基于可学习的掩码表征动态紧凑神经辐射场

徐耀建

摘要

动态神经辐射场 (Dynamic NeRF) 增强了 NeRF 技术来模拟移动场景。然而，它们是资源密集型的并且难以压缩。最近一些基于特征网格的方法已经显示出从输入 2D 图像快速学习隐式神经表示的优越性能。然而，在对动态场景进行建模时，这种显式表示很容易导致模型尺寸过大。为了解决这个问题，我们的关键思想是减少特征网格的时空冗余，这种冗余本质上是由于场景的自相似性而存在的。为此，我们提出了可学习的掩码用于紧凑型动态神经辐射场表征，详细地说对于空间平面，为了提高网格特征的稀疏性，我们建议在空间网格平面上添加一种新颖的可训练掩蔽方法。实验结果表明，在渲染指标相对不变的情况下，相比于 K-Planes 的模型参数大小，我们实现了 8 倍的模型参数量大小减少。

关键词：动态 NeRF；可学习的掩码；K-Planes

1 引言

体积视频记录动态 3D 场景的内容，允许用户从任意角度观看捕获的场景，具有广泛的应用，例如虚拟现实、增强现实、远程呈现等。传统方法通常将体积视频表示为一系列纹理网格，这些网格由多视图立体 [1] 或深度融合重建 [2]。此类技术需要复杂的硬件（例如多视图 RGB-D 传感器）来实现高重建精度，但成本昂贵且限制了其应用场景。

最近，隐式神经表示 [3] 已成为仅从 2D 图像恢复 3D 场景的有前途的策略。作为代表性工作，神经辐射场 (NeRF) [3] 采用 MLP 网络来预测任意 3D 点的密度和颜色，并通过体积渲染技术从 2D 图像中有效地学习模型参数。DyNeRF [4] 通过引入时变潜在代码作为 MLP 网络的附加输入，将 NeRF 扩展到动态场景，从而使模型能够对场景的每帧内容进行编码。尽管这些方法实现了令人印象深刻的渲染质量，但它们的训练速度非常慢，尤其是在动态场景中。例如，DyNeRF 需要超过 1000 个 GPU 小时来学习 10 秒的体积视频。

为了克服这个问题，一些方法利用显式表示，例如特征 volumes[5] 或特征平面 [6] 来加速训练过程。它们通常将可学习的特征向量存储在显式结构中，并使用插值技术将特征向量分配给任意场景坐标，然后将其输入轻量级 MLP 网络以预测密度和颜色。通过减少 MLP 网络中的参数数量，这些方法实现了训练的显着加速。然而，显式表示很容易占用大量存储空间，很难扩展到动态 3D 场景。

2 相关工作

人们已经进行了各种尝试将 NeRF 扩展到动态场景。现有的方法可以分为四种类型：基于形变的方法、基于流的方法、从时空输入直接推理以及基于分解的方法。

基于变形的方法了解场景的 3D 结构如何变形。使用估计的变形将每帧的 3D 辐射场扭曲为单个或多个规范帧。变形被参数化为 3D 平移 [7]、通过角轴表示的旋转和平移 [8] 或加权平移 [9]。

另一方面，基于流的方法估计连续帧中 3d 点的对应关系。神经场景流场 [10] 估计两个时间戳的辐射场之间的 3D 场景流。Xian 等人 [11] 提出了跨越 4D 时空场的辐照度场，并具有一些来自先验知识的约束。

尽管基于变形和基于流的方法显示出成功的结果，但它们也有一些缺点。例如，基于扭曲的方法无法处理拓扑变化。由于这些方法将每个输入帧扭曲为单个规范场景，因此很难正确表示规范场景中新出现或消失的 3D 辐射场。HyperNeRF [12] 学习表示多维规范形状基础的超空间。神经场景流场 [10] 通过引入遮挡掩模来解决问题，遮挡掩模用于估计场景流不适用的区域。然而，在训练过程中需要同时学习掩模和流场，这使得训练过程变得复杂，并且依赖于额外的信息，例如单目深度或光流估计。

DyNeRF [4] 在不估计形状变形或场景流的情况下，使用简单的神经网络来训练动态场景的 NeRF，尽管实验仅在同步多视图视频上进行。它利用 3D 位置和时间帧作为输入，并通过一系列完全连接的神经网络来预测颜色和密度。

基于分解的动态 NeRF。原始 NeRF 的主要缺点之一是训练速度慢。提出了一些并行工作来加速 NeRF 训练。其中，基于网格的表示，例如 Plenoxels [13]、直接体素网格优化 [14] 在训练时间方面表现出优越的性能，只需几分钟即可学习合理的辐射场。将网格表示与哈希函数相结合进一步提高了特征网格的效率和训练速度 [15]。Fang 等人 [5] 首先应用体素网格表示来训练动态 NeRF，与基于神经网络的方法相比，获得了更快的训练速度。Guo 等人 [16] 还提出了一种用于快速训练的可变形体素网格方法。

