Rigid Object Tracking in a NeRF World

Sanjari Srivastava
sanjari4@stanford.edu

Siddharth Tanwar
tanwar@stanford.edu

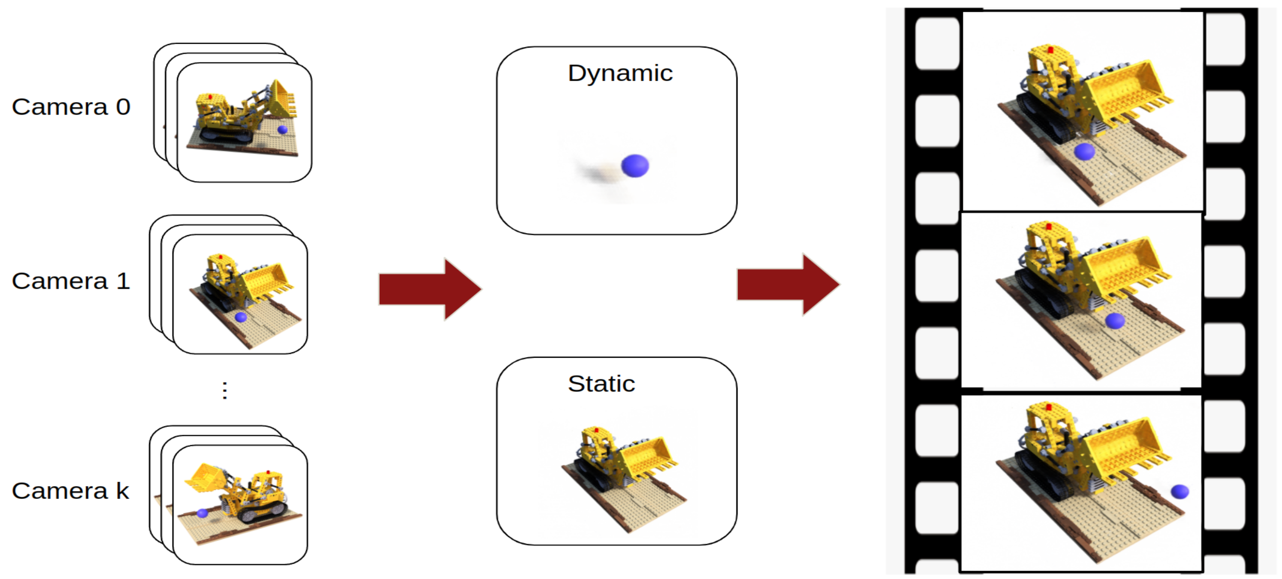


Figure 1: Overview of implemented method: Given Multi-view RGB videos of a dynamic scene, we decompose it into static and dynamic components, which are used to generate novel 4D views.

Abstract

While neural scene representation models, such as neural radiance field (NeRF), are capable of accurately representing scenes when they are static, a single NeRF model is not able to directly capture dynamic objects in view. In this project, we focus on learning neural scene representations for dynamic scenes. We implement "STaR: Self-supervised tracking and reconstruction of rigid objects in motion with neural rendering" [18], which utilizes 2 NeRF models, each to learn the static and the dynamic scene representations respectively. This method can learn the representation of a single, rigid, moving object placed in a static scene. Since the code and dataset of [18] are not yet available, we wrote our own implementation, for which we heavily used [16] as a reference. In order to obtain a dynamic scene with object pose information, we created our own blender dataset containing a moving ball against a lego truck. We implemented entropy regularization and appearance initialization (Section 3.2 in [18]) as well to improve the training and performance of our model. We compare our implemented method STaR with other off-the-shelf neural scene representations: vanilla NeRF[5], DirectVoxGo[14] and D-NeRF[10] training each for a similar number of iterations. We evaluate all models using MSE, PSNR, SSIM and LPIPS calculated on the generated test set images. We highly encourage readers to view animated results here and our code is available at `https://github.com/tanwars/nerf-star`.

1. Introduction

Humans have the capability to perceive the world as 2D images and extract meaningful 3D information from it which is not only restricted to the geometry but also properties such as material, weight, affordance etc. To endow robots with similar capabilities, we require rich representations of 3D scenes and tracking moving objects in this scene. Traditionally, this has been done through hand crafted data structures and hand chosen features [4, 7, 6]. However, such representations are rigid, hard to infer, and not scalable. Neural scene representations (NSRs) seek to alleviate these problem. These representations are capable of parameterizing scene surfaces smoothly, can sample at arbitrary resolutions and are extremely memory compact and memory scalable with scene complexity. Recent advances in NSRs [5, 12, 8, 17, 13, 1, 11] make it an exciting fields of study today.

Application of these representations to robotics is becoming increasingly popular owing largely to their nice properties of differentiability. [13] have shown real-time application of NSRs for simultaneous localization and mapping. NSRs have been used in vison-only robot navigation as well by [1]. Methods like NeRF by [5] are able to generate photo-realistic images by over-fitting to a particular scene using multiple samples. On the other hand, methods like scene representation networks by [12] are capable of learning the scene from a single observation. While significant research is needed to find the right neural representations for robotic tasks, they are nonetheless likely to become common-place representations in robotics.

