NeRF4SeRe: Neural Radiance Fields for Scene Reconstruction

Wei Jiang* Kangrui Mao* Gu Zhang* Haoyu Zhen*

Abstract

*Neural implicit representations have shown encouraging results in various domains, including promising progress in **Scene Reconstruction** (SeRe). In this project, we implement many off-the-shelf architectures of **Neural Radiance Field** (NeRF). Also, for the tasks where the camera poses are noised, we introduce and implement some modules to optimize and refine them during training and testing. Our code is available at* https://github.com/nerf-sere/SeRe.

1. Introduction

Reconstructing and rendering large-scale indoor scenes from RGB images is challenging but crucial for various applications in computer vision and graphics, including AR, VR, e-commerce, and robotics. Our contributions are as follows:

1. We implement some off-the-shelf models and architectures: NeRFusion[1], INeRF, NICE-SLAM, vannila NeRF and Instant-NGP. Some of them are migrated to instant-ngp pipeline. We reproduce NeRFusion independently. For the other methods, we also implement the dataloader for loading Replica Dataset.
2. We introduce NNeRF to tackle the problem that the camera poses with noises.
3. Moreover, we write a `backward` function for ray-marching, so the gradient from `xyzs` and `dirs` can propagate to `rays_o` and `rays_d`, which is an extension for Torch-NGP project.

2. Related Work

Scene Reconstruction. The problem is defined below. Let S be the scene to be reconstructed. We train our neural networks to approximate two functions f and g such that

$$f[(\boldsymbol{x}_0, \boldsymbol{p}_0), (\boldsymbol{x}_1, \boldsymbol{p}_1), \cdots, (\boldsymbol{x}_N, \boldsymbol{p}_N)] = \boldsymbol{S} \qquad (1)$$

*These authors contributed equally to this work. The authors are listed in alphabetical order of last name.

[1]Official repository is still cleaning. The authors haven't released runnable (neither test nor train) code for the critical fusion modules till the moment we finished this report.

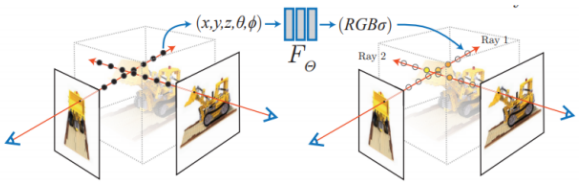

Figure 1. NeRF Architecture

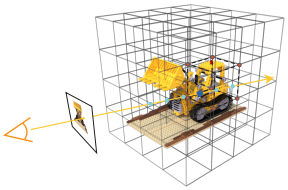

Figure 2. Ray Marching in Grid

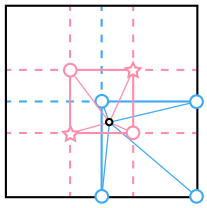

Figure 3. Hash Grid

and

$$g[\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_N, \boldsymbol{p}_0] = [\boldsymbol{S}, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_N] \qquad (2)$$

where $\boldsymbol{x}_i$ and $\boldsymbol{p}_i$ are RGB image and camera pose in a sequence of N images.

There are many different methods and problem settings for 3D scene reconstruction. Along with the development, people's research focus turns from single-image reconstruction to multiple-image reconstruction.

The common 3D representation for single image reconstruction includes voxel representation, point cloud representation, and mesh representation. **CoReNet** [7] jointly reconstructs multiple objects from a single image via a coherent reconstruction network. **Realpoint3d** [12] retrieve the nearest 3D shape as an extra input of the reconstruction network to generate a fine-grained point cloud. **GeoPIFu** [1] uses a deep implicit function to represent clothed body shapes. It preserves global shape regularity as well as details of clothes by aligning and fusing local geometry and pixel features.

The multi-view reconstruction mainly includes two parts: rigid reconstruction and non-rigid reconstruction. As to the first one, **Pix2Vox++** [13] base on RNNs to generate a coarse volume for each input image. The method fuses all the coarse volumes through a multiscale context-aware

