

# Supplementary Materials for

## STaR: Self-supervised Tracking and Reconstruction of Rigid Objects in Motion with Neural Rendering

### A. Overview

In this document, we provide technical details in support of our main paper. Below is a summary of the contents.

- Sec. **B**: Description of supplementary video;
- Sec. **C**: Mesh reconstruction and 6D pose tracking results;
- Sec. **D**: MLP architecture and volume rendering details;
- Sec. **E**: Synthetic and real-world data preparation;
- Sec. **F**: Derivation of SE(3) pose Jacobian.

### B. Video

We encourage the reader to watch our supplementary video at <https://wentaoyuan.github.io/star>, where we visualize the following results.

- We first show a comparison of STaR against NeRF [3], NeRF-time and NeRF-W [2] on the rendering of novel spatial-temporal views on the *lamp and desk* and *kitchen table* sequences. The rendered videos are 20x slow motion of the training videos from a continuously varying camera view unseen during training, where STaR achieves superior perceptual quality compared to the baselines.
- Then, we visualize the decomposition of static and dynamic components learned by STaR on the *moving banana* sequence by showing how static background and dynamic foreground can be seamlessly removed during spatial-temporal novel view rendering. Similarly, the rendered video is a 20x slow motion of the training videos from a continuously varying camera view unseen during training. We also visualize the disparity map rendered by STaR.
- Finally, we show results of photorealistic animation of unseen object motion in *lamp and desk* and *moving banana*. Specifically, we compose the static and dynamic NeRFs using a set of poses significantly different from the poses seen during training (see Fig. 6 in the main paper for a visualization of the trajectories) and rendered the composed NeRF from a continuously varying camera view unseen during training. Remarkably,

without any prior knowledge about the scene geometry of the object motion, STaR is able to learn a factored representation that allows it to photorealistically synthesize novel spatial-temporal views of novel object motion, which no existing method can do.

### C. Additional Applications

**Mesh Reconstruction** The separation of static and dynamic components learned by STaR allows us to reconstruct a 3D mesh of the unknown dynamic object. Specifically, we can query the dynamic NeRF using a dense 3D grid over a training camera’s view frustum, then threshold the density outputs (i.e. setting  $\sigma^D = 0$  if  $\sigma^D < \sigma_{\min}$ ) and run marching cubes to obtain a 3D mesh. Fig. 1 visualizes the reconstructed meshes of the dynamic objects compared to the ground truth in the two synthetic videos used in our paper, *lamp and desk* and *kitchen table*.

We also use MeshLab to compute the mean Hausdorff distance between the reconstructed mesh and the ground truth. We report the distances in Tab. 1 as percentage of the ground truth mesh’s bounding box diagonal. We use voxel size 0.002 for *lamp and desk*, voxel size 0.0001 for *kitchen table*, and density threshold 50 for both scenes. The reconstructed meshes are post-processed by excluding everything outside of a manually specified 3D bounding box and aligned with the ground truth meshes using ICP.

	Lamp and desk	Kitchen table
Hausdorff distance	0.55%	3.59%

Table 1: Hausdorff distance between the reconstructed and ground truth mesh as percentage of the ground truth mesh’s bounding box diagonal.

**6DoF Pose Tracking** In addition to reconstructing geometry and appearance, STaR also outputs the relative 6D pose between the static and dynamic volume. We can use the output poses to accurately track the relative motion of the

