LU-NeRF: Scene and Pose Estimation by Synchronizing Local Unposed NeRFs — Supplementary Material

Zezhou Cheng[1,2] Carlos Esteves[2] Varun Jampani[2] Abhishek Kar[2]
Subhransu Maji[1] Ameesh Makadia[2]
[1]University of Massachusetts, Amherst [2]Google Research

Contents

1. Experimental details

1.1. Training GNeRF

We trained GNeRF [7] using the official codebase[1]. The scores of GNeRF in our experiments are overall better than those reported in VMRF [11], likely because we trained GNeRF for more iterations. In our experiments, we train GNeRF for 60K iterations which take 48 hours on a single NVIDIA GeForce GTX 1080Ti. We notice that GNeRF is prone to mode collapse in the adversarial training stage, *i.e.*, the generator produces the same or similar sets of outputs with negligible variety, which is a well-known issue for GAN-based models [3]. To achieve similar performance reported in GNeRF and VMRF, we train 5 GNeRF models per prior pose setting and report the results from the best one selected according to the performance on the validation set. Specifically, 35% of the training trials (26 out of 75) suffered from the mode collapse issue on the unordered image collections.

[1] https://github.com/quan-meng/gnerf

1.2. Test-time optimization for view synthesis

Following the procedure established by prior works [6, 7] for evaluating novel view synthesis, we register the cameras of the test images using the transformation matrix obtained via Procrustes analysis on the predicted and groundtruth cameras of the training images; we then optimize the camera poses of the test images with the photometric loss to factor out the effect of the pose error on the view synthesis quality. In the GNeRF pipeline, the test-time optimization is interleaved with the training process, so the total number of iterations depends on the number of training steps seen for the best checkpoint. In our experiments, GNeRF runs approximately 4230 test-time optimization iterations.

1.3. Qualitative and quantitative comparisons

In the main text, we quote the scores from VMRF where the results on Hotdog are missing. Here we also train GNeRF using the official codebase and report the results in Table 1. This allowed us to generate GNeRF results on the Hotdog scene, and observe the mode collapse reported in the previous section.

Overall, our method outperforms both GNeRF and VMRF under an unconstrained pose distribution, while also being more general – our method works with arbitrary 6 degrees-of-freedom rotations (6DOF), while the baselines assume upright cameras (2DOF) on the sphere, even when the range of elevations is not constrained.

2. Supplementary analysis

2.1. Pose synchronization and refinement

Table 2 demonstrates the necessity of our pose synchronization and refinement steps. The pose synchronization aggregates the local pose estimation from LU-NeRF and provides a rough global pose configuration, and the pose refinement step further improves the global poses.

	Chair			Hotdog			Drums			Lego			Mic		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
GNeRF 90°	31.30	0.95	0.08	32.00	0.96	0.07	24.30	0.90	0.13	28.52	0.91	0.96	31.07	0.09	0.06
GNeRF 120°	28.31	0.92	0.12	25.91	0.92	0.15	22.04	0.88	0.19	23.10	0.86	0.95	25.98	0.16	0.08
GNeRF 150°	22.63	0.88	0.22	23.03	0.90	0.24	20.11	0.87	0.21	22.02	0.85	0.93	22.71	0.18	0.12
GNeRF 180° (2DOF)	21.60	0.87	0.25	24.57	0.92	0.18	18.94	0.84	0.30	20.48	0.85	0.20	**23.80**	**0.94**	0.12
Ours (3DOF)	**30.57**	**0.95**	**0.05**	**34.55**	**0.97**	**0.03**	**23.53**	**0.89**	**0.12**	**28.29**	**0.92**	**0.06**	22.58	0.91	**0.08**

Table 1. **Novel view synthesis on unordered image collection.** We trained the GNeRF with the publicly available code. Each GNeRF variation is described by the assumed elevation range. GNeRF 180° is the closest to our method but still has only 2 degrees of freedom for assuming upright cameras. Our method outperforms the unconstrained GNeRF while being more general for considering arbitrary rotations with 6 degrees-of-freedom.

	Rotation Error [°]		Translation Error	
Scenes	Pose Sync.	Pose Refine.	Pose Sync.	Pose Refine.
Lego	16.50	0.07	0.85	0.00
Chair	20.53	4.24	1.08	0.16
Hotdog	21.06	0.23	1.17	0.01
Drums	14.30	0.05	0.86	0.00
Mic	35.48	0.84	1.90	0.02
Mean	21.57	1.09	1.17	0.04

Table 2. **Pose synchronization and refinement.** The pose synchronization step provides a rough global pose configuration, and all camera poses are further optimized during the pose refinement step. We use unordered image collections in this experiment.