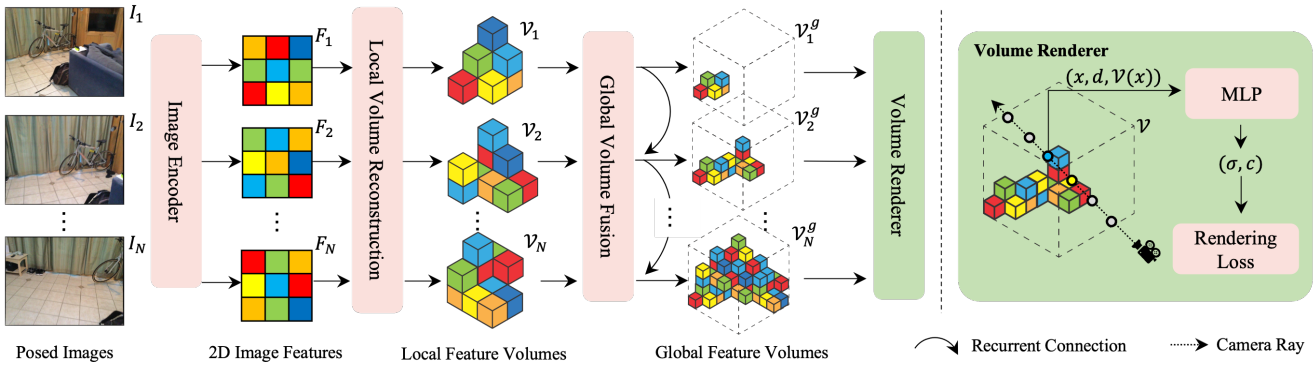

Figure 4. NeRFusion Pipeline. Picture adopted from [17].

fusion module and adopt a refiner to correct the fused volume. As for non-rigid reconstruction, [19] utilize the SMPL model to build up a human avatar from a sparse RGBD video. They perform an initial pairwise alignment over every adjacent two frames of the video and generate a full 3D shape through a global non-rigid registration procedure.

Besides all above, neural implicit representations have gained significant advances. There are also many methods based on it to solve the scene reconstruction problem, which will talk later in this section.

In our work, We will use **Replica** [9] as the dataset for scene reconstruction. The Replica Dataset is a dataset of high-quality reconstructions of a variety of indoor spaces. Although there is abundant information, like depth, normal, segmentation, and texture, we only use RGB images and camera pose to reconstruct the scene.

Neural Radiance Field (NeRF). In recent days, neural implicit representations are receiving increasing attention from researchers. Neural Radiance Field (NeRF) is an optical representation of a scene. An MLP is used to approximate the radiance field function.

As shown in Fig. 1, a NeRF takes in spatial location $\boldsymbol{x} = (x, y, z)$ and viewing direction $\boldsymbol{d} = (\theta, \phi)$, and outputs the RGB color and volumic density σ of the scene at x from d. Images are synthesized with volumic rendering.

Based on NeRF, there are many nerf-based model variants, such as NeRF++, DNeRF, pixelNeRF, etc. **NeRF++** [16] addresses the parametrization issue involved in applying NeRF to 360 captures of objects within large-scale, unbounded 3D scenes and improves view synthesis fidelity in this challenging scenario. **DNeRF** [2] extends neural radiance fields to a dynamic domain, allowing to reconstruct and render novel images of objects under rigid and non-rigid motions from a single camera moving around the scene. **pixelNeRF** [15] is a learning framework that predicts a continuous neural scene representation conditioned on one or a few input images. It mainly resolves the problem that NeRF optimizes the representation of every scene independently, which requires many calibrated views and significant compute time. There are also new models developed for accelerating the training process. Among them, the recent famous work is Instant-NGP, which we will talk about later.

3. Method

We reconstruct the scene under the NeRF paradigm. Using instant-ngp (Sec. 3.2) as our baseline and backbone, we implement NeRFusion architecture (Sec. 3.3) and NICE-SLAM model (Sec. 3.4). For the tasks in which camera poses are of noises (Sec. 3.5), we provide INeRF and introduce NNeRF to refine the poses during inferring and training.

3.1. Preliminaries: NeRF

We first describe the differentiable volume rendering method used by NeRF [3]. Rather than a traditional rasterization process, volume rendering is more like ray tracing. For each pixel on the image plane, we cast a ray $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$ constrained by pre-defined near and far bounds t_n and t_f. Then we can accumulate radiance along the ray according to volume density and get pixel color

$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t), \boldsymbol{d})\mathrm{d}t, \tag{3}$$

where

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\boldsymbol{r}(s))\mathrm{d}s\right). \tag{4}$$

The function $T(t)$ stands for the accumulated transmittance along the ray from t_n to t, i.e., the probability that the ray travels from t_n to t without getting absorbed by other particles. As shown in Fig. 1, we actually sample several points on the ray with some given strategies. This serves as a discrete approximation, which gives the expectation of pixel

color

$$\hat{C}(\boldsymbol{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i \delta_i))\mathbf{c}_i. \tag{5}$$

And

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right), \tag{6}$$

where $\delta_i = t_{i+1} - t_i$ is the distance between adjacent samples.