One of the important problems in robotics is understanding of dynamic scenes using images. For example, tracking moving targets such as pedestrians, other vehicles etc. is crucial for self-driving applications. Moreover, tracking moving objects allows algorithms that rely on static environment constraints for tasks such as SLAM to produce more accurate results. Recent novel methods in this domain make use of NSRs to represent the scene. These methods attain outstanding results with a static environment. An active line of research using NSRs is in representing dynamic scenes and we explore one such method "STaR: Self-supervised tracking and reconstruction of rigid objects in motion with neural rendering" [18] through this project. We implement the method and compare it against other temporal and non-temporal methods presenting qualitative and quantitative results.

1.1. Related Work

Novel view synthesis is a long standing problem where new views are synthesized using arbitrary views of the same scene captured in multiple images. The traditional approaches include Struture-from-Motion, bundle adjustment or light based photography. More recently, deep learning based methods are able to learn volumetric representation from sparse images [[3], [8], [9], [17]]. Some examples which we focused on were depth supervised nerf model DS-NeRF [3] and a fast training model which utilized voxel based grid optimization, DirectVoxGo [2]. Another class of NeRFs are deformable NeRFs which evaluate deep learning based methods on scenes undergoing a variety of deformations. D-NeRFs or Dynamic-NeRFs ([10]) optimizes an underlying deformable volumetric function from a sparse set of input monocular views without the need of ground-truth geometry nor multi-view images. It considers time as an additional input to the system, and splits the learning process into two stages: one that encodes the scene into a canonical space and another that maps this canonical representation into a dynamic scene at a particular time. Similarly, STaR [18] uses two NeRFs to learn the static and the dynamic model respectively, but the dynamic model uses 3D points transformed to the object frame instead of using time as an input. This allows us to completely decouple the dynamic object representation from the static scene. This is a significant advantage over deformable NeRF models because we can not only synthesize images at any time step, but also arbitrarily change the object pose and generate novel views. This makes our implemented method also related to other object-level compositional methods such as [15], [19], [8].

2. Problem Statement

The key task that we wished to accomplish through this project was as follows,

Decomposition of a scene into static and dynamic components represented as NSRs and tracking the dynamic rigid objects.

For our problem, we assumed observations of a single rigid object moving in an unknown environment. To accomplish this task, we followed STaR [18], which is a novel method for tracking and reconstructing dynamic scenes from multi-view RGB videos. To make the scope of the problem smaller, we assumed that we are provided with the rigid transform of the moving object with respect to the world frame. This enabled us to avoid having to solve the problem of rigid pose optimization (Section 3.2 in [18]) for the purpose of this project. Since we required a known object-to-world transform, we created our own synthetic dataset in blender which provided us with this information, instead of using a real world video dataset.

3. Methodology

3.1. Preliminaries: Neural Radiance Field

A neural radiance field (NeRF) is a small fully connected network (MLP) trained to reproduce input views of a single scene using a rendering loss, which allows creation of novel views. It maps a 3D scene coordinate $r(s) \in \mathbb{R}^3$ and view direction $\mathbf{d} \in S^2$ to volumetric density $\sigma \in \mathbb{R}$ and the view dependent emitted radiance $\mathbf{c} \in \mathbb{R}^3$ in that spatial location. ($\mathbf{r}(s) = \mathbf{o} + s\mathbf{d}$ is the point on the ray from camera origin o with a depth of s). The color of the pixel is found by integrating the accumulated radiance along these sampled points on the ray, and can be approximated by the following

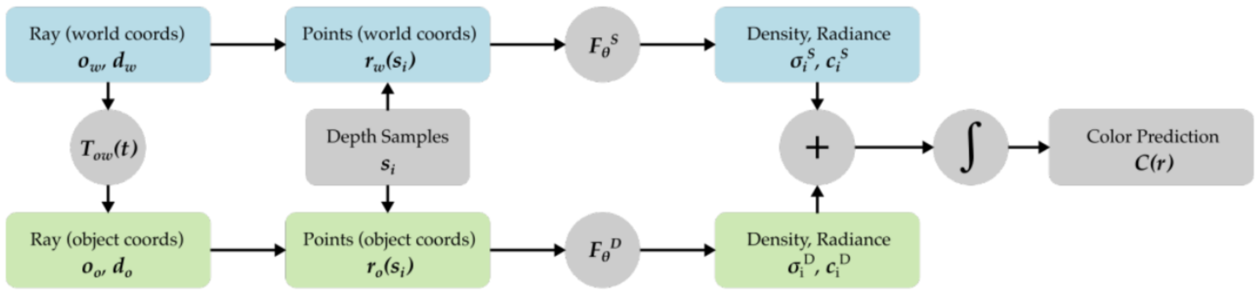


Figure 2: Representation of STaR architecture for learning single, rigid body motion from sparse images. (Image taken from [18])

equation:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \alpha_i \mathbf{c}_i \tag{1}$$

$$\text{where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \left(s_{j+1} - s_j\right)\right) \tag{2}$$

$$\text{and } \alpha_i = 1 - \exp\left(-\sigma_i \left(s_{i+1} - s_i\right)\right) \tag{3}$$

s_i denotes the i^{th} point sampled in between the near bound and the far bound along any ray and $\sigma_i = \sigma\left(\mathbf{r}\left(s_i\right)\right), \mathbf{c}_i = \mathbf{c}\left(\mathbf{r}\left(s_i\right)\right)$ are the evaluated volume density and radiance at the sampled points.

