Non-Rigid Neural Radiance Fields: Reconstruction and Novel View Synthesis of a Dynamic Scene From Monocular Video
—Supplemental Material—

Edgar Tretschk
MPI for Informatics, SIC

Ayush Tewari
MPI for Informatics, SIC

Vladislav Golyanik
MPI for Informatics, SIC

Michael Zollhöfer
Facebook Reality Labs Research

Christoph Lassner
Facebook Reality Labs Research

Christian Theobalt
MPI for Informatics, SIC

Overview

We provide further illustrations of the loss function in Sec. S.1. We discuss a number of training details in Sec. S.2. Next, we provide some implementation details in Sec. S.3. In Sec. S.4, we describe the capture of our data. Sec. S.5 contains details on how we visualize the additional output modalities of our method and some more results. We show an additional result of our ablation study in Sec. S.6. We provide more details on the experimental settings of our comparisons and some additional results in Sec. S.7. Then in Sec. S.8, we present the extensions to multi-view data and view-dependent effects mentioned in the main paper. Sec. S.9 contains the scene editing tasks mentioned in the main paper. Sec. S.10 contains a limitation example of our method. Finally, Sec. S.11 contains some preliminary qualitative comparisons to a concurrent, non-peer-reviewed work.

S.1. Loss Illustration

Fig. S.1 illustrates $L_{divergence}$ in 2D.

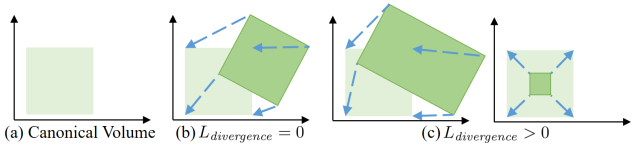


Figure S.1. $L_{divergence}$ encourages the offsets field to (b) preserve local volume rather than (c) losing it while deforming.

S.2. Training Details

While the network weights are optimized as usual, the latent codes $\mathbf{l}_i$ are auto-decoded, i.e., they are treated as free variables that are directly optimized for, similar to network weights, instead of being regressed. This is based on the auto-decoding framework used in DeepSDF and earlier works [10, 6].

We initialize $\{\mathbf{l}_i\}_i$ to zero vectors. For implementing the radiance field, we use the same architecture as in NeRF [5]. The ray bending network is a 5-layer MLP with 64 hidden dimensions and ReLU activations, the last layer of which is initialized with all weights set to zero. The rigidity network is a 3-layer MLP with 32 hidden dimensions and ReLU activations, with the last layer initialized to zeros. The output of the last layer of the rigidity network is passed though a $\tanh$ activation function and then shifted and rescaled to lie in $[0, 1]$. We train usually for 200k iterations with a batch of 1k randomly sampled rays. At training and at test time, we use 64 coarse and 64 fine samples per ray in most cases. We use ADAM [2] and exponentially decay the learning rate to 10% from the initial $5 \cdot 10^{-4}$ over 250k iterations. For dark scenes, we found it necessary to introduce a warm-up phase that linearly increases the learning rate starting from $\frac{1}{20}$th of its original value over 1000 iterations. The latent codes are of the dimension 32. We train between six and seven hours on a single Quadro RTX 8000.

Since we consider a variety of types of non-rigid objects/scenes and deformations, we find it necessary to use scene-specific weights for each loss term. We have found the following ranges to be sufficient for a wide range of scenarios: ω_{rigidity} lies in $[0.01, 0.001]$ and typically is 0.003, ω_{offsets} lies in $[60, 600]$ and typically is 600, and $\omega_{\text{divergence}}$ lies in $[1, 30]$ and typically is 3 or 10. In our experience, NR-NeRF is fairly insensitive to ω_{offsets}. Rather rigid objects benefit from higher $\omega_{\text{divergence}}$, while fairly non-rigid objects need lower $\omega_{\text{divergence}}$. Finally, we increase ω_{rigidity} whenever we find the background to be unstable. We start the training with each weight set to $\frac{1}{100}$th of its value, and then exponentially increase it until it reaches its full value at the end of training.

S.3. Implementation Details

Our code is based on a faithful PyTorch [7] port [11] of the official Tensorflow NeRF code [5]. We use the official FFJORD implementation [1] to estimate Eq. 5 from the main paper. If the camera extrinsics and intrinsics are not given, we estimate them using the Structure-from-Motion (SfM) implementation of COLMAP [8, 9]. We find COLMAP to be quite robust to non-rigid 'outliers'. As we are interested in estimating smooth deformations, we only apply positional encoding to the input of the canonical NeRF volume, not to the input of the ray bending network. We will make our source code available.