3.2. Instant-NGP

Despite its high compressibility, NeRF is optimized globally. That is to say, any optimization step would somehow change every single parameter of the MLPs. This could lead to memory loss in long training schemes. Besides, although the MLPs are shallow, large numbers of queries would still lead to terrible rendering speed and tremendous memory consumption. One solution is to substitute a large proportion MLPs with explicit representations and leave extremely small MLPs as output layers. Take voxel grids as an example, we query the grid directly with interpolation as is illustrated in Fig. 2 and give some transforms to get color and density. Thanks to its explicity and locality, only seen voxels would be updated in backward steps. In addition, querying become way faster, and gradient propagations across multiple layers are no longer required.

Hash grids. Given the sparsity of 3D voxel grids, it is natural to utilize hash tables to represent such data structures. This could further compress the space usage and speed up querying as well. For these benefits, Instant-NGP [4] proposed a special type of hash grid as a universal representation for spacial tasks including NeRF, TSDF, etc. An inevitable topic about hash tables is hash collisions. When two different queries point to an identical value, things get tricky. There is a naïve solution to extend the hash table and solve collisions.

Multiple resolution. When it comes to more filled voxels, the hash grid would collapse to a normal grid with a redundant hash function. Instant-NGP solves collisions in an impressive way. Instead of busy solving collisions, Instant-NGP lets collisions as they are, and constructs multi-resolution hash tables to make this up. When the grid gets queried, features from all resolutions will be concatenated together as result. As is shown in Fig. 3, the higher the resolution grows, the more parameters there will be in the grid, and therefore more hash collisions (stars) follow. However, there exist no hash collisions in lower resolution, so the output features would differ. As is proved in [4], hash collisions are scattered almost uniformly in the space for Instant-NGP, so we would hardly expect a "true" collision to happen for Instant-NGP. To explain its effectiveness, we have an analogy with Fourier frequencies. Lower resolutions, just like lower frequencies, give base features for a point, while higher resolutions would give more various details which get repeated more frequently across positions.

3.3. NeRFusion

In the vanilla NeRF [3] pipeline, we cast rays on the scene repeatedly and minimize the difference between renderings and ground truth images. Though magically lifting 2D images to 3D models, the paradigm requires tedious per-scene optimization to obtain results. Inspired by the voxel grid representations and pixel-aligned methods by [8], we propose a method to reconstruct a NeRF with a feed-forward network, in which image features are unprojected onto 3D voxels, properly fused, and then re-rendered to synthesize novel views. But actually, the topic has already been researched by NeRFusion [17], which, just as is named, aims to forward fuse images into a NeRF. Even more, there also exist other similar works [11, 15] achieving feed-forward NeRF construction with image-based rendering approaches. Among all these methods, NeRFusion stands out for its reconstruction of a NeRF representation from an arbitrary number of input images once and for all. With detachable fusion modules, it decouples inference and rendering with long-lasting voxel grids as representations of whole scenes. Furthermore, with the fancy global fusion module with memory, it is also possible to refine the model with supplementary images once available. Thanks to the decoupled inference and rendering, the scene representations could be even further finetuned with the classic NeRF training pipeline. The overall architecture of NeRFusion is shown in Fig. 4.

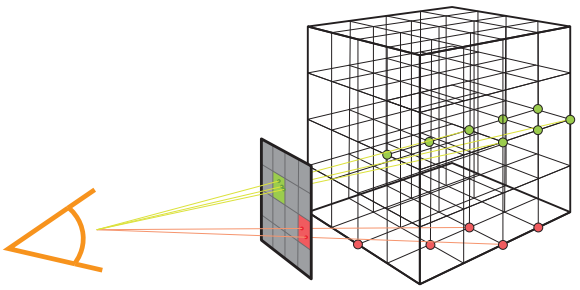

Figure 5. Back projection.

Image encoder. A pre-trained 2d backbone E (MnasNet 1.0 [10] in our implementation) serves as the image encoder in NeRFusion. From a continuous sequence of input images (e.g., a video clip) $\{I_1, I_2, \ldots, I_N\}$, it gets us the corresponding feature maps $\{F_1, F_2, \ldots, F_N\}$, where $F_n = E(I_n), n \in [1, N]$.