隐式神经表示的压缩相关工作。基于特征网格和特征平面的方法可以显着提高效率，但它们也会带来大量的存储要求。为了缓解这个问题，[17] 提出使用剪枝策略来消除特征网格中不必要的参数。[18] 将 3D 网格分解为较低维度。它利用 VM 分解将 3D 网格分解为多个 2D 矩阵，甚至利用 CP 分解将其分解为多个 1D 向量。虽然 CP 分解显着提高了存储效率，但它导致了不可忽略的性能损失。[19] 实现了可以通过所提出的排名残差学习策略动态调整的压缩。矢量量化 [20] 是一种经典的压缩技术，广泛应用于 2D 图像和视频压缩任务。其主要思想是通过聚类将相似的码本合并为一个，从而减少存储冗余。最近，VQAD[21] 和 VQRF [22] 使用矢量量化来实现静态 3D 场景的压缩。VQAD 在训练模型的同时学习码本，而 VQRF 则以后处理的方式实现压缩。[23] 利用傅里叶变换进行压缩，能够捕获高频细节，同时保持紧凑。[24] 提出使用小波系数来提高参数稀疏性。

3 本文方法

3.1 K-Planes 方法概述

本文是基于 K-Planes[25] 的代码基础上，实现模型的参数量压缩，同时保持渲染质量不下降。

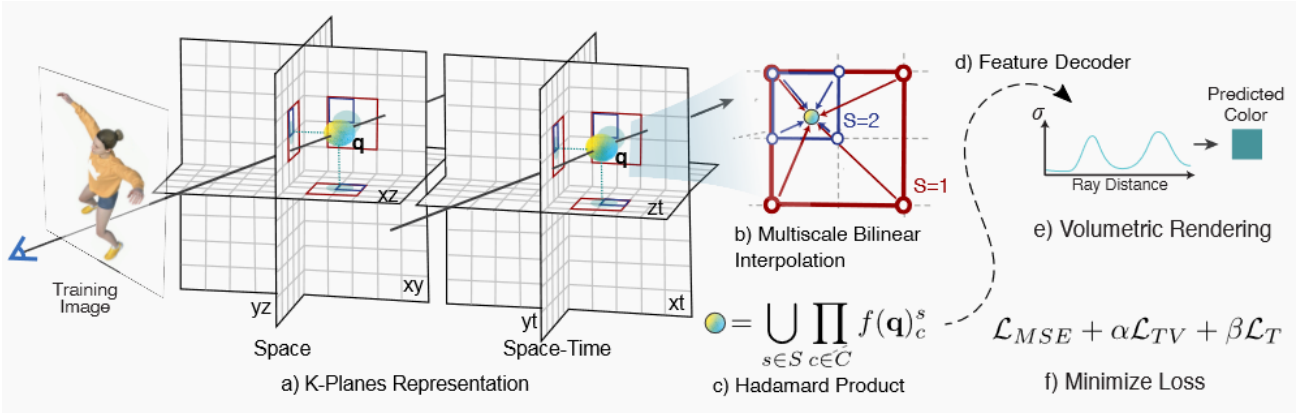


图 1. K-Planes 模型的整体架构。

K-Planes 方法执行过程：

(a) K-Planes 将 4D 动态体积分解为六个平面，其中三个用于空间，三个用于时空变化。从空间中发射一条射线，为了获得射线上的 4D 采样点 $q = (x, y, z, t)$ 的特征值，首先将该点投影到每个平面中，在其中 (b) 每个特征平面进行多尺度双线性插值。(c) 将插值相乘，然后在多尺度 S 上连接。(d) 这些特征可以使用小型 MLP 或显式线性解码器进行解码得到采样点的 color 和 density。(e) 最终遵循标准的体积渲染公式来预测光线的颜色和密度。通过 (f) 空间和时间的简单正则化和最小化重建损失来优化模型，实现端到端的可微分训练。

我们的改进点：

- 原有的 K-Planes 借鉴了 Mip-NeRF 360 的 Proposal Sampling Strategy, 为了加速射线采样的训练速度，我们使用了 NerfAcc 提供的空间区域跳跃采样替换了原有的采样策略，加快了训练速度。
- 在空间平面（XY，XZ，YZ）上添加了一个可学习的掩码，在训练过程中对于网格不重要的特征将会被置于 0。
- 提供了一个压缩和解压缩的管道，将训练完的模型通过压缩管道，实现有效的低内存存储。