S.4. Data

We show results on a variety of scenes recorded with three different cameras: the Kinect Azure, a Blackmagic, and a phone camera. Since the RGB camera of the Kinect Azure exhibits strong radial distortions along the image border, we use the manufacturer-provided intrinsics and distortion parameters to undistort the recorded RGB images beforehand. We extract frames at 5 fps from the recordings, such that scenes usually consist of 80 to 300 images, at resolutions of 480×270 (Blackmagic, and Sony XZ2) or 512×384 (Kinect Azure).

S.5. Output Modalities

S.5.1. Visualizations

Rigidity Scores In order to visualize the estimated rigidity, we need to determine the rigidity of the ray associated with a pixel. We choose to define the rigidity of such a ray as the rigidity of the point j closest to an accumulated weight $\sum_{k=0}^{j-1} \alpha_k$ of 0.5, *i.e.,* closest to the median. In practice, this usually gives us the rigidity at the first visible surface along the ray.

Correspondences To visualize correspondences, we treat the canonical volume as an RGB cube, i.e., we treat the xyz coordinate in canonical space as an RGB color. Since this would result in very smooth colors, we split the canonical volume into a voxel grid of 100^3 RGB cubes beforehand. We pick the ray point that determines the pixel color similar to the rigidity visualization.

S.5.2. More Results

See Fig. S.2 for more results of the output modalities of NR-NeRF.

Canonical Volume Since the canonical volume is not supervised directly, it is conceivable that it could have baked-in deformations. The last row of Fig. S.2 contains renderings of the canonical volume without any ray bending applied. The canonical volume is a plausible state of the scene and does not show baked-in deformations. We thus find it to be sufficient to bias the optimization towards a desirable canonical volume by initializing the ray bending network to an identity map and by our regularization losses.

S.6. Ablation Study

Fig. S.3 contains an additional ablation study result that demonstrates the importance of the divergence regularization for foreground stability.

S.7. Comparisons

In this section, we provide more details on the experimental settings of the comparisons.

S.7.1. Prior Work and Baseline

We start with the trivial baseline of rigid NeRF [5], which cannot handle dynamic scenes. We consider two variants: view-dependent rigid NeRF, as in the original method [5], and view-independent rigid NeRF, where we remove the view-direction conditioning.

We next introduce *naïve NR-NeRF*, which adds naïve support for dynamic scenes to rigid NeRF: We condition the neural radiance fields volume on the latent code. Thus, for latent code $\mathbf{l}_i$, we have $(\mathbf{c}, o) = \mathbf{v}(\mathbf{x}, \mathbf{l}_i)$. This allows the neural radiance fields volume to output time-varying color and occupancy. Unlike NR-NeRF's ray bending, naïve NR-NeRF does not have an explicit, separate deformation model. Instead, the volume needs to account for appearance, geometry and deformation at once. Note that for test images i, we do backpropagate gradients into the corresponding latent code $\mathbf{l}_i$.

Finally, we compare to Neural Volumes [4], for which we use the official code release. We use the standard settings as a starting point, but set the number of training iterations to $100k$, which leads to a training time of about two days on an RTX8000 GPU. Neural Volumes uses an image encoder to regress a latent code that conditions the geometry, appearance and deformation regression on the current time step. Since this design assumes a multi-view setup, we need to adapt it to our monocular setting. Instead of picking three fixed camera views that are always input into the encoder, we input the single image of the current time step. In particular, at test time, we input the test image. Since we do not have access to a background image, we set the estimated background image to an all-black image. We furthermore consider two variations: (1) following the original Neural Volumes method, the geometry and appearance template is conditioned on the latent code (*NV*), and (2) the geometry and appearance template is independent of the latent code (*modified NV*). In the latter case, the latent code only conditions the warp field, which is similar to our method.

