F2-NeRF: Fast Neural Radiance Field Training with Free Camera Trajectories

By: Mustafa Khan

Abstract

This paper presents a novel grid-based NeRF called ***F^2-NeRF (Fast-Free-NeRF)*** *for novel view synthesis, which enables arbitrary input camera trajectories and only costs a few minutes for training. Existing fast grid-based NeRF training frameworks, like Instant-NGP, Plenoxels, DVGO, or TensoRF, are mainly designed for bounded scenes and rely on space warping to handle unbounded scenes. Existing two widely-used space-warping methods are only designed for the forward-facing trajectory or the 360° object-centric trajectory but cannot process arbitrary trajectories. In this paper, we delve deep into the mechanism of space warping to handle unbounded scenes. Based on our analysis, we further propose a novel space-warping method called perspective warping, which allows us to handle arbitrary trajectories in the grid-based NeRF framework. Extensive experiments demonstrate that F^2-NeRF is able to use the same perspective warping to render high-quality images on two standard datasets and a new free trajectory dataset collected by us. Project page: https://totoro97.github.io/projects/f2-nerf.*

- This paper focuses on handling unbounded scenes by using a space warping method.

https://arxiv.org/pdf/2303.15951.pdf

https://github.com/totoro97/f2-nerf

Motivation

Prior works use “grid-based” methods such as: Plenoxels, DVGO, TensorRF, Instant-NGP.

- **Enables fast training** within a few minutes but **comes at the cost of high memory-consumption** (grows in cubic order with the size of the scene).

Method

- To represent unbounded scenes, a **commonly-adopted strategy is to use a SPACE-WARPING METHOD that maps an unbounded space to a bounded space.**

There Are 2 Common Warping Functions

- **Normalized Device Coordinate (NDC)** warping map an infinitely-far view frustum to a bounded box by squashing the space along the z-axis.
- **Inverse-Sphere Warping** maps an infinitely large space to a bounded sphere by the sphere
  inversion transformation. Typically used for 360 degree object-centric unbounded scenes.

▼ Explanation of above warping functions

1. **Normalized Device Coordinate (NDC) Warping:**

   Given a 3D point (x, y, z) in world space, the NDC warping can be represented mathematically as:

   a. Perspective Projection:
   Perform perspective projection to obtain the projected point (x', y', z'):

$$x' = x/z$$
$$y' = y/z$$
$$z' = z$$

   b. Homogeneous Coordinates:
   Convert the projected point to homogeneous coordinates:

$$[x', y', z', 1]$$

   c. Perspective Divide:
   Divide the homogeneous coordinates by the w component:

$$x'' = x'/w$$
$$y'' = y'/w$$
$$z'' = z'/w$$

   The resulting point (x'', y'', z'') represents the 3D point in normalized device coordinates within the range $[-1, 1]$ along each axis.

2. **Inverse-Sphere Warping:**

   Given a 3D point (x, y, z) in the original unbounded space, the inverse-sphere warping can be represented mathematically as:

   a. Calculate the distance (r) of the point from the origin:

$$r = \sqrt{x^2 + y^2 + z^2}$$

   b. Calculate the normalized direction vector (d) of the point:

$$d = (x/r, y/r, z/r)$$

c. Map the point to the surface of the sphere:

$$x' = d.x/r$$
$$y' = d.y/r$$
$$z' = d.z/r$$

After these transformations, the resulting point (x', y', z') lies on the surface of the bounded sphere.

Limitations of Above Warping Functions: These two warping **methods assume special camera trajectory patterns and cannot handle arbitrary ones**.

- Method is really bad for “free trajectories” (where trajectory is long and contains multiple objects of interest). **Why?**
  - Because of imbalanced allocation of spatial representation capacity.
    - If trajectory is narrow and long, regions in scenes are empty and invisible to any input views. Yet grids of prior methods are allocated evenly in space regardless of if empty or not. So much of representation capacity is wasted on empty space.

Introducing….

Perspective Warping: A General Space-Warping Scheme

- First, **represent location of a 3D point p as concatenation of 2D coordinates of projections of p in the input image** → and then **map these 2D coords into a compact 3D subspace using PCA**.
  - This is a generalization to NDC and inverse-sphere warping to arbitrary trajectories.