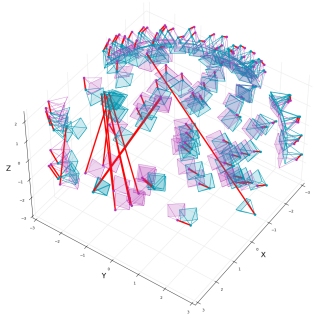

Figure 1. **Camera pose predictions on Drums.**

Scenes	LU-NeRF w/o depth regularization			LU-NeRF w/ depth regularization		
	Mean	Median	Max	Mean	Median	Max
Chair	14.18	13.33	38.26	**9.41**	**4.89**	**33.05**
Hotdog	10.75	9.49	**29.41**	**10.69**	**7.77**	33.29
Lego	10.50	10.08	**28.27**	**5.58**	**1.33**	30.88
Mic	12.88	11.70	**25.99**	**10.27**	**7.05**	30.03
Drum	11.32	10.44	**24.10**	**5.37**	**2.40**	29.27
Mean	11.93	11.08	**29.21**	**8.26**	**4.69**	31.30

Table 3. **Effect of depth regularization on the pose estimation.** We report the mean relative rotation error (°) with and without depth regularization. The relative rotations are computed between the center camera and its neighbors within a mini-scene. The relative rotation error (*lower is better*) is defined as the rotation angle between the predicted relative rotations and the groundtruth.

2.2. Case study on Drums

In this section, we take a closer look at the performance of our model on Drums, the worst-performing scene in Table 1. The mean rotation error over the 100 cameras is 12.39° (see Table 1 of the main paper). Figure 1 shows the estimated camera poses juxtaposed with the ground truth cameras after Procrustes alignment. We can see that there is a small cluster of poorly posed images. Since Procrustes finds the optimal least-squares global alignment between predicted and true camera poses, it is severely affected by these outlier images. A subtle consequence of this is that the test time optimization, described in Sec. 1.2, may not be sufficient to evaluate the novel view synthesis results accurately and quantitatively. Due to the exaggerated misalignment from Procrustes in Drums, we may need to increase the number of iterations in order to converge to a more accurate viewpoint. Indeed, we find simply increasing the number of test-time optimization iterations from 100 to 1000 dramatically improves the rendering metrics: PSNR increases from 14.26 to 23.53, SSIM increases from 0.71 to 0.89, and LPIPS decreases from 0.30 to 0.12.

2.3. Effect of depth regularization

Similar to RegNeRF [9], we encourage the smoothness of the predicted depth maps and apply a depth regularization on small patches. We sample patches rendered from the cameras whose poses are jointly optimized with the 3D representation, while RegNeRF uses groundtruth poses for the observed views and samples the patches from unobserved viewpoints. We find that such depth regularization is crucial to the success of LU-NeRF. Table 3 shows that incorporating the depth regularization significantly improves the pose estimation accuracy of LU-NeRF – the median relative rotation errors decrease from 11.08° to 4.69° while the mean drops from 11.93° to 8.26°. Even though the maximum relative rotation error is smaller without the depth regularization, the Shonan averaging [5] fails to converge to a reasonable global pose configuration.

2.4. Computational cost

Table 4 presents the computational cost of the proposed framework. We randomly sample 30 mini-scenes and report the average training time for LU-NeRF-1 and LU-NeRF-2. The LU-NeRFs for different mini-scenes are independent

and thus can be trained in parallel. The training of LU-NeRFs and the global pose refinement with BARF [6] can be significantly accelerated with some recent advances in learnable scene representations (*e.g.* PlenOctrees [10], InstantNGP [8]).

Stage	Running time
LU-NeRF-1	1.08 hours
LU-NeRF-2	0.89 hours
Pose synchronization	3.18 seconds
Pose refinement	4.40 hours

Table 4. **Computational cost.** We report the mean time for training a single LU-NeRF-1, a single LU-NeRF-2, and the final refinement step on an NVIDIA Tesla P100. The pose synchronization step runs on CPU and has a negligible running time.