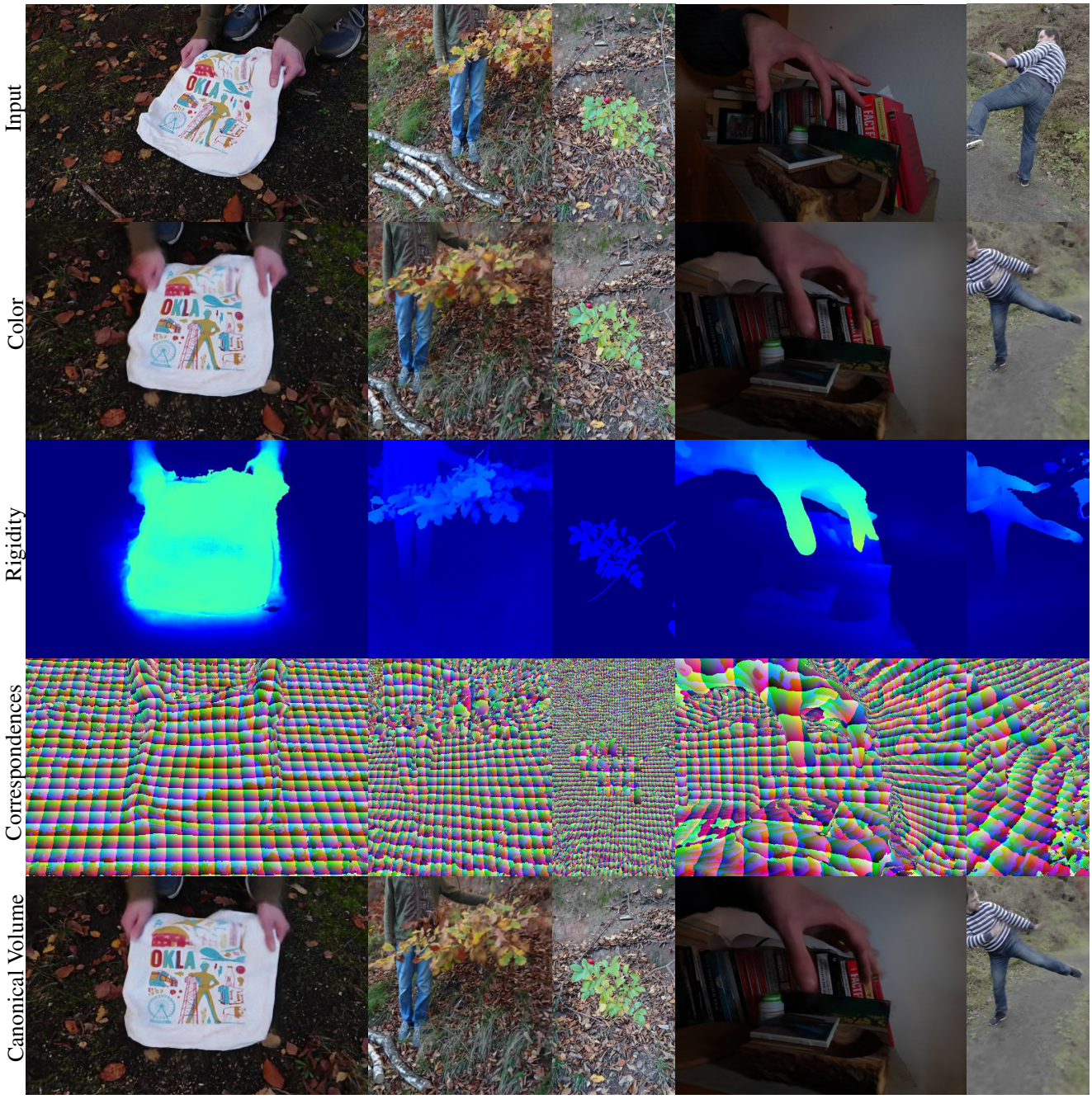
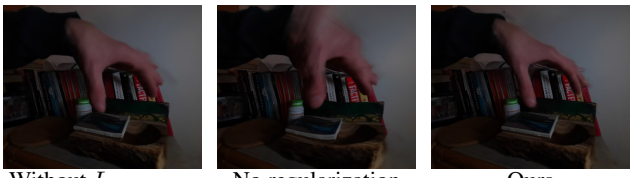


Figure S.2. NR-NeRF can render a deformed state captured at a certain time step into a novel view. We visualize here this novel-view rendering and additional output modalities as seen from the novel view, namely rigidity scores, correspondences, and the canonical volume. The canonical volume is a plausible state of the scene and does not show baked-in deformations. We refer to the supplemental video for video results.

S.7.2. Training/Test Split

For quantitative evaluation, we require a test set. In the comparison section, we split the images into training and test images by partitioning the temporally-ordered images into consecutive blocks, each of length 16. The first twelve images of each block are used as training data, while the remaining four are used for testing. In our setting, test images still require corresponding latent codes to represent the



Without $L_{divergence}$ | No regularization | Ours

Figure S.3. Ablation Study. We render the input scene into a novel view to determine the stability of the non-rigid objects. We show the results of removing the divergence loss, all three regularization losses, and none of the losses.

deformations. Therefore, we treat test images like training images except that we do not backpropagate into the canonical volume or the ray bending network. However, we do use gradients from test images to optimize the corresponding latent codes. (Note that test images *solely* influence test latent codes, as is typical for auto-decoding [6]. [1]) All other results shown in this paper, outside of the comparisons, treat all images as training images since we train scene-specific networks. Furthermore, qualitative baseline results in the supplemental video show both training and test time steps.