Also, note that a NeRF is internally composed of two MLPs which are optimized together: “coarse” and “fine”. Firstly, the coarse grained model is evaluated on linearly sampled points and then a second set of points is sampled from the pieces of the rays which are more ”relevant”. The color value can be said to be a weighted average of c_i’s weighted by $T_i\alpha_i$ and importance sampling is performed on the same rays using these weights so that more samples are allocated to regions which contain visible content.

The training loss is the mean squared error between the actual and the predicted pixel color values, $C(\mathbf{r})$ and $\hat{C}(\mathbf{r})$.

3.2. STaR

The model described above can only represent time-invariant scenes. To tackle this, STaR uses two NeRFs (each containing a fine and a coarse model, hence four MLPs in total) which are optimized simultaneously. The first NeRF (static, *S*) is exactly the same as was described above and the other model learns the dynamics of the scene (dynamic, *D*) by employing the same algorithm but on 3-D points transformed to the object frame instead of the world coordinates,

$$F_\theta^S : \mathbf{r}\left(s_i\right), \mathbf{d} \rightarrow \sigma_i^S, \mathbf{c}_i^S \tag{4}$$

$$F_\theta^D : \mathbf{r}\left(s_i\right), \mathbf{d}, \xi(t) \rightarrow \sigma_i^D, \mathbf{c}_i^D \tag{5}$$

$\xi(t)$ represents the set of time dependent rigid object poses that define the world to object frame transformation.

The color values obtained by both the models (1) are combined together using alpha blending. The final $\hat{C}(\mathbf{r})$ is given by the following equation:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \left(\alpha_i^S \mathbf{c}_i^S + \alpha_i^D \mathbf{c}_i^D\right) \tag{6}$$

$$\text{where } T_i = \exp\left(-\sum_{j=1}^{i-1} \left(\sigma_j^S + \sigma_j^D\right)\left(s_{j+1} - s_j\right)\right) \tag{7}$$

$$\text{and } \alpha_i^S = 1 - \exp\left(-\sigma_i^S \left(s_{i+1} - s_i\right)\right) \tag{8}$$

The overall architecture is shown in Fig. 2.

3.3. Optimizing STaR

STaR optimizes a composite objective (Eq. 9 - 10) where the first term denotes the mean squared error of the predicted color values and the actual color values ($\hat{C}_f$ and $\hat{C}_c$ are the alpha-blended outputs of the fine and the coarse models respectively). The second is an **Entropy Regularization** term which regularizes the entropy $\mathcal{H}$ of the rendered transparency values given by the static and the dynamic models.

$$\mathcal{L} = \sum_{\mathbf{r}\in\mathcal{R}} \left(\left\|\hat{C}_c(\mathbf{r}) - C(\mathbf{r})\right\|^2 - \left\|\hat{C}_f(\mathbf{r}) - C(\mathbf{r})\right\|^2\right) \tag{9}$$

$$+ \beta \sum_{i=1}^{M} \left(\mathcal{H}\left(\alpha_i^S\right) + \mathcal{H}\left(\alpha_i^D\right) + \mathcal{H}\left(\alpha_i^S, \alpha_i^D\right)\right) \tag{10}$$

The entropy term itself consists of two parts. The first part $\mathcal{H}\left(\alpha_i^S\right) + \mathcal{H}\left(\alpha_i^D\right)$ represents the self entropy of individual models which push the transparency values to be either 0 or 1. The equation for computing $\mathcal{H}(\alpha_i^S)$ is provided below and $\mathcal{H}(\alpha_i^D)$ can be computed in the same way:

$$\mathcal{H}\left(\alpha_i^S\right) = \alpha_i^S \log \alpha_i^S + \left(1 - \alpha_i^S\right) \log\left(1 - \alpha_i^S\right)$$

The term $\mathcal{H}\left(\alpha_i^S, \alpha_i^D\right)$ is the cross entropy terms which prevents both models from learning large densities at the same point, forcing them to learn dissimilar components of the scene.

$$\mathcal{H}\left(\alpha_i^S, \alpha_i^D\right) = \left(\bar{\alpha}_i^S \log \bar{\alpha}_i^S + \bar{\alpha}_i^D \log \bar{\alpha}_i^D\right)\left(\alpha_i^S + \alpha_i^D\right)$$

We also use the technique **Appearance initialization**, where the static model is first trained alone to get a good initialization for the static scene. This is followed by the combined training of the static and dynamic model as described in Eqs. 4-5 to obtain a much better representation for the dynamic object. For our project, we used a checkpoint of nerf-lego trained for 200k iterations to obtain this initialization.

4. Experimental Setup

4.1. Dataset

Since the STaR [18] paper did not have a publicly available dataset, we generated our own synthetic dataset in blender with a static and dynamic component. We reused scripts from NeRF [16] dataset repository, and add a moving ball to the lego scene. We capture images of dimension 400×400 for 15 timesteps, and from 15 camera angles at each time step. Figs. 3 and 4 show snapshots of the collected dataset and the camera angles used. For each captured image, we store the following metadata in a .json file: image file name, camera ID, frame ID, camera pose matrix (in world frame), and the object pose matrix (in world frame). We split this data into 100 training images, 100 test images and 25 validation images. Our dataset, trained models, and results can be found at the following google drive link: `https://bit.ly/3IoESNh`.

