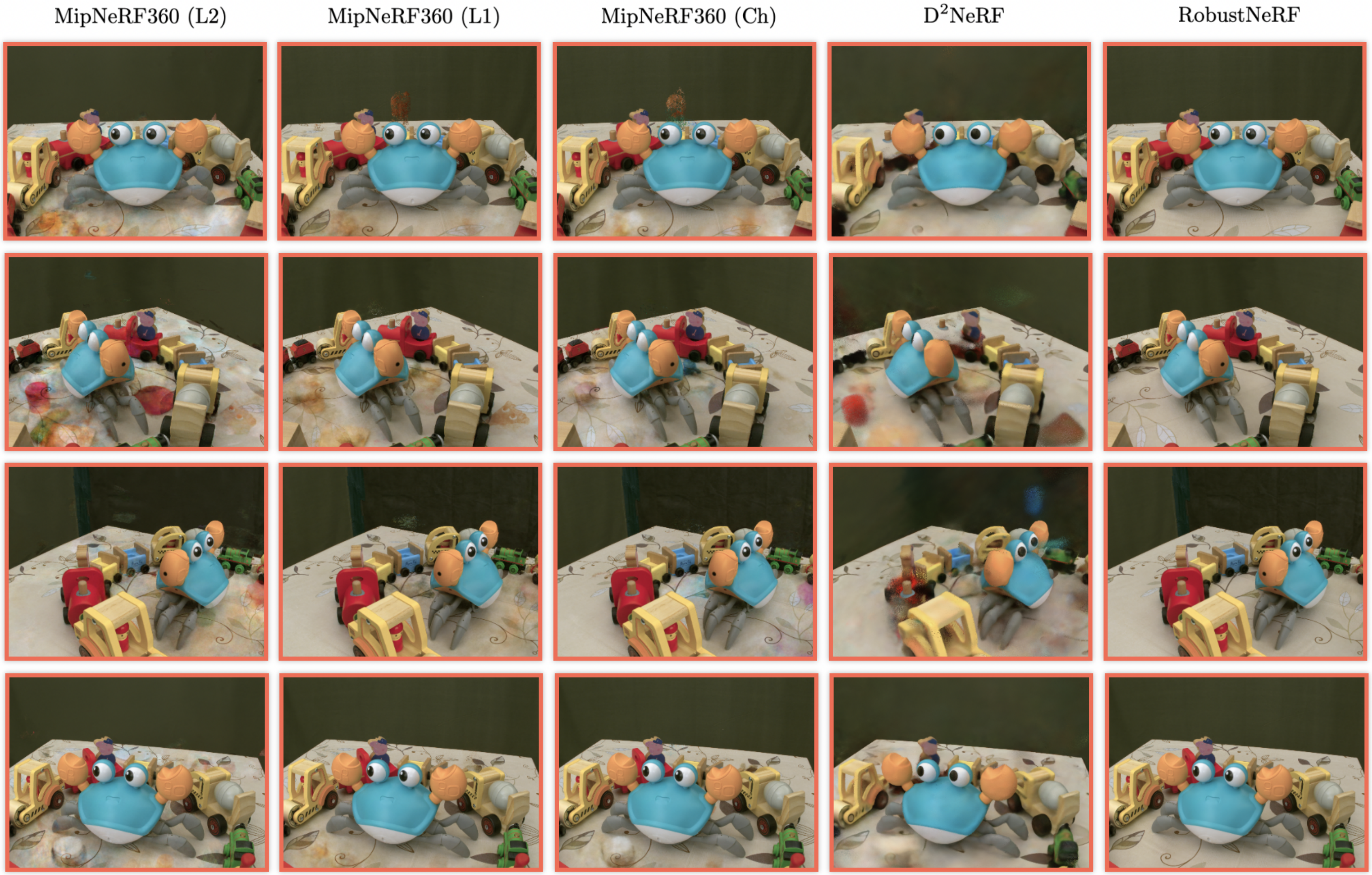

Figure 20. **Crab –** Qualitative results on Crab.