S.7.3. Additional Results

See Fig. S.4 for more qualitative results and Fig. S.7 for quantitative results on background stability under novel views.

S.8. Extensions

As mentioned in Sec. 4 from the main paper, we can extend our approach easily to work with multi-view data and view-dependent effects.

S.8.1. Multi-View Data

Our approach naturally handles multi-view data. Although we mainly work with monocular data, we can use multi-view data to investigate the upper quality bound of our approach under ideal real-world conditions.

Method Instead of each image having its own time step and hence latent code, images taken at the same time step share the same latent code. This ensures that the canonical volume deforms consistently within each time step.

Data and Settings We use a multi-view dataset that has 16 camera pairs evenly distributed around the scene, which sufficiently constrains the optimization such that we find the training to not need any regularization losses. We train at the original resolution of 5120×3840 for 2 million training iterations with 4096 rays per batch and 256 coarse and 128 fine samples. These highest-quality settings lead to a training time of 11 days on 4 RTX8000 GPUs, and a rendering time of about 10 minutes per frame on the same hardware.

Results See Fig. S.5 and the supplemental video for results on five consecutive time steps.

S.8.2. View Dependence

We can optionally add view-dependent effects, like specularities, into our model.

Method Determining the view direction or ray direction is not as trivial as for the straight rays. Instead, we need to calculate the direction in which the bent ray passes through a point in the canonical volume. We consider two options of doing so: exact and slower, or approximate and faster.

Exact: We obtain the direction of the bent ray $\tilde{\mathbf{r}}$ at a point $\tilde{\mathbf{r}}(j)$ via the chain rule as $\nabla_j \tilde{\mathbf{r}}(j) = \frac{\partial \tilde{\mathbf{r}}(j)}{\partial \mathbf{r}(j)} \cdot \frac{\partial \mathbf{r}(j)}{\partial j} = J \cdot \mathbf{d}$, where J is the 3×3 Jacobian and $\mathbf{d}$ is the direction of the straight ray. We compute J via three backward passes (one for each output dimension), which is computationally expensive.

Approximate: To reduce computation, we can approximate the direction at the ray sample via finite differences as the normalized difference vector between the current point $\tilde{\mathbf{r}}(j)$ and the previous point $\tilde{\mathbf{r}}(j-1)$ along the bent ray (which is closer to the camera).

Results On multi-view data, conditioning on the viewing direction reduces the presence of subtle, smoke-like artifacts, which the canonical volume typically employs to model view-dependent effects without view conditioning. This is especially visible for the specularities on the face and the handle of the kettlebell. Without view-dependent effects, the reconstructed face still appears to exhibit specularities, but these are *incorrectly* modeled via smoke-like artifacts in the surrounding air. See Fig. S.5 and the supplemental video for results.

For quantitative results on monocular sequences, see Tab. S.8.2. However, as Fig. S.6 shows, we find our formulation to lead to artifacts in some cases. We hypothesize that the combination of both significant motion and novel views significantly different from input views is too underconstrained for view-dependent effects. For example, non-rigid NeRF might incorrectly overfit to subtle correlations between deformation and camera position at training time. However, we want to emphasize that better formulations and regularization in future work may make view-dependent effects work in these challenging scenarios.

[1] The only tweak we add is that we align the optimization landscapes of the training and test latent codes by optimizing the test latent codes jointly with the training latent codes during training. This does not lead to any information leakage from the test images to any component except for the test latent codes.