4.2. Training Specifications

We trained our models on a GCP VM instance with the machine specs in Table 1. Our code-base is implemented in python using Pytorch and is developed on top of the code from [16].

OS	Ubuntu 20.04
CPU RAM	8 GB
GPU	NVIDIA Tesla K80
GPU Memory	12 GB

Table 1: Training machine specs

4.3. Model Specifications

We use four multi-layer perceptrons (MLPs): two as the coarse and fine NeRFs for the static component and similarly for the dynamic component. Each model is 8 layers deep with each layer 256 wide with ReLU activation. The input is embedded to a higher dimension using periodic functions before passing to the MLP. We use an ADAM optimizer in training. The hyper-parameters for training are given in Table 2.

Parameter	Value
learning rate α	0.0005
α decay rate	0.1
β (From Eq. 10)	0.002
ADAM β	(0.9,0.999)

Table 2: List of parameters. A detailed list of parameter values is listed in the codebase `https://github.com/tanwars/nerf-star`

5. Experiments

We compare our implemented method with two non-temporal methods: NeRF [5], and DirectVoxGo [14], as well as a temporal method: D-NeRF [10]. We trained the non-temporal methods to demonstrate their inability to capture the dynamic component of the scene. Comparison with D-NeRF is shown to demonstrate the advantages of STaR against other temporal models.

5.1. Qualitative Comparison

5.1.1 Vanilla NeRF

Since this is a non-temporal method, all images are treated as captured at the same time step. We train a basic NeRF model [5] for 14,000 iterations ($\sim$7 hours of training on our hardware). We, then, use this trained model to render images from novel view directions generated for visualization. Fig. 5a shows some examples of the rendered images. We can see that the moving ball is rendered as a volume cloud on its trajectory and its image not sharp. This is expected since the model assumed all images to occur at the same time step so learnt ball positions at different spatial coordinates and mixed them into a single image. For comparison, Fig. 6 shows the ground truth images.

5.1.2 DirectVoxGo

Since this is a non-temporal method, all images are treated as captured at the same time step. We train the model [14] for 20,000 iterations ($\sim$1 hour of training on our hardware). We chose this model for comparison for two primary reasons: firstly, this method trains much faster compared to the basic NeRF implementation, and secondly, the network flow for this model is different than a typical NeRF and using this for comparison allowed for variety. Fig. 5b shows the rendered images obtained from this method for novel

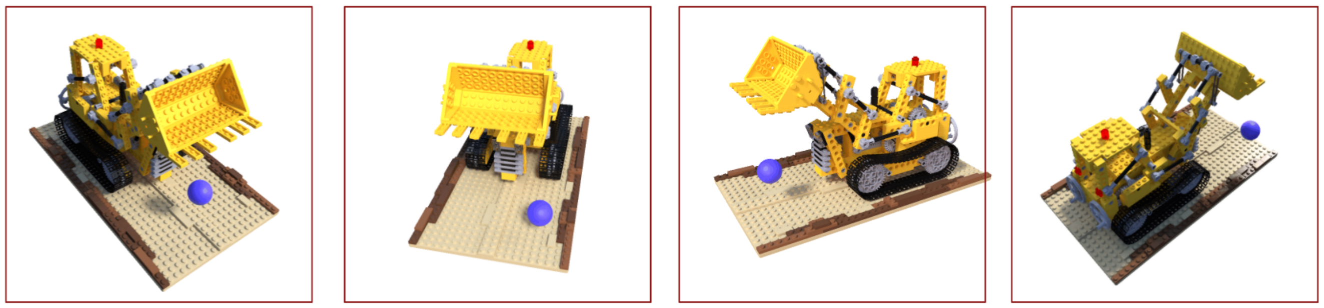


Figure 3: Synthetic dataset generated in blender using 15 camera views and 15 time steps. **Static scene**: lego truck. **Dynamic component**: the purple ball.

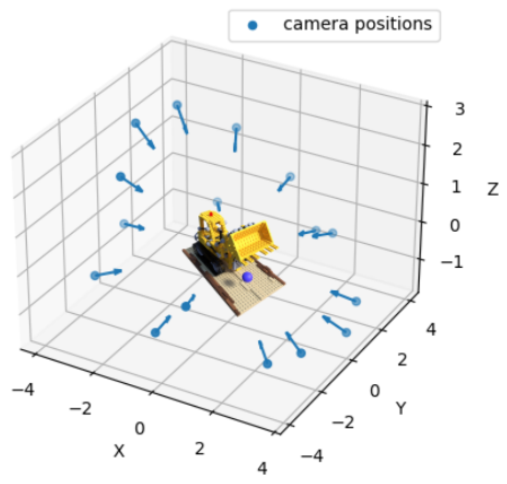


Figure 4: Camera positions for the synthetic dataset

view points. While the results are sharper and qualitatively better looking than the NeRF model, it still fails to sharply predict the position of the ball in the image and there are clearly visible artifacts.

5.1.3 D-NeRF

D-NeRF is a temporal method and takes into account the time step for each frame in estimating the novel image view. We trained this model for 14,000 iterations ($\sim$7 hours of training on our hardware). We show the results of the novel view synthesis in Fig. 5c. We see poor performance from this method although the ball position is sharper. We attribute the poor performance to low number of training iterations as the loss was still decreasing when we stopped the training. For reference, the authors performed $\sim$800,000 iterations for their results. This would have taken a large amount of time and resources and we were limited in both aspects for the class project.