3. Implementation details

3.1. Building connected graphs

Given a distance function $\text{dist}(\cdot,\cdot)$, image descriptors $\{f_i\}^N$ and corresponding cameras $\{C_i\}^N$ where $i \in \{0,\dots,N-1\}$ is the image index and N is the total number of images, as the first step, we build a minimal spanning tree (MST) using Kruskal's algorithm. Each node on the MST represents a camera and the weight W_{ij} on the edge that connects camera C_i and C_j is the distance $\text{dist}(f_i, f_j)$ between the descriptors of image I_i and I_j. To ensure each mini-scene contains at least K images ($K = 5$ by default), we augment the MST by adding nearest neighbors for nodes that have less than $K-1$ connected nodes in the MST. We also ensure that each edge appears in both mini-scenes centered at the endpoints of the edge, such that there are at least two measurements for the relative pose between two connected cameras. Having multiple measurements allows for estimating the confidence of the predicted relative poses and identifying LU-NeRF failures (see Sec. 3.2).

3.1.1 Distance functions

Motivation. We intentionally experiment with simplistic approaches to compute image similarity in our graph-building procedure since our primary contribution in this work is the local-to-global pose and scene estimation starting with LU-NeRF on mini-scenes. In practice, depending on the application context, there are likely different cues or weak supervision that can be exploited for graph building (as we do for ordered sequences). We leave it to future work to explore more sophisticated unsupervised/self-supervised techniques for building neighborhood graphs.

In our experiments, we tried two different features to build the connected graph: self-supervised DINO features [4] and raw RGB values with ℓ_1 distance.

DINO features. We extract semantic object parts by applying K-means clustering on the image collections [2]. The number of parts is 10 by default in our experiments. We build an image descriptor in a similar way as the Histogram-of-Gradients (HoG). Specifically, we evenly split the part segmentation maps into 4×4 grid and compute a part histogram within each cell. We then normalize the histogram per cell into a unit vector and use the concatenated 16 histograms as the image descriptors. The cosine distance between these descriptors is used to build the MST and final graph.

ℓ_1 on RGB values. We estimate the distance between two images as the minimum ℓ_1 cross-correlation over a set of rotations and translations. Formally, we compute

$$\text{dist}(I_1, I_2) = \underset{t,\theta}{\text{argmin}} \sum_x \mid I_1(x) - I_2(R_\theta x + t) \mid, \quad (1)$$

where x is the pixel index, t is in a 5×5 window centered at 0, R_θ is an in-plane rotation by θ and $\theta \in \{0, \pi\}$. When the minimum is found at $\theta \neq 0$, we augment the input scene with the in-plane rotated image and correct its estimated pose by θ.

We found that considering the in-plane rotations here is useful because of symmetries – some scenes of symmetric objects can contain similar images from cameras separated by large in-plane rotations. This is problematic for LU-NeRFs because they initialize all poses at identity. Augmenting the scene with the closest in-plane rotations makes the target poses closer to identity and helps convergence.

Metric selection In experiments with unordered image collections, we used the ℓ_1/RGB metric for Lego and Drums, and the DINO metric for Chair, Hotdog, and Mic. The RGB metric fails to build useful graphs for Hotdog, Mic and Ship, while the DINO metric fails for Lego, Drums, and Ship. No graph-building step is necessary on ordered image collections since the order determines the graph.

3.1.2 Analysis

Figure 2 presents the MST, the connected graph, and image pairs that are connected in the graph on the Chair scene from the NeRF-synthetic dataset when using the DINO features. Surprisingly, the self-supervised ViT generalizes well on unseen objects and the learned representations are robust to viewpoint change (see the last column of Figure 2).

Figure 3 presents an analysis of the connected graphs built with DINO and RGB features. Both features provide outlier-free connected graphs on Chair. The graphs built with DINO contain much fewer outliers on Hotdog and Mic, while RGB features induce clearer graphs on Drums and Lego. Both DINO and RGB features produce more outliers on Ship than other scenes.

Optimal graph vs noisy graph. To analyze the effect of graph building on the unordered image collection, we build

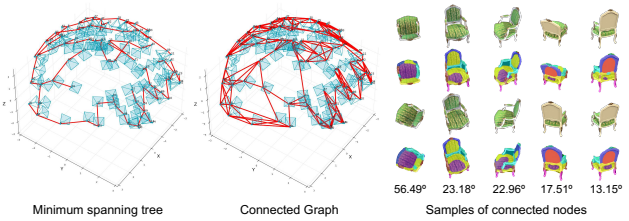

Figure 2. **Graph built with DINO features on Chair.** The minimum spanning tree (left), the connected graph (middle), and samples of connected image pairs (right). In the right panel, each column presents two images that are connected on the graph (1st and 3rd row), the corresponding part co-segmentation maps [2] (2nd and 4th row), and rotation distance between the two views (bottom).