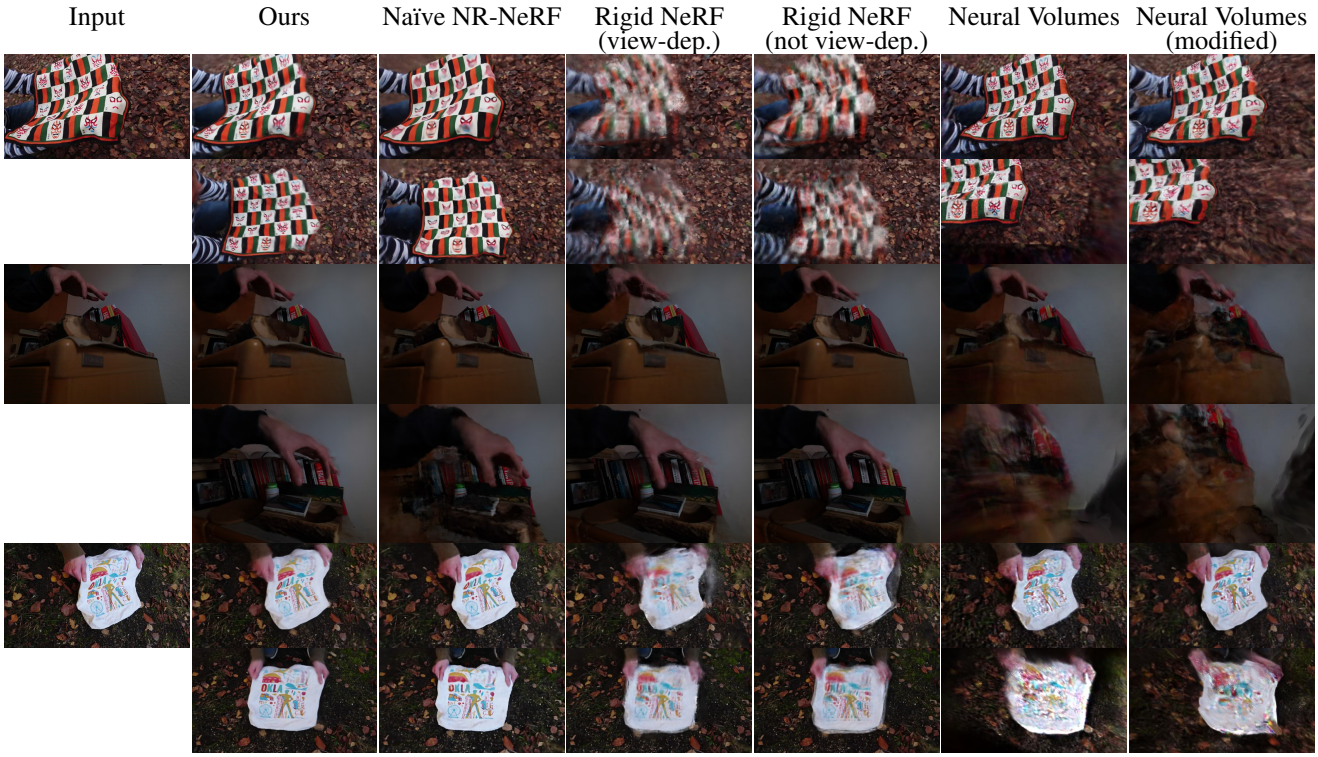


Figure S.4. We show one time step each from each sequence and compare input reconstruction quality (first row) and novel view synthesis quality (second row).

	Ours	Ours (appx.)	Ours (exact)	Naïve	Rigid (cond.)	Rigid (no cond.)	NV	NV (mod.)
PSNR	24.70	*25.15*	25.07	**25.83**	22.24	21.88	14.13	14.10
SSIM	0.758	**0.766**	*0.765*	0.738	0.662	0.659	0.259	0.263
LPIPS	0.197	*0.191*	**0.190**	0.226	0.309	0.313	0.580	0.583

Table S.1. Quantitative Results Averaged Across Scenes. We evaluate our method (1) without view conditioning, (2) with approximate view conditioning, and (3) with exact view conditioning, naïve NR-NeRF, rigid NeRF [5] (1) with view conditioning and (2) without view conditioning, and Neural Volumes [4] (1) without and (2) with modifications. For PSNR and SSIM [13], higher is better. For LPIPS [12], lower is better. As in Table 1 in the main document, we use 18 scenes here, with an average length of 146 frames and a minimum of 41 and a maximum of 453 frames.

S.9. Simple Scene Editing

We can manipulate the learned model in further simple ways: foreground removal, temporal super-sampling, deformation exaggeration or dampening, and forced background stabilization. Having discussed only foreground removal in the main paper due to space constraints, we here present the other editing tasks.

S.9.1. Time Interpolation

We can linearly interpolate between consecutive time steps to enable temporal super-resolution since NR-NeRF optimizes a latent code $\mathbf{l}_i$ for every time step i. We refer to the supplemental video for results.

S.9.2. Deformation Exaggeration/Dampening

We can manipulate the deformation even further. Specifically, We can exaggerate or dampen deformations relative to the canonical model by scaling all offsets with a constant $m \in \mathbb{R}$: $(\mathbf{c}, o) = \mathbf{v}(\mathbf{x} + m\mathbf{b}(\mathbf{x}, \mathbf{l}_i))$. Fig. S.8 contains examples.