Local fusion. With 2D features ready, we then unproject them back into the 3D space. Since projection usually

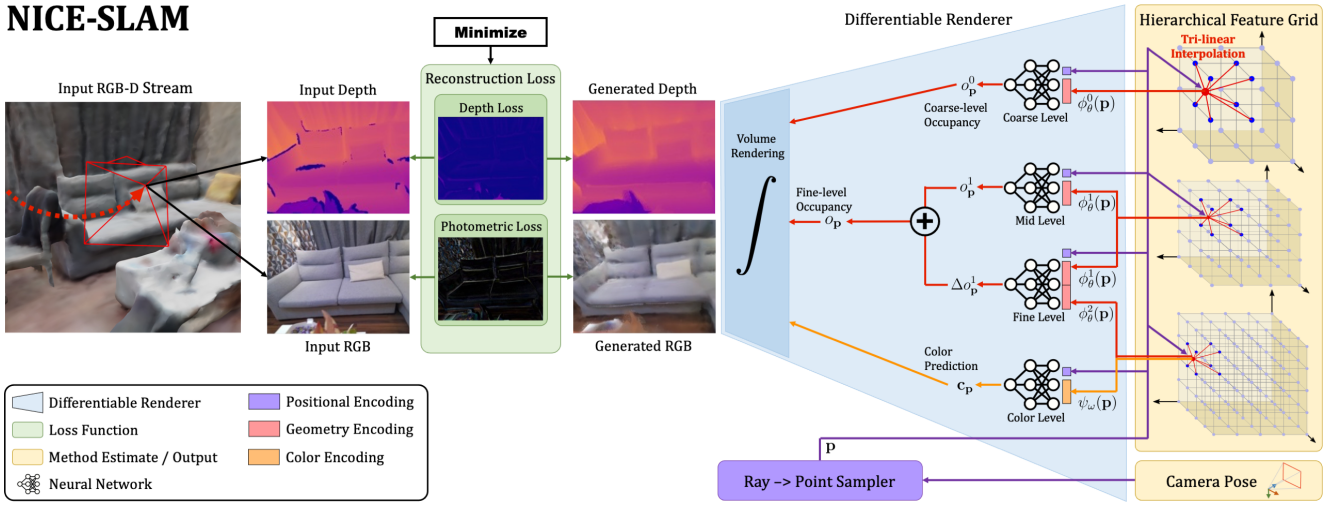

Figure 6. NICE-SLAM Architecture

refers to the transformation from the world (model) coordinates to viewport coordinates, we naturally name the procedure back projection. As is shown in Fig. 5, back projection is actually implemented by "forward" projection. Consider an empty grid $\mathcal{V}_n \in R^3 \times F$ with coordinates $\{p_{n,1}, p_{n,2}, \ldots, p_{n,r_n^3}\}, p_{n,i} \in R^3$, where r_n is resolution of $\mathcal{V}_n$. We project every voxel point $p_{n,i}$ onto the image plane to fetch an interpolated feature vector $\mathcal{V}_n(p_{n,i}) \leftarrow f_{n,i}$ from the feature map, which can also be seen as a back projection from 2D feature points to their corresponding voxels. Besides an information highway from image to voxel, the back projection also helps cull unseen voxels for better sparsity. Later, the voxel is projected to aligned camera coordinates and passed through a 3D CNN, which gives the final local volume $\mathcal{V}_n$. Note that in implementation, three continuous frames are fused together as one local volume, forming an easy multi-view stereo.

Global fusion. All these local volumes are fed into a GRU-based global fusion module, it memorized intermediate states while outputting fusion of all input local reconstructions. This step repeatedly merges local frames, making them compact and available for later use, which is critical for once-for-all fusion.

Rendering. With the global feature volume, we can cast rays as in Instant-NGP [4]. Both RGB and depth maps are available with volume rendering. Specially, we do not have a proper renderer for sparse voxel, so we make an algorithm to convert it into an NGP hash grid with the identical resolution, which would be further discussed in Appendix A.1. And further, the inferenced voxel grid (or the converted NGP grid) can be finetuned in the classic NeRF pipeline with rendering loss.

Training scheme. For better stability, we follow [17] to adopt a staged training scheme. In the first stage, we train the local fusion module for several epochs, the fused local volumes are directly rendered into images and optimized with rendering loss. For the second stage, the global fusion is also included, the whole model is trained in an end-to-end manner as is illustrated in Fig. 4. As the final stage, the model would be finetuned on a specific scene on demand.

3.4. NICE-SLAM

NICE-SLAM [18] takes an **RGB-D** image stream as input and outputs both camera pose as well as a learned scene representation in form of a hierarchical feature grid. [18] proposes to use a differentiable rendering process which integrates colors and occupancy from scene reconstruction. The architecture is shown in Fig. 6. However, NICE-SLAM is not our main architecture by the fact that the goal is to reconstruct the meshes from RGB images. Thus we will only analyze the results compared to that of other NeRF paradigms.

3.5. Denoise Module

Real-world datasets may not provide accurate camera pose. Please consider the following two situations. (1)If you have trained a model for one scene, when you want to test the model on a new picture in the scene, you may get an inaccurate camera pose. So the rendering result of a new picture will be bad. (2)The second is a worse one. The camera pose in your training data is not accurate, so the rendering model will learn in the wrong way, which will fail the model.