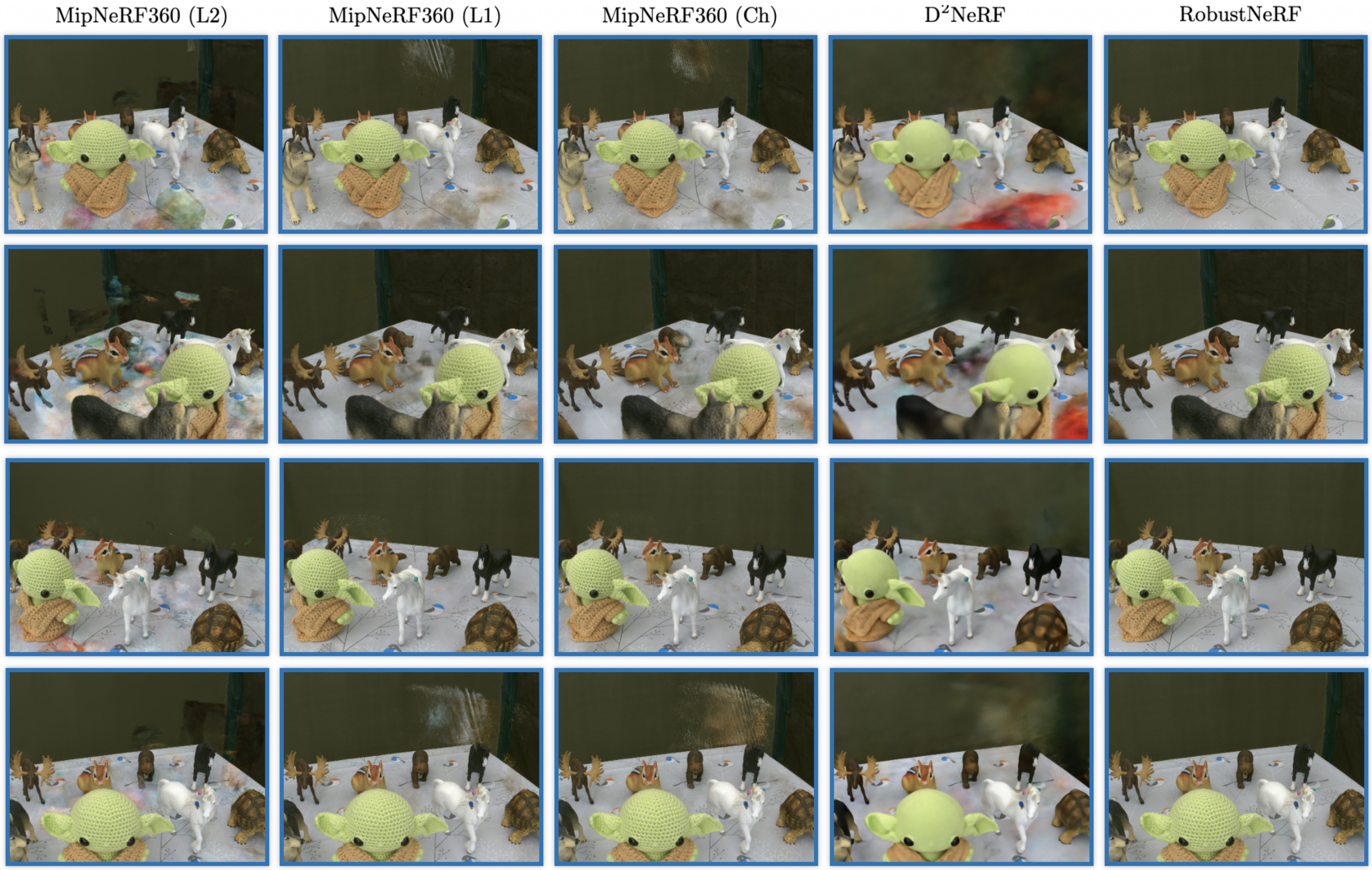

Figure 21. **BabyYoda –** Qualitative results on BabyYoda.