S.9.3. Forced Background Stabilization

Since we do not require any pre-computed foreground-background segmentation, NR-NeRF has to assign rigidity scores without supervision. Occasionally, this insufficiently constrains the background and leads to small motion. We can fix this in some cases by enforcing a stable background at test time: we set the regressed score to 0 if it is below some threshold r_{min}. If the rigid background has sufficiently small scores assigned to it relative to the non-rigid part of the scene, this forces the background to remain static for all time steps and views. For results, we refer to the supplemental video.

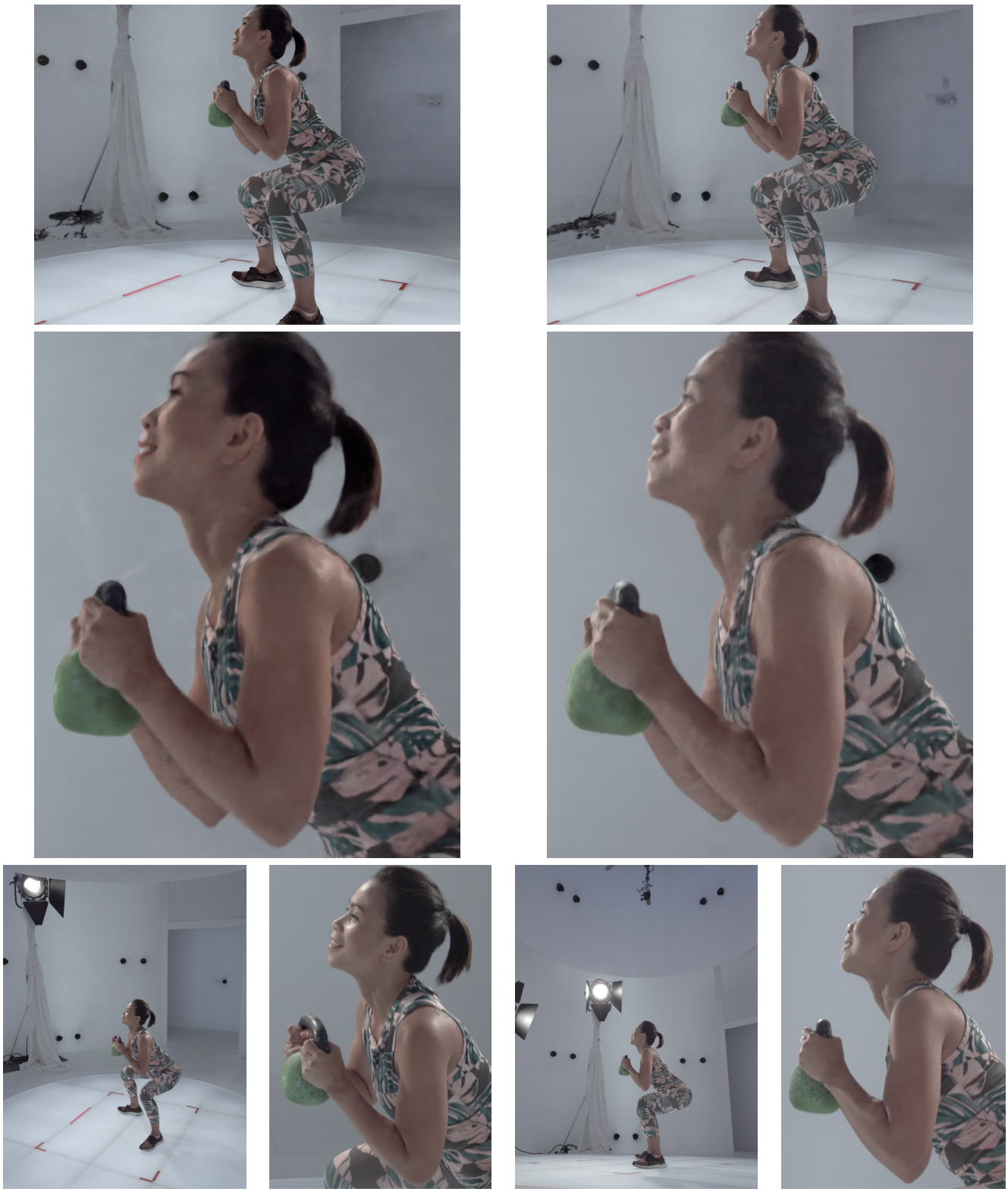


Figure S.5. We explore the upper quality bound of our proposed method using a highly controlled multi-view setting. We can extend our method such that it handles view-dependent effects. Results on the left are without view dependence, while those on the right are with view dependence. We show a full rendering by NR-NeRF (first row), zoom-ins thereof (second row), and input images from the two closest input cameras.