INeRF [14]. In the first situation, We propose a module to refine the camera pose and further predict the unknown pose from the new testing picture. The pipeline is shown in Fig.7

The weight of our NeRF model is frozen. So we try to

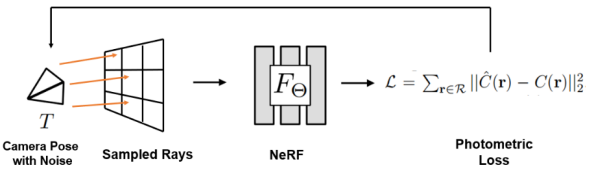

Figure 7. Pipeline to refine the camera pose when testing.

estimate the photometric loss w.t. camera poses in Eq. 7.

$$\hat{T} = \underset{T\in \mathrm{SE}(3)}{\operatorname{argmin}} \mathcal{L}(T \mid I, W) \tag{7}$$

where W is the weights of NeRF model, T is the camera pose.

This idea came to us independently when solving the problem. However when we use gradient descent to optimize the pose. Directly optimizing the pose is not effective. So we followed the solution in INeRF. We can represent the estimated camera pose $\hat{T}_i$ [6] as :

$$\hat{T}_i = e^{[\mathcal{S}_i]\theta_i} \hat{T}_0$$

where the $e^{[\mathcal{S}_i]\theta_i}$ is represented as Eq. 8.

$$e^{[\mathcal{S}]\theta} = \begin{bmatrix} e^{[\omega]\theta} & K(\mathcal{S},\theta) \\ 0 & 1 \end{bmatrix} \tag{8}$$

As the training process in forward NeRF, we should sample the rays to calculate the loss to decrease the complexity. The sampling strategy includes:

- Random Sampling
- Interest Point Sampling
- Interest Region Sampling

The interest region sampling strategy is based on the interest point sampling strategy. The strategy will sample from the dilated masks centered on the interest points. It can prevent the local minima caused by only sampling from interest points.

NNeRF. Let's go back to the second situation we mentioned at the beginning of the section. Under this condition, the solution way is to use backpropagation to optimize the model and refine the camera pose Simultaneously. However, this method can't work well. Inspired by NICE-SLAM [18], we propose an optimization method which we call it NNeRF to solve the problem.

The one iteration in the optimization consists of two steps. In the first step, we freeze the camera pose and optimize the model's weight. In the second step, we freeze the model weights and only optimize the camera pose. The optimal points follow different distributions for camera pose the model weights. So the iteration optimization will fix one's position and optimizes the other to the local optimum. By iteration, both of them will try to approach the global optimum.

4. Experiments

We evaluate our model on Replica [9] Datasets. In specific, we use the renderings from [18] for fair comparison. We select the metrics (shown in Tab. 1) proposed from [5] to estimate the performance of the reconstructed meshes. We also compute the Chamfer Distance which reads:

Metrics	Form
Accuracy	$\mathrm{mean}_{x\in X}(\min_{y\in Y} \|x-y\|)$
Comp	$\mathrm{mean}_{y\in Y}(\min x \in X \|x-y\|)$
Precision	$\mathrm{mean}_{x\in X}(\min y \in Y \|x-y\| < 0.05)$
Recall	$\mathrm{mean}_{y\in Y}(\min x \in X \|x-y\| < 0.05)$
F-score	$2 \times \mathrm{Prec} \times \mathrm{Recall}/(\mathrm{Prec} + \mathrm{Recall})$
Dist1	$\sum_{x\in X} \min_{y\in Y} \|x-y\|^2$
Dist2	$\sum_{y\in Y} \min_{x\in X} \|x-y\|^2$

Table 1. 3D metrics in Atlas

$$\mathcal{CD}(X,Y) = \sum_{x\in X} \min_{y\in Y} \|x-y\|^2 + \sum_{y\in Y} \min_{x\in X} \|x-y\|^2$$

where X is the predicted point cloud and Y is the GT. We sample 30000 points uniformly from the meshes to generate the point cloud.

As for image synthesis, we compute the PSNR (higher is better) and LPIPS (lower is better) of generated images.