- In order to implement the perspective warping in a grid-based NeRF framework, we further propose **a SPACE SUBDIVISION ALGORITHM** to adaptively use coarse grids for background regions and fine grids for foreground regions

Math:

- Coordinate $\boldsymbol{y}_j = G(\boldsymbol{x}_j)$
- $\boldsymbol{z}_j = \boldsymbol{M}\boldsymbol{y}_j$
  - Projection matrix $\boldsymbol{M}$ maps $\boldsymbol{y}_j$ to $\boldsymbol{z}_j$.
    - $\boldsymbol{M}$ found by minimizing $||\boldsymbol{M}^T\boldsymbol{z}_j - \boldsymbol{y}_j||_2^2$).
    - **Insight:** The solution to the above problem is essentially **PCA**.

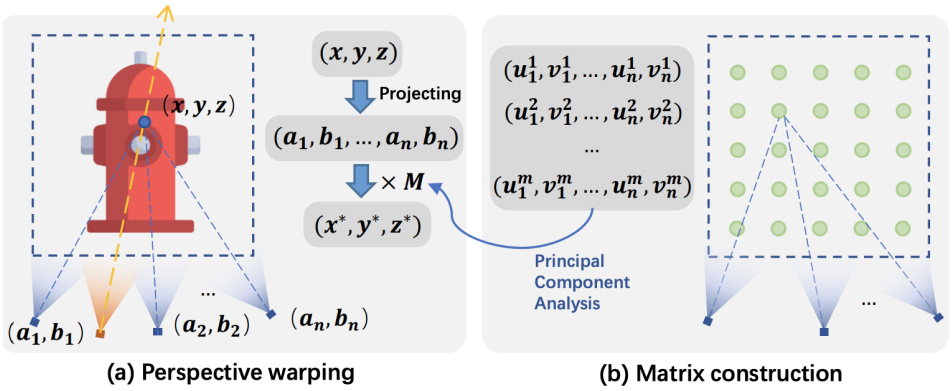


Figure 3. **Our perspective warping method.**

- More generally: $F(\boldsymbol{x}) = \boldsymbol{M}G(\boldsymbol{x})$
  - $F(\boldsymbol{x})$ describes a warping function **equal** to a **projection matrix** times the **original**, **unbounded coordinate.**
- **Intuition:** An unbounded coordinate ($\boldsymbol{y}_j$) is projected to **WARP SPACE.**

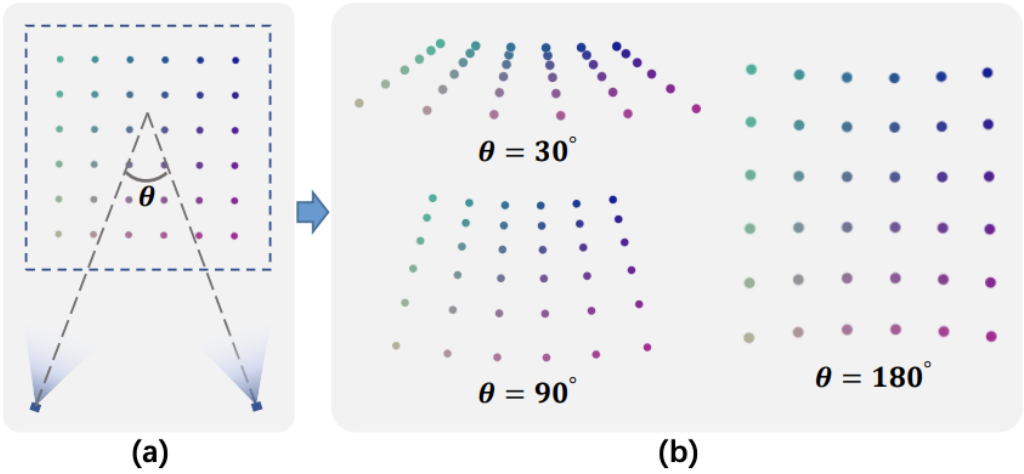


Figure 4. **Visualization of the effect of perspective warping.** (a) Points in the original Euclidean space. (b) Points in the warp space and the corresponding camera angles.