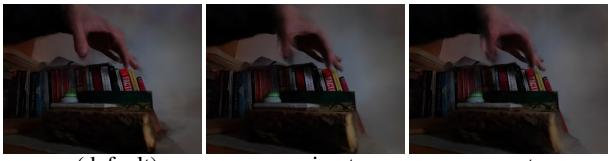


Figure S.6. While NR-NeRF extended with view-dependent effects (approximate or exact) gives similar results to the default NR-NeRF for many monocular scenes, we sometimes observe artifacts for difficult novel views.

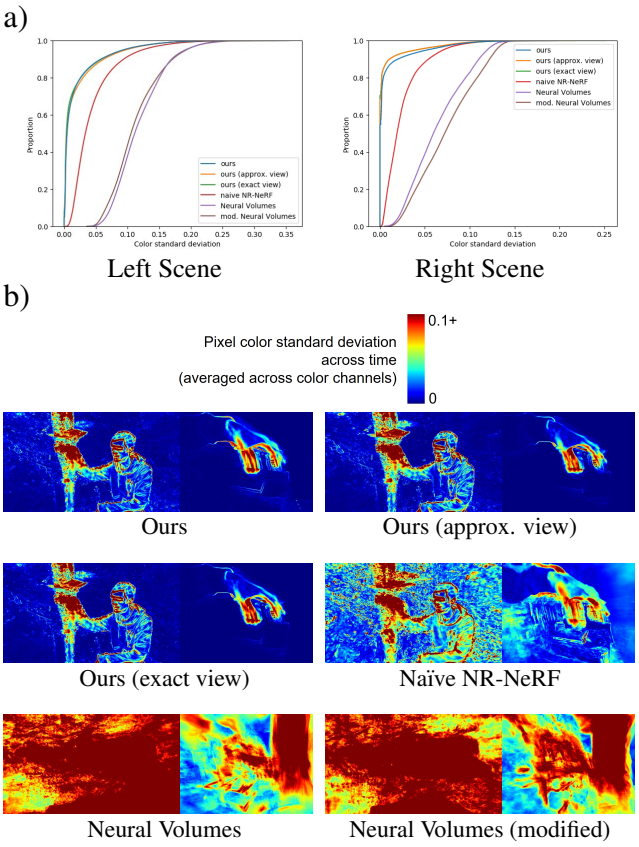


Figure S.7. Background Stability. We quantify the difference in background stability between our method, its variants with view dependence, naïve NR-NeRF, and Neural Volumes. To that end, we render all test time steps of the input sequence into a fixed novel view and compute the standard deviation of each pixel's color across time to measure color changes and hence background stability. **a)** We show cumulative plots across all pixels, where NR-NeRF and its variants (left-most curves) have the most stable background. **b)** We then show how those instabilities are distributed in the scene. The results of NR-NeRF and its variants show the least instability in the background.

S.10. Limitations

We do not account for appearance changes that are due to deformation or lighting changes. For example, temporally changing shadowing in the input images is an issue, as Fig. S.10 demonstrates.

Foreground removal can fail if a part of the foreground is entirely static (*e.g.,* the foot in Fig. S.9).

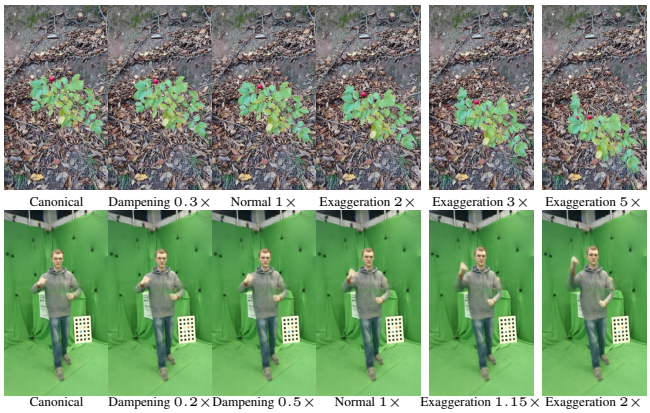


Figure S.8. We exaggerate or dampen the motion relative to the canonical model, and render the result into a novel view.

Figure S.9. (Left) the groundtruth input image and (right) a rendering without non-rigid foreground.

Figure S.10. The input (left) is reconstructed by NR-NeRF (middle). The bottom of the image exhibits local shadowing absent at other time steps, which leads to a high reconstruction error (right).