We also test the results of denoised module. We denote the ground truth of camera pose is $\hat{T}$, our estimated pose is T. They can be represented as $T = [R|t]$. We calculated the translation error and rotation error as Eq. 9.

$$\begin{aligned} d_t(\mathbf{T}_1, \mathbf{T}_2) &= \|\mathbf{t}_1 - \mathbf{t}_2\| \\ d_\alpha(\mathbf{T}_1, \mathbf{T}_2) &= \arccos \frac{1}{2}\left(\mathrm{tr}\left(R_1 \cdot R_2^{\mathrm{T}}\right) - 1\right) \end{aligned} \tag{9}$$

4.1. Qualitative Results

We show our qualitative Results on Replica Dataset under different methods using different representation methods, like image, gif and interactive mesh. Because we can't show all the results in this report, so for more information, please refer to our presentation slides https://sere-cv-project.netlify.app/. (Loading the contents may **take some time because of online rendering**. Please wait for it.) In it, we show the constructed different scenes using gifs. We also show the mesh of Instant-NGP and Nice-Slam, which are interactive. You can drag the mouse to zoom in and zoom out, and change the viewing angle. Besides that, we also show the reconstruction results under the dymanic process from the noised initial camera pose to refined pose using our denoised module.

NeRF The rendering result of the vanilla NeRF on Replica is shown in Fig. 8.

Figure 8. NeRF Rendering

Intant-NGP Here we show some reconstruction results of different scenes in Fig. 9 with Instant-NGP. For more different scenes and views, please refer to our slides.

Figure 9. Instant-NGP Renderings

NICE-SLAM We show the results of NICE-SLAM using rendered mesh. Fig. 10 shows one view of the mesh. For more views and interactive results, please refer to our slides.

Figure 10. Colored Mesh Reconstructed with NICE-SLAM

NeRFusion In Fig. 11 we display renderings from fused voxels produced by NeRFusion. Global fusion results are more blurred than local ones. This is because local voxels are fused from near views, the model can easily cheat the renderer by providing a "flat" result. And as is shown in the images, the depth reconstructions from global voxels are way better.

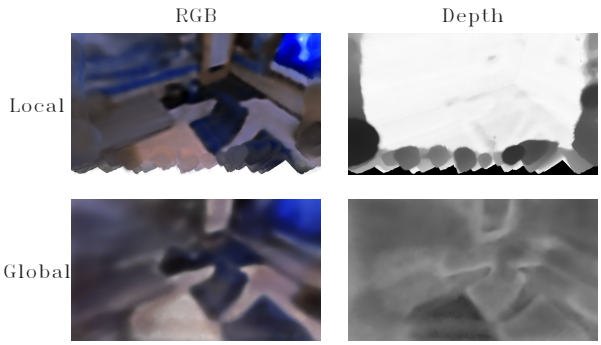

Figure 11. Renderings from NeRFusion

Denoised Module Fig.12 shows the rendering picture under initial noised camera pose and refined camera pose. We add a ground truth background to better visualize the dynamic change. We can see the denoised picture look nearly the same as Fig. 8, which verifies the effect of our module. In slides we have the gif to show the whole denoising process.

Figure 12. Noised and Denoised Results

4.2. Quantitative Results

Mesh Results Here we show the qualitative Results for reconstructed mesh in Tab. 2.

Metrics	Instant-NGP	Nice-Slam
Dist1	0.0264	0.0077
Dist2	0.0818	0.0085
Chamfer-Dist	0.2144	0.0162
Precision	0.9110	0.9867
Recall	0.8470	0.9835
F-Score	0.8778	0.9851

Table 2. Mesh Metrics

NeRF Results For NeRF-like methods the rendered RGB images are also evaluated along with reconstruction

time, which are shown in Tab. 3. Note that we implement NeRFusion with rather low grid resolution ($96 \times 96 \times 96$) than Instant-NGP (16 levels, up to $4096 \times 4096 \times 4096$). Furthermore, the 2D backbone gives downsampled feature map, and the model is trained for only 2 epochs each stage. The lower statics are acceptable. However, its results are not totally better than the Instant-NGP variant with same grid resolution. We could attribute this to the unsearched hyperparameters. Also, this could be explained as a trade-off between quality and generalizability as NeRFusion will generalize to any new scenes once trained. As is reported in [17], NeRFusion can also reach PSNR as high as 26. We are still investigating into this issue.

Denoised Results We add random noise to the ground truth pose. Here we show the rotation error and translation error between the camera pose and ground truth in Tab. 4. The sampling strategy is interest region sampling.

5. Supplementary and Future Works

In our implementation of NeRFusion [17], adjacent images are fused together to form a local volume. It is worth trying to larger the frame gaps between images in one local volume, so as to achieve more 3D-aware local fusion model. NeRfusion also allows per-scene finetuning after fusion done. We have implemented this while there is no time left for experiments. Once the experiments are finished, results would be appended to the report. For simplicity, we do not implement NeRFusion with as high grid resolutions as Instant-NGP [4]. Our grid conversion support girds with any resolution. And multiple resolutions would get supported with tiny modifications, for which we will give a proof-of-concept algorithm in appendencies. More experiments on various resolutions would be done later. And also, we have proposed a grid scaling approach based on residual links in our presentation. It is crucial to convert one grid into an multi-resolution one. The effectiveness would be discussed in future updates.