5.1.4 STaR - trained from scratch

We trained our model for 14,000 iterations as well ($\sim$8 hours of training on our hardware). In this version, we used no entropy regularization or appearance initialization. Fig. 5d shows the results of the rendered images from novel views. We can see that while the quality of the image overall is poor, the ball is much more clearly visible in a single position. The overall quality is poor again because of the low number of training iterations which was a limiting factor due to limited resources. When compared to D-NeRF, however, we see significantly better training output in a similar number of iterations.

5.1.5 STaR - trained with a checkpointed static model

A significant advantage of a method such as STaR over methods such as D-NeRF is the explicit separation of dynamic and static components in the scene. We have seen in Fig. 5d that by training for just 14,000 iterations does not result in qualitatively good results for either STaR or D-NeRF. However, with the STaR model, we can swap out the static NeRF model weights with the online available weights of the lego truck trained for $\sim$200,000 iterations and train the static and dynamic components jointly for another 14,000 iterations to get good qualitiative results as shown in Fig. 5e.

Essentially, the STaR pipeline allows us to quickly learn a dynamic component in a known static scene since the two models are separable. Such a decomposition is not possible for a model such as D-NeRF and hence the advantage. In this model, we also added the entropy regularization term to our loss function, to improve the model's performance.

5.2. Quantitative Comparison

Table 3 shows the evaluation of the quantitative metrics: Mean Squared Error (MSE), Peak Signal-to-noise Ratio (PSNR), SSIM, and LPIPS [20] on the test set for the models listed above. We see that on the compositie scene,

Method	MSE ↓	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF [5]	0.00878	20.56	0.8392	0.2036
DVGO [14]	0.00794	21.01	**0.9167**	**0.1177**
D-NeRF [10]	0.01597	17.97	0.74001	0.3240
Ours	0.01104	19.57	0.7488	0.3054
Ours (ckpt)	**0.00695**	**21.57**	0.8886	0.1721

Table 3: Quantitative results on composite scene. Evaluated on the test image set.

the metrics show that a trained STaR model has the highest PSNR and lowest MSE. DVGO, however, has a much lower LPIPS value compared to the other methods. This indicates the images generated by this method are most realistic looking to the human eye. This result is most likely because of the strong influence of the static component of the scene since the ball is only a small part of the entire image. We should also be able to counter this by training the NeRF based models for a much larger number of iterations. An idea to get metric values to more accurately represent what we observe qualitatively is to separate the images into static and dynamic components by cropping the dynamic object using a bounded box and evaluating the metrics for the two components separately.

5.3. Separation of Static and Dynamic Components

To further demonstrate that our trained STaR model is actually learning the underlying static and dynamic components of the scene, we rendered images from the static and dynamic NeRFs separately. Fig. 7 shows an example of these rendered images. We can see that the network does learn the static and dynamic components of the scene. We can also extract the mesh geometry from these individual components and is shown in Fig. 8. This demonstrates that our trained model is capable of learning the static and dynamic components of a scene.

5.4. Novel Pose of Dynamic Object

We further demonstrate the ability of the model to render the scene with an arbitrary user provided trajectory of the dynamic object. Fig. 9 shows the ball placed on top of the lego truck. The model never saw this placement of the object during training and yet is able to render the images. This is another advantage of this method over D-NeRF which cannot synthesize images with novel object placement.

6. Conclusion

In this project, we implemented a method STaR [18] that decomposes a scene into dynamic and a static components and learns an NSR for each. We used NeRF as our NSR in this implementation, however, the method is agnostic to any NSR that has an input, output mapping similar to that of a NeRF model. We used blender to generate synthetic dataset with a single moving ball in an otherwise static scene and demonstrated that our method effectively learns both the static and dynamic components of the scene. We further compared our implementation to other open source methods and show the qualitative and quantitative comparison. We highlight the advantages of this method in terms of novel object placement and efficient learning when compared to deformable NeRF models. Our code is available at `https://github.com/tanwars/nerf-star`, supplementary material at `https://bit.ly/3IoESNh` and animations at `https://bit.ly/3IskudY`.

6.1. Future Directions

Learning object transformation: We assume in this project that the object pose is known or estimated by a prior method. We can incorporate the object pose learning in the framework itself as demonstrated in [18] and learn it at training time.

Generalization: While we implement the method using NeRF as the underlying model, the STaR method is agnostic to the structure of the underlying model as long as it is has the input-output mapping of a NeRF model. This allows us to exptend this methodology to models such as DVGO [14].

Real World Dataset: With either an object pose estimation method, or incorporating object pose learning in the framework, it should be possible to test our implementation on a real world dataset.

References

[1] M. Adamkiewicz, T. Chen, A. Caccavale, R. Gardner, P. Culbertson, J. Bohg, and M. Schwager. Vision-only robot navigation in a neural radiance world. *arXiv preprint arXiv:2110.00168*, 2021.