S.11. Additional Comparisons

We show some preliminary qualitative comparisons to the concurrent, non-peer-reviewed work Neural Scene Flow Fields [3] in Fig. S.11.

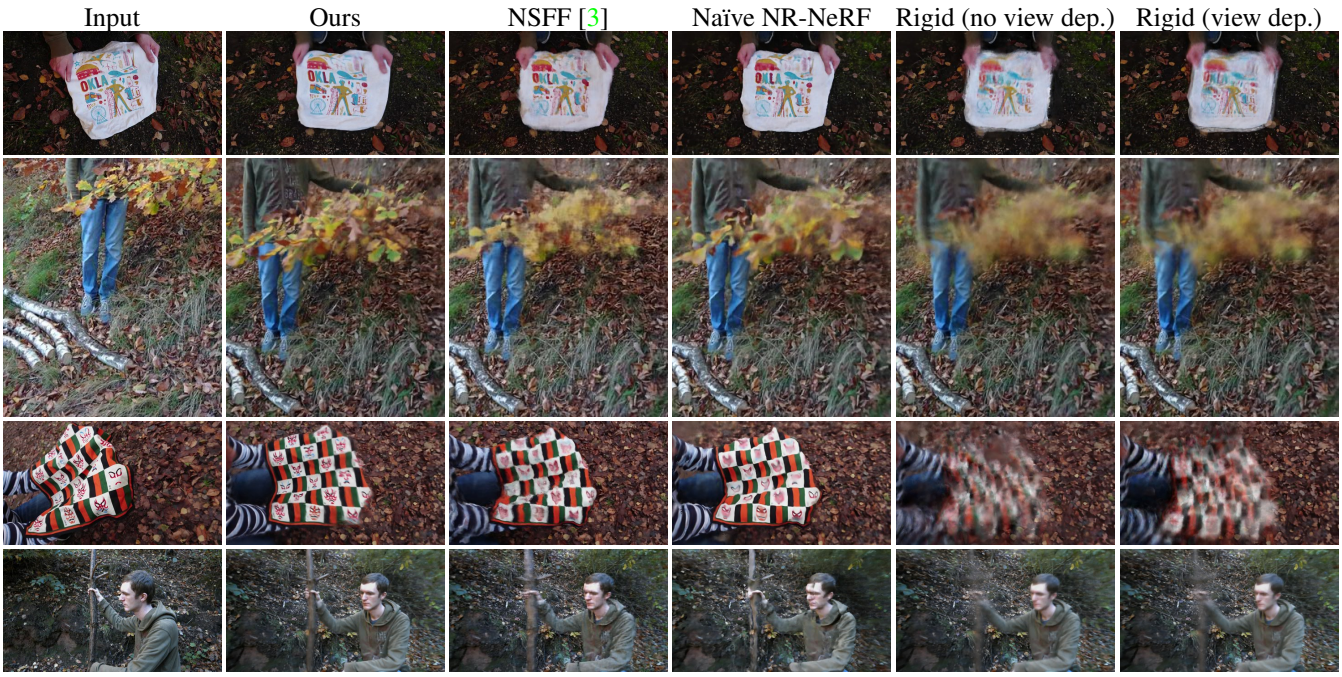


Figure S.11. Under challenging novel view scenarios, our method benefits from the geometry and appearance information that the canonical volume has accumulated from all time steps, which allows NR-NeRF to output sharp results. Both the concurrent, non-peer-reviewed Neural Scene Flow Fields [3] and naïve NR-NeRF however entangle deformation with geometry and appearance by conditioning the 'canonical' volume on a time-dependent deformation latent code. This makes sharing information across time more difficult, leading to blurrier results in challenging novel view scenarios compared to NR-NeRF's results. Finally, rigid NeRF shows a blurry mix of the deformations observed over the entire input sequence, which highlights the need to account for deformations in the scene.

References

[1] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural Scene Flow Fields for Space-Time View Synthesis of Dynamic Scenes. *arxiv*, 2020.

[4] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Trans. Graph. (SIGGRAPH)*, 38(4), 2019.

[5] 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. In *ECCV*, 2020.

[6] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[8] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[9] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[10] Shufeng Tan and Michael L. Mayrovouniotis. Reducing data dimensionality through optimizing neural network inputs. *AIChE Journal*, 41(6):1471–1480, 1995.

[11] Lin Yen-Chen. Nerf-pytorch. https://github.com/yenchenlin/nerf-pytorch/, 2020.

[12] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[13] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.