Space Subdivision

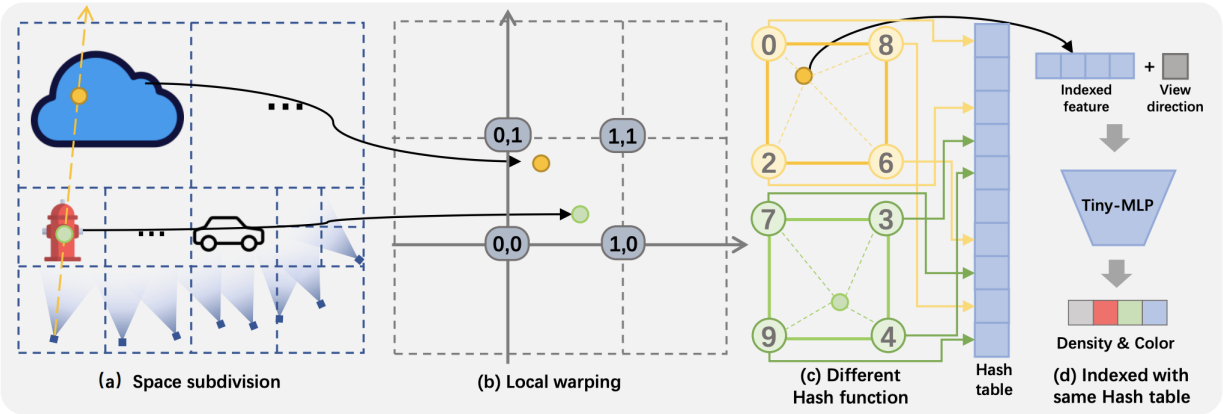
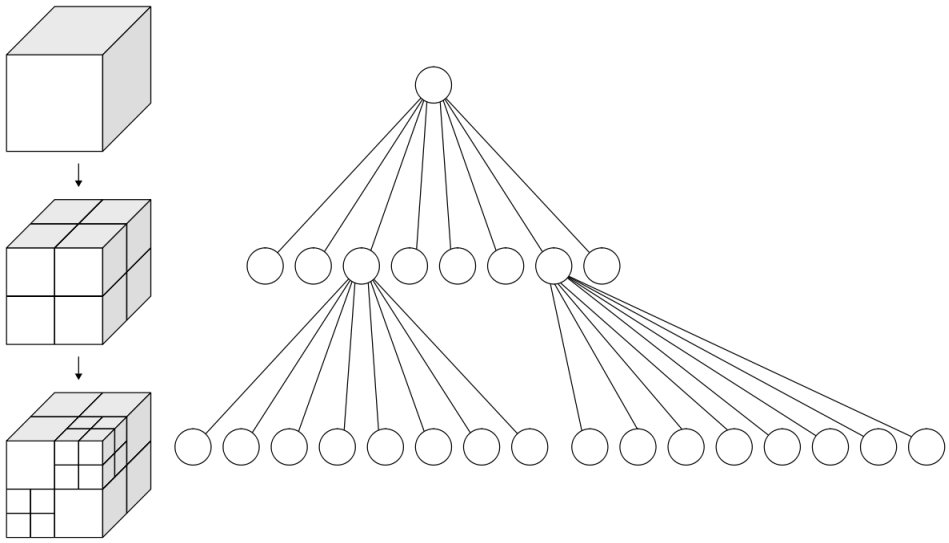


Figure 6. **Pipeline of F^2-NeRF .** (a) Given a large region of interest, we subdivide the space according to the input view frustums. (b) For each sub-region, we construct a perspective warping function based on the visible cameras. The densities and colors are decoded from the scene feature vectors fetched from the same hash table (d) but using different hash functions (c).

- Use **octree data structure** to store subdivided regions.
  - Start with large bounding box (512 x bounding box size of all camera centers)

- Perform check-and-subdivide procedure
  - On a tree node with s as side length, retrieve all visible cameras whose **view frustums intersect with this node**.
  - If **any visible camera center whose distance** d to node center is $\boldsymbol{d \leq \lambda s}$ ($\lambda = 3$ by default), **node is subdivided** into 8 child nodes with $s_{new} = s/2$.
    - Further check distance and **repeat procedure till we get all** $\boldsymbol{n_l}$ **leaf nodes**.
  - **Else**, current node is small enough. Stop subdividing it. **Mark it as a leaf node.**



- **Adding more cameras step:** For regions that are visible by more than $n_c = 4$ cameras, further select n_c visible cameras by making the minimal pairwise distance of selected cameras as large as possible.

- **Warp Step:** Apply warping function F_i to each leaf node to map it to a region around the origin of its warp space.

Scene Representation

Once you have points in warp space, we need to build our grid based scene representation. This will eventually allow us to predict color and density for a given point in the warp space.

- Warping functions are different for different leaf nodes → so we have n_l warp spaces.
- Instead of building n_l grid representations on each warp space, we **assume all warping functions map different leaf nodes to the same warp space** and **build a hash-grid representation on the warp space with multiple hash functions**.