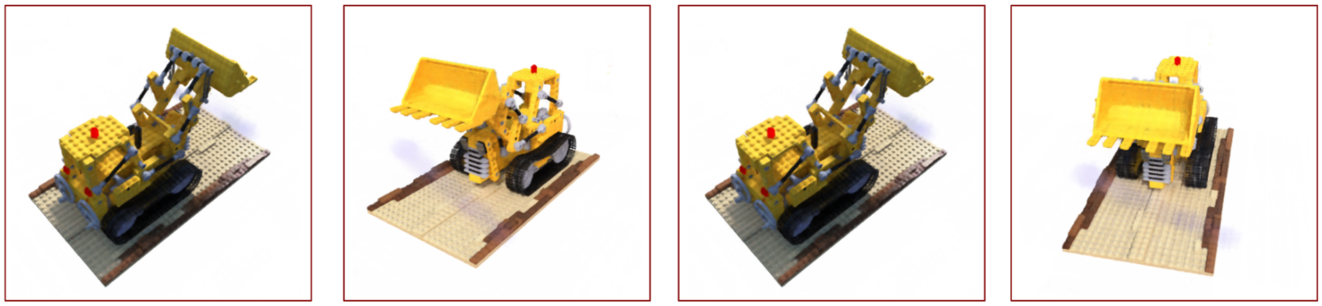
[2] Z. Chen and H. Zhang. Neural marching cubes. *arXiv preprint arXiv:2106.11272*, 2021.

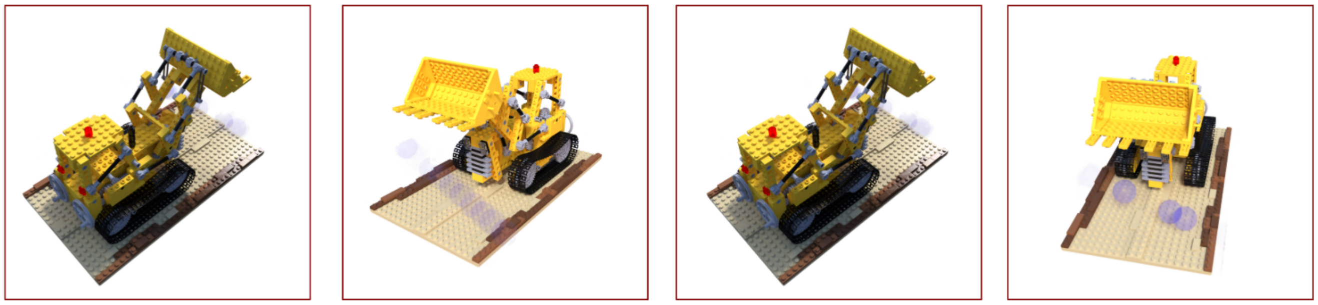
[3] K. Deng, A. Liu, J.-Y. Zhu, and D. Ramanan. Depth-supervised nerf: Fewer views and faster training for free. *arXiv preprint arXiv:2107.02791*, 2021.

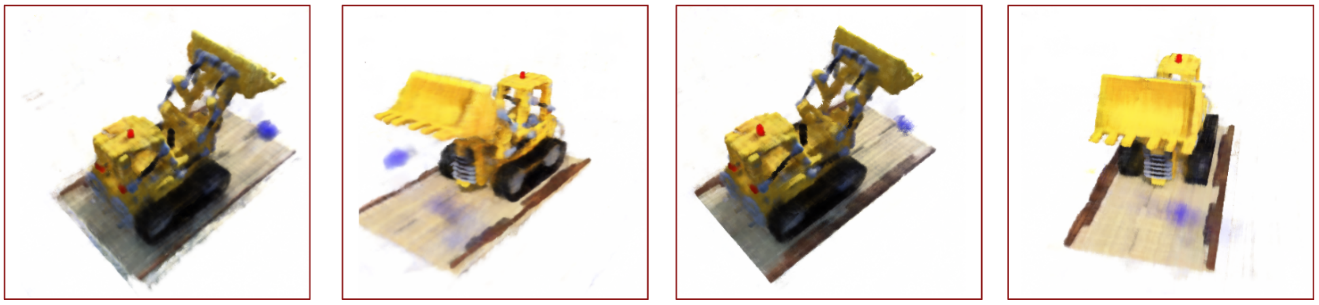
[4] D. Gálvez-López and J. D. Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

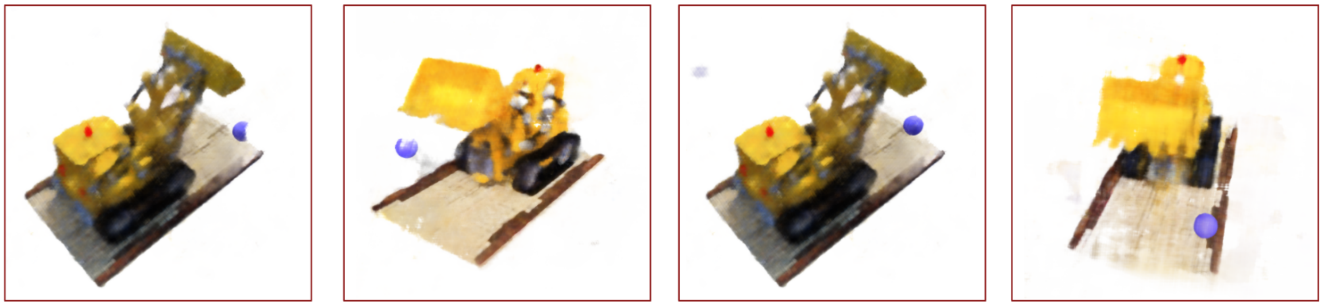
[5] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[6] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE international symposium on mixed and augmented reality*, pages 127–136. IEEE, 2011.