3.2 空间跳跃采样

为了避免在场景空白的无效采样，使用了 NerfAcc 提供的空白空间跳跃采样策略，把采样点更多的分配在物体表面。

```
# 首先定义占有网格类，输入它的三维网格的分辨率，和它的bbox大小，当射线穿过占有网格，
# 如果网格为0说明此块区域为空白，则不需要采样，当网格为1，则需要采样
self.occupancy_grid = nerfacc.OccGridEstimator(
                roi_aabb=aabb.reshape(-1),resolution=self.occ_grid_reso,
                levels=occ_level
            )

def sigma_fn(t_starts, t_ends, ray_indices):
    t_origins = rays_o[ray_indices]
```

```
    if t_origins.shape[0] == 0:
        return torch.zeros((0,), device=t_origins.device)
    t_dirs = rays_d[ray_indices]
    t_times = (
        timestamps[ray_indices] if timestamps is not None else None
    )
    positions = t_origins + t_dirs * (t_starts + t_ends)[:, None] / 2.0
    return self.field.get_density(positions[:, None], t_times)[0][
        :, 0
    ].squeeze(-1)

def rgb_sigma_fn(t_starts, t_ends, ray_indices):
    t_origins = rays_o[ray_indices]
    if t_origins.shape[0] == 0:
        return torch.zeros(
            (0, 3), device=t_origins.device
        ), torch.zeros((0,), device=t_origins.device)
    t_dirs = rays_d[ray_indices]
    t_times = (
        timestamps[ray_indices] if timestamps is not None else None
    )
    positions = t_origins + t_dirs * (t_starts + t_ends)[:, None] / 2.0
    field_out = self.field(
        positions[:, None], t_dirs[:, None], t_times
    )
    return field_out["rgb"][:, 0], field_out["density"][:, 0].squeeze(-1)

def step_before_iter(self, step):
    if self.use_occ_grid and self.training:
        def occ_eval_fn(x):
            requires_timestamps = len(self.field.grids[0]) == 6
            density =
                self.field.get_density(x[:,None],timestamps=torch.rand_like(x[:,
                0])*2 -1 if requires_timestamps else None)[0][:,0]
            return density * self.occ_step_size

        self.occupancy_grid.update_every_n_steps(
            step=step, occ_eval_fn=occ_eval_fn, ema_decay=0.99
        )
    elif self.use_proposal_weight_anneal:
        # anneal the weights of the proposal network before doing PDF sampling
        N = self.proposal_weights_anneal_max_num_iters
        # https://arxiv.org/pdf/2111.12077.pdf eq. 18
```

```
        train_frac = np.clip(step / N, 0, 1)
        bias = lambda x, b: (b * x) / ((b - 1) * x + 1)
        anneal = bias(train_frac, self.proposal_weights_anneal_slope)
        self.proposal_sampler.set_anneal(anneal)

ray_indices, t_starts, t_ends = self.occupancy_grid.sampling(
    rays_o,
    rays_d,
    sigma_fn=sigma_fn,
    near_plane=nears[0, 0],
    far_plane=fars[0, 0],
    render_step_size=self.occ_step_size,
    stratified=self.training,
    alpha_thre=self.occ_alpha_thres,
)
rgb, accumulation, depth, _ = nerfacc.rendering(
    t_starts,
    t_ends,
    ray_indices,
    n_rays=rays_o.shape[0],
    rgb_sigma_fn=rgb_sigma_fn,
    render_bkgd=bg_color[0],
)
```

3.3 可学习的掩码

在 mask 初始化时，初始的维度和（XY,XZ,YZ）平面相同，在训练的过程中如果 mask 的值为 0，则对应的平面特征会被设置为 0，最后为了实现端到端的 mask 训练，加入了一个 mask 损失，使得 mask 越接近于 0。

```
# only mask space
def init_mask_param(self, res_index):
    mask_coefs = nn.ParameterList()
    for i in range(6):
        if i != 2 and i != 4 and i != 5:
            new_mask_coef = nn.Parameter(
                torch.ones_like(self.grids[res_index][i]))
            mask_coefs.append(new_mask_coef)
    return mask_coefs

mask = torch.sigmoid(self.masks[scale_id][t])
plane = (plane * (mask >= 0.5) - plane * mask).detach() + plane * mask
```

```
if self.model.field.use_mask and self.mask_weight > 0:
    mask_loss = 0.
    for i in range(4):
        mask_loss = mask_loss + sum([
            p.sum()
            for p in self.model.field.masks[i].parameters()
        ])
    loss = loss + self.mask_weight * mask_loss
```

3.4 压缩流程

通过我们提出的掩码方法，空间平面网格中的零点比例可以高达 80% 以上。然而，稀疏表示本身并不会减少总大小。我们描述了我们提出的稀疏网格参数压缩管道。我们不是按原样存储网格，而是单独存储非零系数和指示哪些系数非零的位图（或掩码）。尽管使用 1 位位图，但由于参数数量较多，总体位图大小也较大。为了减少位图大小，我们提出了一个具有以下三个阶段的压缩管道：n 位转换、行程编码（RLE）和霍夫曼编码。压缩管线如下图所示。