Hash grid with multiple hash functions:

- **Sharing the same warp space** for different leaf nodes will **inevitably lead to conflicts.**
- In **InstantNGP, there is only one hash function** to compute hash values for grid vertices.
- So we use **different hash functions for different leaf nodes:**

$$\text{Hash}_i(\hat{\mathbf{v}}) = \left(\bigoplus_{k=1}^{3} \hat{\mathbf{v}}_k \pi_{i,k} + \Delta_{i,k} \right) \mod L,$$

- Where $\hat{\boldsymbol{v}}$ is grid vertices, $\pi_{i,k}$ and $\Delta_{i,k}$ are random large prime numbers, L = length of hash table, and $\bigoplus$ is the bitwise xor operation.
- **Intuition:** Using different numbers $\pi_{i,k}$ and different offsets $\Delta_{i,k}$ leads to different hash functions in equation above.
- The computed hash value will be used in indexing the hash table to retrieve a feature vector for the vertex $\hat{\boldsymbol{v}}$ and then the feature vector of the point z is trilinearly-interpolated from 8
  vertex feature vectors.

- **This bullet point is still a bit unclear. What does trilinearly-interpolated from 8 vertex feature vectors mean?**
- Answer: Trilinear interpolation from 8 vertex feature vectors means estimating the feature vector for a point (of interest) within a 3D grid by blending the feature vectors of the surrounding eight grid vertices using weights determined by their relative distances.

Perspective Sampling

- A proper warping function F also gives us a guideline for sampling points on rays in volume rendering.

3.5. Perspective sampling

A proper warping function F also gives us a guideline for sampling points on rays in volume rendering. According to Definition 1, the distance between two points in the proper warp space equals the sum of distances between two projected points on image planes. In this case, by uniformly sampling the points in the warp space, we get a non-uniform sampling in the original Euclidean space but an approximately uniform sampling on images, which improves the sampling efficiency and brings more stable convergence. Specifically, considering a sample point $\mathbf{x}_i = \mathbf{o} + t_i\mathbf{d}$ where $\mathbf{o}, \mathbf{d}$ are the camera origin and direction respectively, we first compute the Jacobian matrix $J_i \in \mathbb{R}^{3\times3}$ of the perspective warping function F at $\mathbf{x}_i$. Then, the next sample point $\mathbf{x}_{i+1} = \mathbf{x}_i + \frac{l}{\|J_i\mathbf{d}\|_2}\mathbf{d}$, where l is a preset parameter controlling the sampling interval and we make a linear approximation here as discussed in the supplementary material.

Results On Free Dataset

Method	Tr. time	PSNR↑	SSIM↑	LPIPS(VGG)↓
NeRF++ [62]	hours	23.47	0.603	0.499
mip-NeRF-360 [3]	hours	**27.01**	**0.766**	**0.295**
mip-NeRF-360$_{\text{short}}$	30m	22.04	0.537	0.586
Plenoxels [58]	25m	19.13	0.507	0.543
DVGO [39]	21m	23.90	0.651	0.455
Instant-NGP [26]	6m	24.43	0.677	0.413
F^2-NeRF	12m	**26.32**	**0.779**	**0.276**

Table 1. **Results on the Free dataset**. In mip-NeRF-360$_{\text{short}}$, we early stop the training to make them finished in 30 minutes. Training times are evaluated on a 2080ti GPU.

- For ***Free*** dataset: While **mip-NeRF does eventually get better render quality/clear results** they spend a **long time** to converge to it.