(a) NeRF [5]


(b) DirectVoxGo [14]


(c) D-NeRF [10]


(d) STaR trained from scratch **(Our implementation)**

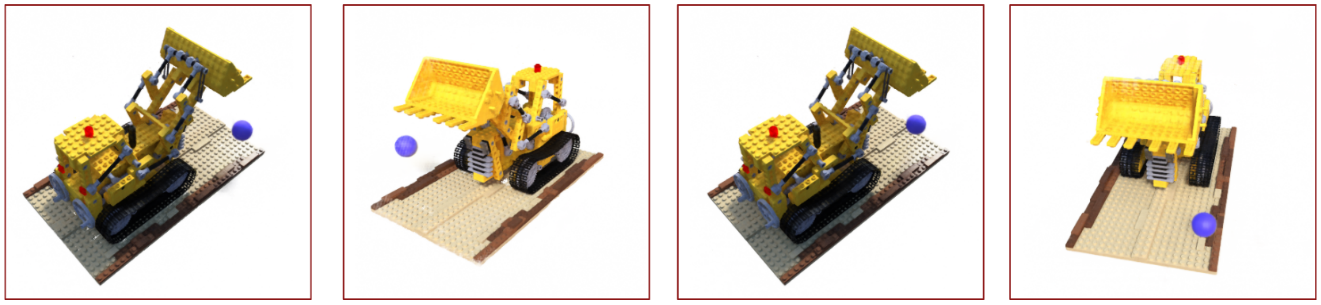

(e) STaR with checkpointed static model **(Our implementation)**

Figure 5: Qualitative results showing synthesized images for four randomly picked views from the test set using different methods.

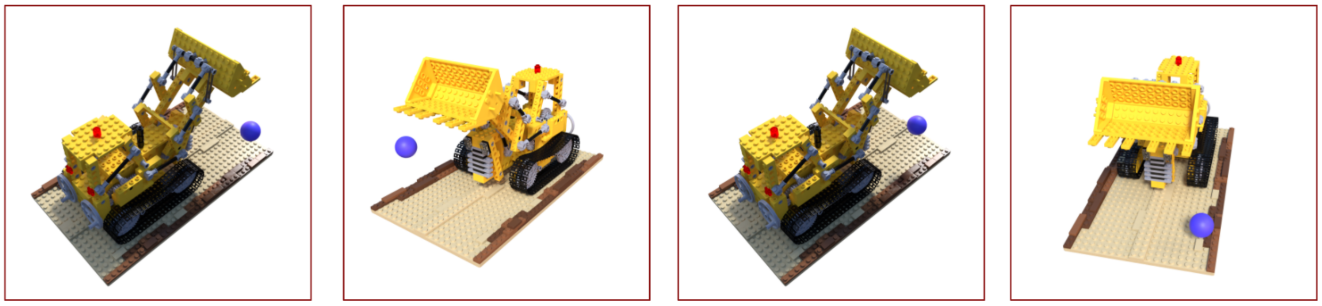


Figure 6: Ground truth test images for which the views are synthesized in Fig 5

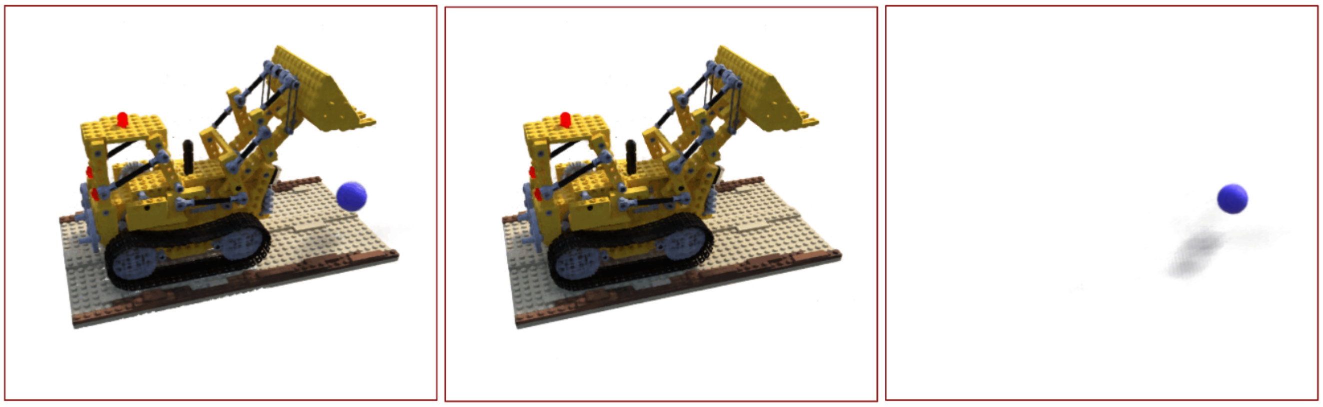


Figure 7: (left) a complete synthesized scene using both the static and dynamic components, (center) rendered image using only the static model, (right) rendered image using only dynamic model. The figure shows that the static and dynamic models learn the static lego truck and the dynamic ball respectively.