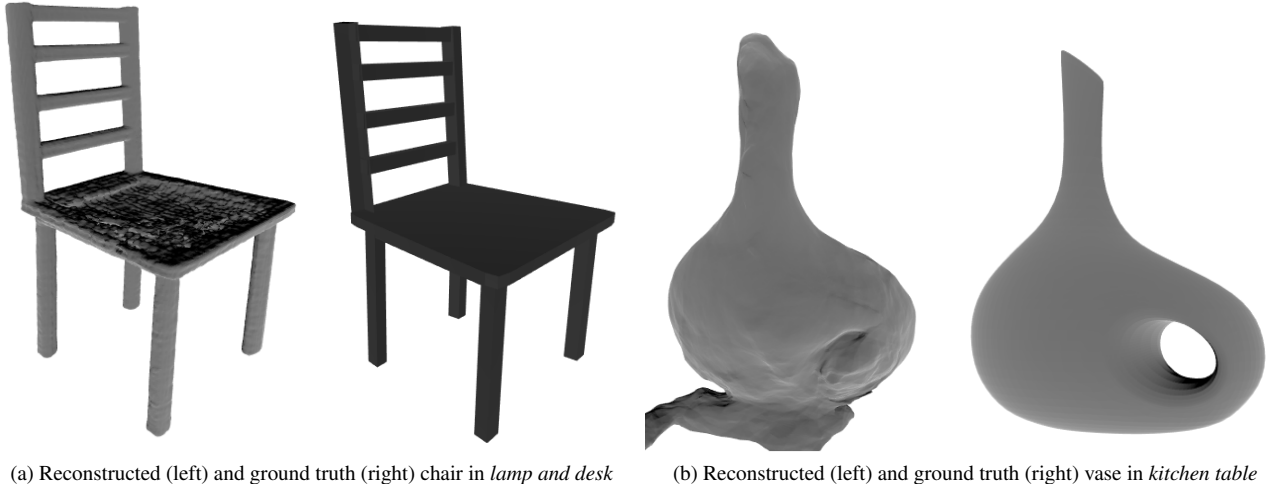


Figure 1: Comparison of reconstructed mesh of the dynamic object and the ground truth.

dynamic object. In Tab. 2, we report error in the relative pose difference of the dynamic object between neighboring key frames estimated by STaR compared against the ground truth. The rotation error is computed in degree and the translation error is computed as percentage of the diagonal of the object’s 3D bounding box.

	Lamp and Desk	Kitchen table
Rotation error	0.502	3.198
Translation error	0.76%	3.60%

Table 2: Rotation and translation error of the dynamic object’s relative motion between key frames estimated by STaR. Translation error is computed as percentage of the object’s bounding box diagonal.

## D. Implementation Details

**MLP Architecture** Fig. 2 shows the architecture of MLPs used by STaR (NeRF), NeRF-time and NeRF-W respectively. STaR uses the same MLP architecture as NeRF for both static volume  $\mathcal{F}_\theta^S$  and dynamic volume  $\mathcal{F}_\theta^D$ . NeRF-time shares the same MLP architecture except for using positional-encoded time as additional input. In practice, the time parameter before positional encoding is the frame index linearly projected on to the interval  $[-1, 1]$ . NeRF-W (more precisely, its variant NeRF-U since we don’t use appearance embedding) uses the same MLP architecture as NeRF for the coarse network, but its fine network takes an additional 16-dimensional latent code and outputs transient density, color and uncertainty in addition to static density and color. Please refer to [2] for more details about the architecture of NeRF-W.

**Volume Rendering** For STaR and all baselines, we use 64 stratified samples per ray for the coarse network and additional 64 importance samples (in total 128 samples) for the fine network. Following [3], we add small Gaussian noise to the density outputs during appearance initialization but turn it off during online training. We adjust the absolute scale of the camera’s view frustum so that it roughly lies within the cube  $[-1, 1]^3$ . For synthetic data, this can be done by scaling the rendered content. For real data, we translate the camera poses so that the world coordinate center aligns with the center of the average camera’s view frustum. Then we scale the camera poses’ translation component by half of the distance from the near bound to the far bound.

## E. Data Preparation

**Synthetic Data Generation** The synthetic video sequences are rendered with Blender Cycles rendering engine using photorealistic assets created by professional designers on Blend Swap. The 8 training cameras are arranged in a  $2 \times 4$  array, focusing on the same target point in the scene. The evaluation camera is also looking at the same direction from a similar distance but its location is different from the training cameras. The camera poses remain fixed throughout the video.

**Real World Data Capture** The real-world video sequence is captured using a 20-camera rig. The cameras are arranged in a  $2 \times 10$  array. We discard 3 cameras that are not synced with the others, use 16 cameras for training and 1 remaining camera for evaluation. The camera poses are also fixed and can be obtained by running COLMAP’s structure from motion pipeline on images from the first frame. The original image resolution is  $2704 \times 2028$  and is downsampled to  $676 \times 507$  for training and evaluation.