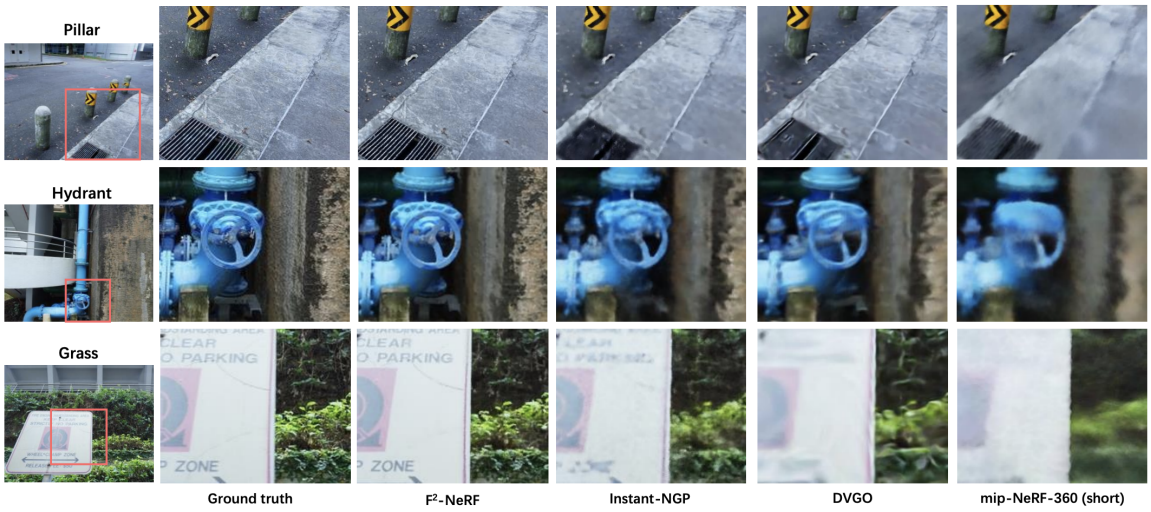


Figure 8. **Visual comparison on the Free dataset.**

Method	Tr. time	PSNR↑	SSIM↑	LPIPS(VGG)↓
NeRF++ [62]	hours	26.21	0.729	0.348
mip-NeRF-360 [2]	hours	**28.94**	**0.837**	**0.208**
Plenoxels [58]	22m	23.35	0.651	0.471
DVGO [39]	16m	25.42	0.695	0.429
Instant-NGP [26]	6m	26.24	0.716	0.404
F^2-NeRF	14m	**26.39**	**0.746**	**0.361**

Table 2. **Results on the NeRF-360-V2 dataset.**

Method	Tr. time	PSNR↑	SSIM↑	LPIPS(VGG)↓
NeRF [25]	hours	26.50	0.811	0.250
mip-NeRF [2]	hours	**26.60**	**0.814**	**0.246**
Plenoxels [58]	17m	26.29	0.839	0.210
DVGO [39]	11m	26.34	0.838	0.197
TensoRF [6]	48m	**26.73**	0.839	0.204
Instant-NGP [26]	6m	25.09	0.758	0.267
F^2-NeRF	13m	26.54	**0.844**	**0.189**

Table 3. **Results on the LLFF dataset.**

- For forward-facing dataset (LLFF) and 360 degree objectcentric dataset (NeRF-360-V2): Perspective warping is **competitive/similar to other methods in both time and accuracy.**

Notes From Appendices

Camera Rectification

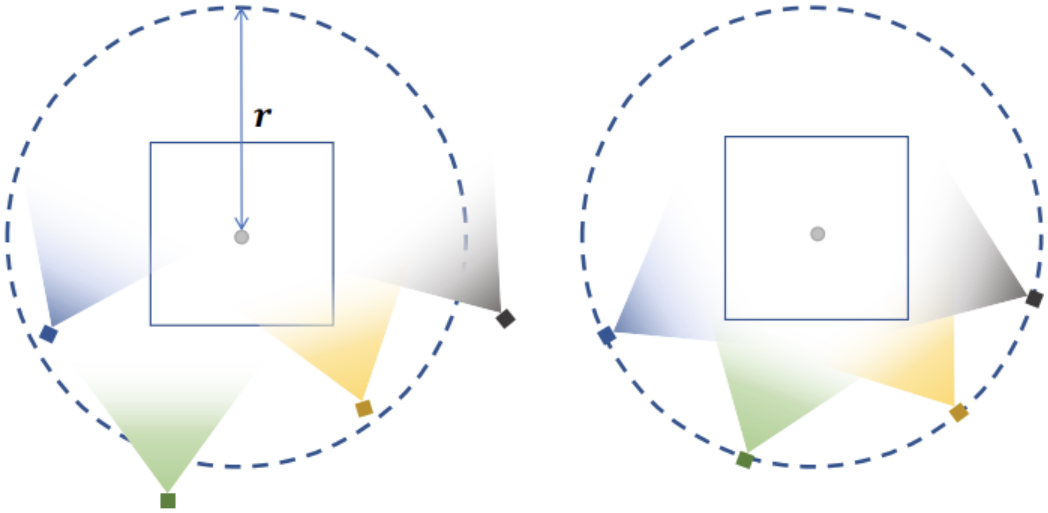


Figure 10. **Camera rectification.**

Still going through it…