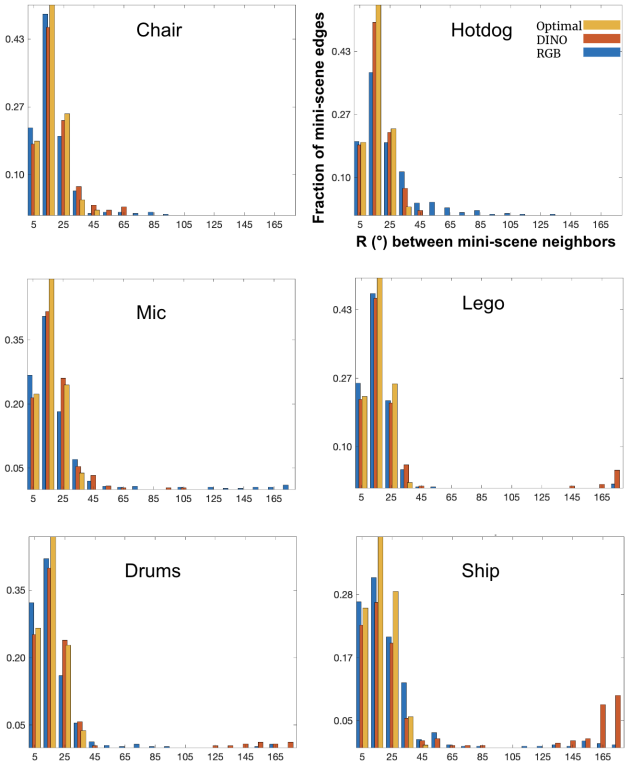

Figure 3. **Graph statistics.** We compare the rotation distance between mini-scene neighbors on the optimal graph built with groundtruth camera poses, the graph built with DINO features, and the one built with RGB features. For most scenes both DINO and RGB mini-scenes include outlier images (close to 180° distance) which our pipeline needs to deal with.

an *oracle* outlier-free connected graph with groundtruth camera poses. Table 5 compares the performance of our method with the optimal graphs and noisy graphs built with DINO/RGB features. Outliers in the connected graph may hurt the performance of LU-NeRF. Nevertheless, with our simple graph-building methods based on DINO/RGB features, our method outperforms the baselines when they are not given prior constraints on the camera pose distributions.

We notice that the performance with the optimal graph is worse than that with the noisy graph on Chair. The "optimal" graph minimizes camera distances, but it is not guaranteed to be the best choice for LU-NeRF. *e.g.*, issues like mirror symmetry ambiguity (Sec. **??**) can arise more often when cameras are in close proximity, and there is randomness inherent in training neural networks.

Scenes	Rotation Error [°]		Translation Error	
	Optimal graph	Noisy graph	Optimal graph	Noisy graph
Chair	4.24	2.64	0.16	0.09
Hotdog	0.23	0.24	0.01	0.01
Lego	0.07	0.09	0.00	0.00
Mic	0.84	6.68	0.02	0.10
Drums	0.05	12.39	0.00	0.23

Table 5. **Optimal graphs vs noisy graphs.** The outliers in the noisy connected graph built with DINO/RGB features may hurt the performance of our method in camera pose estimation. The clean graph is built from the ground-truth camera poses.

3.2. LU-NeRF architecture and training details

In the training of LU-NeRF, we do not apply the coarse-to-fine strategy proposed in BARF [6]; we sample 2048 rays per mini-batch and adopt the learning rate schedule for pose and MLP parameters from BARF; we remove the positional encoding and view-dependency, and use a compact 4-layer MLP to reduce the memory cost and speed up the training. We set the initial camera poses to identity. We have experimented with random initialization around identity but observed no significant difference. We terminate the training of LU-NeRF-1 if the average change of the camera rotations in a mini-scene is less than 0.125° within 5k iterations. We train LU-NeRF-2 for 5k iterations with frozen initial poses and then jointly optimize camera poses and neural fields for 45k iterations. We apply depth regularization on small patches (2×2 by default) in both LU-NeRF-1 and LU-NeRF-2.