Our proposed NNeRF is based on Instant-NGP. However, the pytorch implementation Torch-NGP doesn't support the backward function for raymarching, so the gradient from `xyzs` and `dirs` can't propagate to `rays_o` and `rays_d`, which means the photometric loss can't backpropagate to camera pose. To solve the problem, we write a backward function using CUDA & Python for raymarching on our own, which is **an extension for Torch-NGP**. The implementation details is shown in Appendix A.2. We will pull a request to the code repo later. However, learning CUDA is difficult for us at first. So after the implementation for backward function, we have no enough time to fully implement NNeRF. We will finish it and integrate it to our project. Please look forward to it.

You could check out all our future updates in report (including appendices) and code on our code repo `https://github.com/nerf-sere/SeRe`.

References

[1] Tong He, John Collomosse, Hailin Jin, and Stefano Soatto. Geo-pifu: Geometry and pixel aligned implicit functions for single-view human reconstruction, 2020. 1

[2] 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, 2020. 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. *Communications of the ACM*, 65(1):99–106, 2021. 2, 3, 9

[4] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022. 3, 4, 7, 9

[5] Zak Murez, Tarrence Van As, James Bartolozzi, Ayan Sinha, Vijay Badrinarayanan, and Andrew Rabinovich. Atlas: End-to-end 3d scene reconstruction from posed images. In *European conference on computer vision*, pages 414–431. Springer, 2020. 5

[6] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 9

[7] Stefan Popov, Pablo Bauszat, and Vittorio Ferrari. Corenet: Coherent 3d scene reconstruction from a single rgb image, 2020. 1

[8] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2304–2314, 2019. 3

[9] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. 2, 5

[10] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 3

[11] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P Srinivasan, Howard Zhou, Jonathan T Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4690–4699, 2021. 3

[12] Yan Xia, Yang Zhang, Dingfu Zhou, Xinyu Huang, Cheng Wang, and Ruigang Yang. Realpoint3d: Point cloud generation from a single image with complex background, 2018. 1

Metrics	NeRF-200k	Instant-NGP-30k	Instant-NGP-LowRes-30k	NeRFusion-local	NeRFusion-full
PSNR↑	30.2894	10.744995	28.8238	12.6633	14.9708
LPIPS↑	/	0.703447	0.2142	0.6215	0.6250
Recon Time	∼7h	∼11min	∼8min	∼0.5s (per image)	∼15min

Table 3. NeRF Metrics

Pose	Rotation Error(°)	Translation Error
Initial	6.823	8.327
Denoised	0.834	0.003

Table 4. Denoised Results on scene-Room0

[13] Haozhe Xie, Hongxun Yao, Shengping Zhang, Shangchen Zhou, and Wenxiu Sun. Pix2vox: Multi-scale context-aware 3d object reconstruction from single and multiple images. *International Journal of Computer Vision*, 128(12):2919–2935, jul 2020. 1

[14] Lin Yen-Chen, Pete Florence, Jonathan T Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin. inerf: Inverting neural radiance fields for pose estimation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1323–1330. IEEE, 2021. 4

[15] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, 2021. 2, 3

[16] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields, 2020. 2

[17] Xiaoshuai Zhang, Sai Bi, Kalyan Sunkavalli, Hao Su, and Zexiang Xu. Nerfusion: Fusing radiance fields for large-scale scene reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5449–5458, 2022. 2, 3, 4, 7

[18] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12786–12796, 2022. 4, 5

[19] Xinxin Zuo, Sen Wang, Jiangbin Zheng, Weiwei Yu, Minglun Gong, Ruigang Yang, and Li Cheng. SparseFusion: Dynamic human avatar modeling from sparse RGBD images. *IEEE Transactions on Multimedia*, 23:1617–1629, 2021. 2

Appendices

A. Implementation Details

A.1. Voxel to Hash Grid

When we convert a voxel grid to hash grid, the main challenge is hash collisions. In normal Instant-NGP [4] scenario, two colliding points means that they are not spatially close while they yield features from the same location in the hash table when queried with. At one certain feature g_m in the hash table, let the all colliding points on it be $P = \{p_1, p_2, \ldots, p_L\}$. We weight all these points by learning importance. In fact, the influence of p_l on g_m has already been weighted with the gradient during back propagation. So, reweighting is not necessary and we shall treat every p_l evenly. As a result, for estimating g_m from P, we choose to average all points in P.