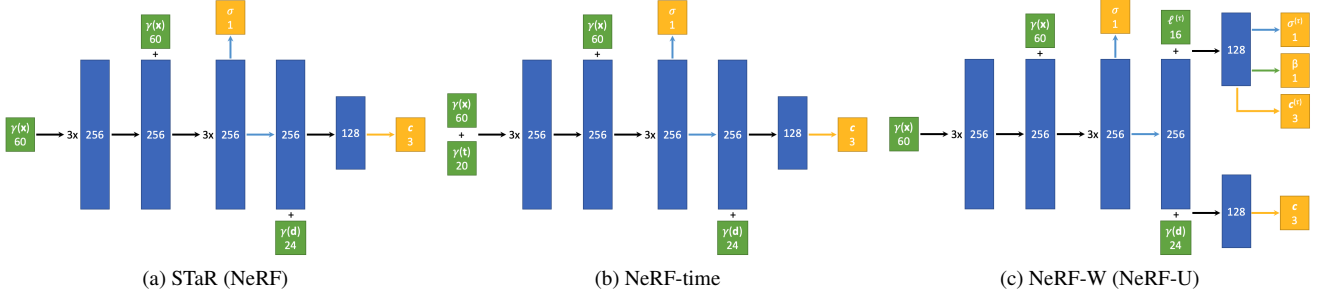


Figure 2: Architecture of MLPs used in STaR (NeRF), NeRF-time and NeRF-W. **Input**, **network layers** and **outputs** are marked in green, blue and yellow respectively, with their dimensions labeled beneath. Blue, black, yellow and green arrows denote linear transformations with **no activation**, **ReLU activation**, **sigmoid activation** and **softplus activation** respectively and + denotes concatenation.  $\gamma$  denotes positional encoding and  $\mathbf{x}, \mathbf{d}, \sigma, \mathbf{c}$  denotes 3D location, viewing direction, volume density and radiance.  $l^{(\tau)}$  is the latent code taken by NeRF-W to generate transient density  $\sigma^{(\tau)}$ , color  $\mathbf{c}^{(\tau)}$  and uncertainty  $\beta$ .

## F. Jacobian for Pose Gradient

Let  $\mathbf{p} \in \mathbb{R}^3$  be a 3D point and  $T \in \text{SE}(3)$  be a pose with associated transformation matrix

$$T = \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 \end{bmatrix} \quad (1)$$

where  $T[\mathbf{p}]$  denotes the transformation of  $\mathbf{p}$  with respect to  $T$ . Let  $\epsilon \in \mathfrak{se}(3)$  be a local perturbation on the  $\text{SE}(3)$  manifold. We are interested in the derivative of  $\exp(\epsilon)T[\mathbf{p}]$  with respect to  $\epsilon$  at  $\epsilon = \mathbf{0}$ :

$$\left. \frac{\partial \exp(\epsilon)T[\mathbf{p}]}{\partial \epsilon} \right|_{\epsilon=\mathbf{0}} = \left. \frac{\partial S[\mathbf{p}]}{\partial S} \right|_{S=\exp(\epsilon)T=T} \left. \frac{\partial \exp(\epsilon)[T]}{\partial \epsilon} \right|_{\epsilon=\mathbf{0}} \quad (2)$$

$$= ([\mathbf{p}^\top \ 1] \otimes \mathbf{I}_3) \begin{bmatrix} \mathbf{0}_{3 \times 3} & -\mathbf{r}_1^\wedge \\ \mathbf{0}_{3 \times 3} & -\mathbf{r}_2^\wedge \\ \mathbf{0}_{3 \times 3} & -\mathbf{r}_3^\wedge \\ \mathbf{I}_3 & -\mathbf{t}^\wedge \end{bmatrix} \quad (3)$$

$$= [\mathbf{I}_3 \quad -(T[\mathbf{p}])^\wedge] \quad (4)$$

where  $\otimes$  denotes Kronecker product. The result is a  $3 \times 6$  Jacobian matrix, where  $\mathbf{x}^\wedge$  is the cross product matrix

$$\mathbf{x}^\wedge = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \quad (5)$$

We encourage the reader to read [1] for more details about the on-manifold optimization of  $\text{SE}(3)$  transformations.

## References

- [1] Jose-Luis Blanco. A tutorial on se (3) transformation parameterizations and on-manifold optimization. *University of Malaga, Tech. Rep*, 3:6, 2010. [3](#)
- [2] 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. *arXiv preprint arXiv:2008.02268*, 2020. [1](#), [2](#)
- [3] 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. pages 405–421, 2020. [1](#), [2](#)