3.3. Synchronization and refinement details

In the pose synchronization step, we apply the off-the-shelf Shonan averaging[2] [5], which solves a convex relaxation of the problem described in Eqn. (1) of the main text, while iteratively converting it to higher dimensional special-orthogonal spaces $\mathbf{SO}(n)$ until it converges. We then optimize the translation with fixed camera rotation.

The input to the Shonan averaging is the relative pose estimations from LU-NeRF and the confidence of these pose estimations. Each pair of cameras may have multiple measures of the relative poses, as each camera may appear in multiple mini-scenes. We apply a simple heuristic to pick one measure from these multiple candidates: given two

[2] https://github.com/dellaert/ShonanAveraging

	Bike	Chair	Cup	Laptop	Shoe	Book
Rotation	24.41	5.61	13.41	23.13	19.36	45.78
Translation	4.09	1.20	0.63	1.79	0.82	1.43

Table 6. **Camera baselines.** We report the average rotation [°] and translation distance between all camera pairs in the sequential data sampled from the Objectron datatset [1].

cameras C_i and C_j and their renderings I_i and I_j, where $i < j$, if the PNSR of I_i in the mini-scene centered at C_i is higher than the PSNR of I_i' in the mini-scene centered at C_j, we use the pose estimation from the mini-scene i as our relative pose estimation between camera C_i and camera C_j.

To resolve the scale ambiguity across different mini-scenes, we first scale each mini-scene so that the MST edges connecting different mini-scenes are scale-consistent (MST construction is described in Sec. 3.1). Specifically, we establish a reference scale by fixing it in one mini-scene and propagating it to others through the MST. We focus on edges linking mini-scene centers and rescale the mine-scenes so that overlapping edges share a consistent scale.

We then obtain the translation by solving a linear system $t_j - t_i = R_i t_{ij}$ where R_i is the rotation of camera i from the Shonan method and t_{ij} is the relative translation between camera i and j from LU-NeRF. Similar to the relative rotations, each pair of cameras may have multiple measures of the relative translation. We use the same heuristic described above to pick one from the multiple measures. The translation optimization has also been implemented in the off-the-shelf Shonan averaging.

In the global pose refinement stage, we closely follow the default setting of BARF [6] which jointly trains the MLP and refines the poses for 200k iterations with a coarse-to-fine positional encoding strategy.

We utilize the camera visualization toolkit from BARF [6] in our main paper and the supplementary material.

3.4. Dataset release and open sourcing.

We will release the newly ordered Blender sequences and open-source the code on our project website[3].

For the sequential data sampled from the Objectron dataset [1], Table 6 reports the average rotation and translation distance between all camera pairs as a reference for our quantitative evaluations in Table 6 in the main text.

References

[1] Adel Ahmadyan, Liangkai Zhang, Artsiom Ablavatski, Jianing Wei, and Matthias Grundmann. Objectron: A large scale dataset of object-centric videos in the wild with pose annotations. In *CVPR*, 2021. 5

[2] Shir Amir, Yossi Gandelsman, Shai Bagon, and Tali Dekel. Deep vit features as dense visual descriptors. *arXiv preprint arXiv:2112.05814*, 2021. 3, 4

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*. PMLR, 2017. 1

[4] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021. 3

[5] Frank Dellaert, David M. Rosen, Jing Wu, Robert Mahony, and Luca Carlone. Shonan rotation averaging: Global optimality by surfing $so(p)^n$. In *ECCV*, 2020. 2, 4

[6] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. BARF: Bundle-adjusting neural radiance fields. In *ECCV*, 2022. 1, 3, 4, 5

[7] Quan Meng, Anpei Chen, Haimin Luo, Minye Wu, Hao Su, Lan Xu, Xuming He, and Jingyi Yu. GNeRF: GAN-based Neural Radiance Field without Posed Camera. In *ICCV*, 2021. 1

[8] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM TOG*, 2022. 3

[9] 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 *CVPR*, 2022. 2

[10] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021. 3

[11] Jiahui Zhang, Fangneng Zhan, Rongliang Wu, Yingchen Yu, Wenqing Zhang, Bai Song, Xiaoqin Zhang, and Shijian Lu. Vmrf: View matching neural radiance fields. In *ACM MM*, 2022. 1

[3] https://zezhoucheng.github.io/lu-nerf/