The algorithm. Here we give an algorithm to convert a standard voxel grid into an Instant-NGP-compatible hash grid. A voxel grid can be represented in sparse form $\{C, F\}$, where $C = \{c_1, c_2, \ldots, c_K\}$ is the set of valid coordinates and $F = \{f_1, f_2, \ldots, f_K\}$ are corresponding features. Given a voxel grid with features $\mathcal{V} = \{C, F\}$ and an empty hash table $\mathcal{T} = \{g_1, g_2, \ldots, g_M\}$, in which g_m are hash grid features, we transfer the values through the following steps. First, we assign each hash grid feature point its ordinal position in flatten sequence. In this step, we use $\sqrt[p]{M}$-ary number system encoding (in pratice, we choose $p = 4$) to avoid data overflow especially for half precision environment. Then we query the hash grid with C to get the corresponding serial indexes I. With I, each K feature f_k in $\mathcal{V}$ is scattered to its corresponding g_{i_k}, and mean reduction is applied to produce the final g_{i_k}. This process is also decribed in Alg. 1.

Algorithm 1 Voxel to Hash Grid.

Input: $\mathcal{V} = \{C, F\}$, an voxel grid with features; $\mathcal{T} = \{g_1, g_2, \ldots, g_M\}$, an empty hash grid
Output: $\mathcal{T}'$, the hash grid filled with features from $\mathcal{V}$

$M \leftarrow |\mathcal{T}|$
$R \leftarrow [1, 2, \ldots, \sqrt[p]{M}]$
$\mathcal{T}[:p] \leftarrow \texttt{MeshGrid}(R, R, R, R)$
$I \leftarrow \mathcal{T}(C)$
$\mathcal{T}' \leftarrow \texttt{ScatterMean}(I, F)$
return $\mathcal{T}'$

A.2. Backward for Ray Marching

Ray marching is implemented with CUDA in Instant-NGP for better performance. However, since classic NeRF [3] optimization does not require gradients in this process, the compute graph truncates here, preventing us from optimizing poses.

Mathematical analysis. Suppose we cast a bunch of rays $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots$, where $\boldsymbol{r}_r = (\boldsymbol{o}_r, \boldsymbol{d}_r)$. And for each ray $\boldsymbol{r}_r$, we sample a sequence $t_{r1}, t_{r2}, \ldots$, resulting in position and direction sequences $\boldsymbol{x}_{r1}, \boldsymbol{x}_{r2}, \ldots$ and $\boldsymbol{d}_{r1}, \boldsymbol{d}_{r2}, \ldots$, where $\boldsymbol{x}_{ri} = \boldsymbol{r}_r(t_{ri}) = \boldsymbol{o}_r + t_{ri}\boldsymbol{d}_r$, $\boldsymbol{d}_{ri} = \boldsymbol{d}_r$. Then we consider the moment when gradients for each $\boldsymbol{x}_{ri}$ and $\boldsymbol{d}_{ri}$ are ready. We calculate gradients for $\boldsymbol{o}_r$ and $\boldsymbol{d}_r$ through

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{o}_r} = \sum_i \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{ri}} \cdot \frac{\partial \boldsymbol{x}_{ri}}{\partial \boldsymbol{o}_r} = \sum_i \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{ri}}, \tag{10}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{d}_r} &= \sum_i \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{ri}} \cdot \frac{\partial \boldsymbol{x}_{ri}}{\partial \boldsymbol{d}_r} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{d}_{ri}} \cdot \frac{\partial \boldsymbol{d}_{ri}}{\partial \boldsymbol{d}_r} \right) \\ &= \sum_i \left(t_{ri} \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{ri}} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{d}_{ri}} \right), \end{aligned} \tag{11}$$

where $\mathcal{L}$ is the objective function.

Implementation. From Eq. 10, 11, to smoothly calculate $\frac{\partial \mathcal{L}}{\partial \boldsymbol{o}_r}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{d}_r}$, we need for every r and i

- $\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{ri}}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{d}_{ri}}$, which is maintained by PyTorch [6];
- value of $t_r i$;
- the correspondence between from the sampled point $(\boldsymbol{x}_{ri}, \boldsymbol{d}_{ri})$ to the ray it belongs to.

For both $t_r i$ and the correspondence , we modify the CUDA kernel to record t_{ri} and $r_{ri} \triangleq r$ synchronized with the generation of $\boldsymbol{o}_{ri}$ and $\boldsymbol{d}_{ri}$. Finally, the gradients will be calculated in parallel for each r and i and then accumulated to their corresponding $\frac{\partial \mathcal{L}}{\partial \boldsymbol{o}_r}$, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{d}_r}$ with scatter sum according to r_{ri}.