[7] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pages 2320–2327. IEEE, 2011.

[8] M. Niemeyer and A. Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11453–11464, 2021.

[9] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. Goldman, S. Seitz, and R. Martin-Brualla. Deformable neural radiance fields. *https://arxiv.org/abs/2011.12948*, 2020.

[10] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. *arXiv preprint arXiv:2011.13961*, 2020.

[11] V. Sitzmann, S. Rezchikov, W. T. Freeman, J. B. Tenenbaum, and F. Durand. Light field networks: Neural scene representations with single-evaluation rendering. *arXiv preprint arXiv:2106.02634*, 2021.

[12] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *arXiv preprint arXiv:1906.01618*, 2019.

[13] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison. imap: Implicit mapping and positioning in real-time. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6229–6238, 2021.

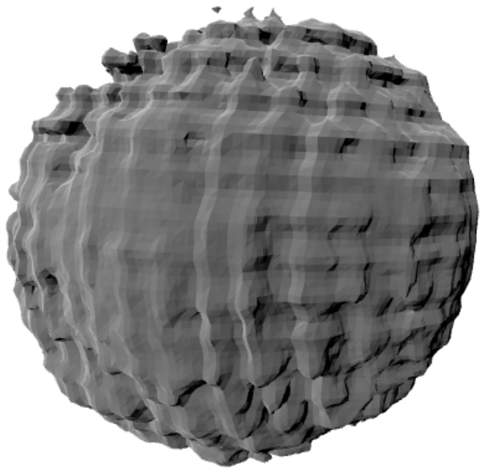
[14] C. Sun, M. Sun, and H.-T. Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. *arXiv preprint arXiv:2111.11215*, 2021.

[15] B. Yang, Y. Zhang, Y. Xu, Y. Li, H. Zhou, H. Bao, G. Zhang, and Z. Cui. Learning object-compositional neural radiance field for editable scene rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13779–13788, 2021.

[16] L. Yen-Chen. Nerf-pytorch. `https://github.com/yenchenlin/nerf-pytorch/`, 2020.

[17] L. Yen-Chen, P. Florence, J. T. Barron, A. Rodriguez, P. Isola, and T.-Y. Lin. inerf: Inverting neural radiance fields for pose estimation. *arXiv preprint arXiv:2012.05877*, 2020.

[18] W. Yuan, Z. Lv, T. Schmidt, and S. Lovegrove. Star: Self-supervised tracking and reconstruction of rigid objects in motion with neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13144–13152, 2021.


(a) Geometry of the ball learnt by the dynamic NeRF model

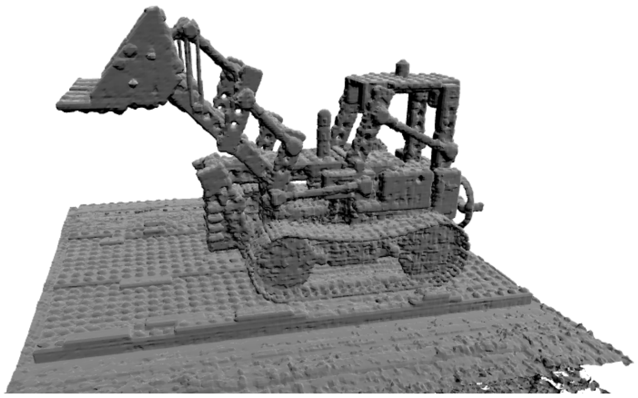

(b) Geometry of the static scene learnt by the static NeRF model

Figure 8: Reconstructed 3D meshes of the dynamic and static models using marching cube.

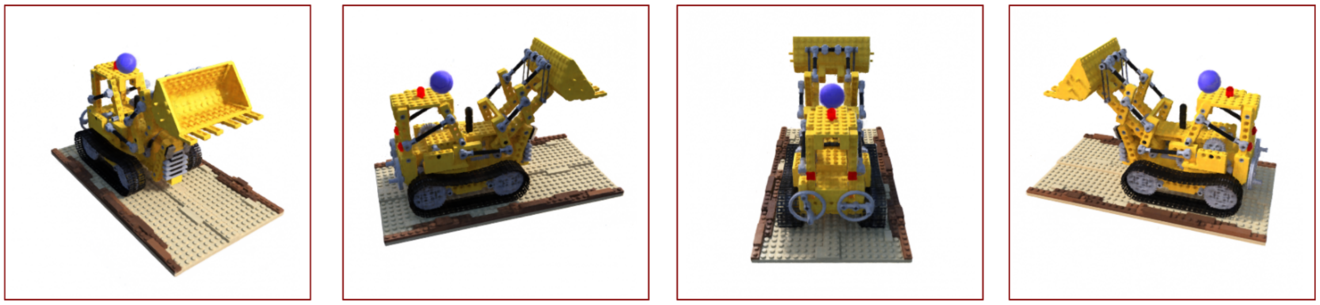


Figure 9: Novel object trajectory rendered. Images are synthesized with the ball placed on top of the lego truck, a scenario never before seen by the models during training. The images showcase the ability of STaR to synthesize novel views with novel trajectories of the object.

[19] K. Zhang, G. Riegler, N. Snavely, and V. Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020.

[20] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.