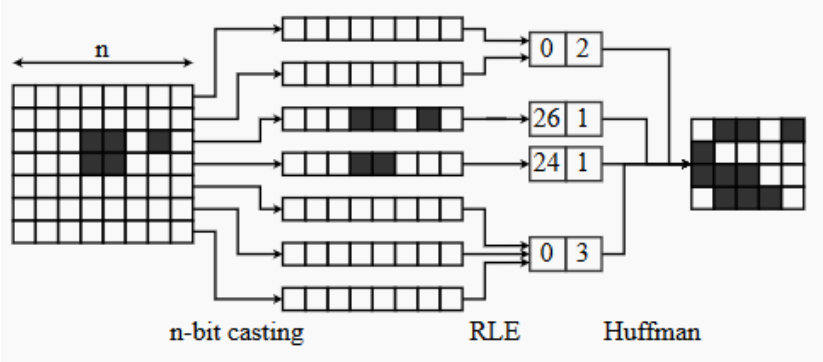


图 2. 压缩流程图

4 实验结果

4.1 与已有开源代码对比

本次实验在 K-Planes https://github.com/sarafridov/K-Planes 代码的基础上，增加了三个模块代码。

- 使用空白空间跳跃采样替换原有的采样策略。
- 提出了一个时空平面的可学习掩码提高特征平面的稀疏性。
- 提供了一个 Mask 压缩管道，包括 n-bit casting，游程编码和哈夫曼编码。

4.2 实验对比

我们在 D-NeRF[7] 单目数据集八个场景 800*800 分辨率进行了对比实验，可以看到相比于 K-Planes，我们的平均 PSRN 指标基本没有下降，单模型的参数量大小减少了 8 倍。space

mask ratio 表示对空间平面（XY，XZ，YZ）压缩比例，越大说明平面越稀疏，压缩率越大。

scene		Hell Warrior	Mutant	Hook	Balls	Lego	T-Rex	Stand Up	Jumping Jacks	average
paper_psnr		24.7291	32.5032	28.0668	41.1594	25.2158	30.6991	33.0900	31.4745	30.8672
our_psnr		24.6541	32.3088	27.9217	40.1563	25.3267	31.6496	32.9941	31.5282	30.8174
paper_ssim		0.9676	0.9809	0.9643	0.9974	0.9592	0.9815	0.9879	0.9828	0.97771
our_ssim		0.9521	0.9691	0.9467	0.9928	0.9464	0.9757	0.9784	0.9724	0.9667
paper_param_size(MB)	space plane size(XY,XZ,YZ)	133.359	136.289	133.359	136.289	130.430	162.734	136.289	139.219	138.496
	space-time plane size(XT,YT,ZT)	11.719	17.578	11.719	17.578	5.859	23.438	17.578	23.438	16.1134
	MLP param size	0.078	0.078	0.078	0.078	0.078	0.078	0.078	0.078	0.07813
	Model size	148.02	156.80	148.02	156.80	139.23	172.73	156.80	165.59	155.499
our_param_size(MB)	space mask ratio	0.8242	0.7967	0.8213	0.6707	0.8671	0.7553	0.7969	0.7610	0.78665
	space-time plane size(XT,YT,ZT)	11.7190	17.5780	11.7190	17.5780	5.8590	23.4380	17.5780	23.4380	16.1134
	MLP param size	0.0781	0.0781	0.0781	0.0781	0.0781	0.0781	0.0781	0.0781	0.07813
	Model size	21.4996	23.6689	21.8403	42.1890	16.1220	28.6732	23.2326	27.7694	25.6244

图 3. D-NeRF 数据集八个场景的实验结果对比图

4.3 一些可视化结果

在 D-NeRF Leog 数据集上，Kplanes 的 PSNR 为 25.2158，模型大小为 139.23MB，我们的 PSNR 为 25.3267，模型大小 16.12MB。

在 D-NeRF StandUp 数据集上，Kplanes 的 PSNR 为 33.09，模型大小为 156.80MB，我们的 PSNR 为 32.9941，模型大小 23.23MB。从而验证了我们的方法有效性，在不降低渲染指标的情况下，可以有效的减少模型大小。

5 展望

在实验的过程中发现，对空间、时空平面同时引入 Mask 会导致 psnr 指标降低 2db，这大大降低了渲染指标，因此未来准备在时空平面上研究如何进一步压缩。

References

[1] Johannes L Schonberger and Jan-Michael Frahm. “Structure-from-motion revisited”. In: *CVPR*. 2016, pp. 4104–4113.

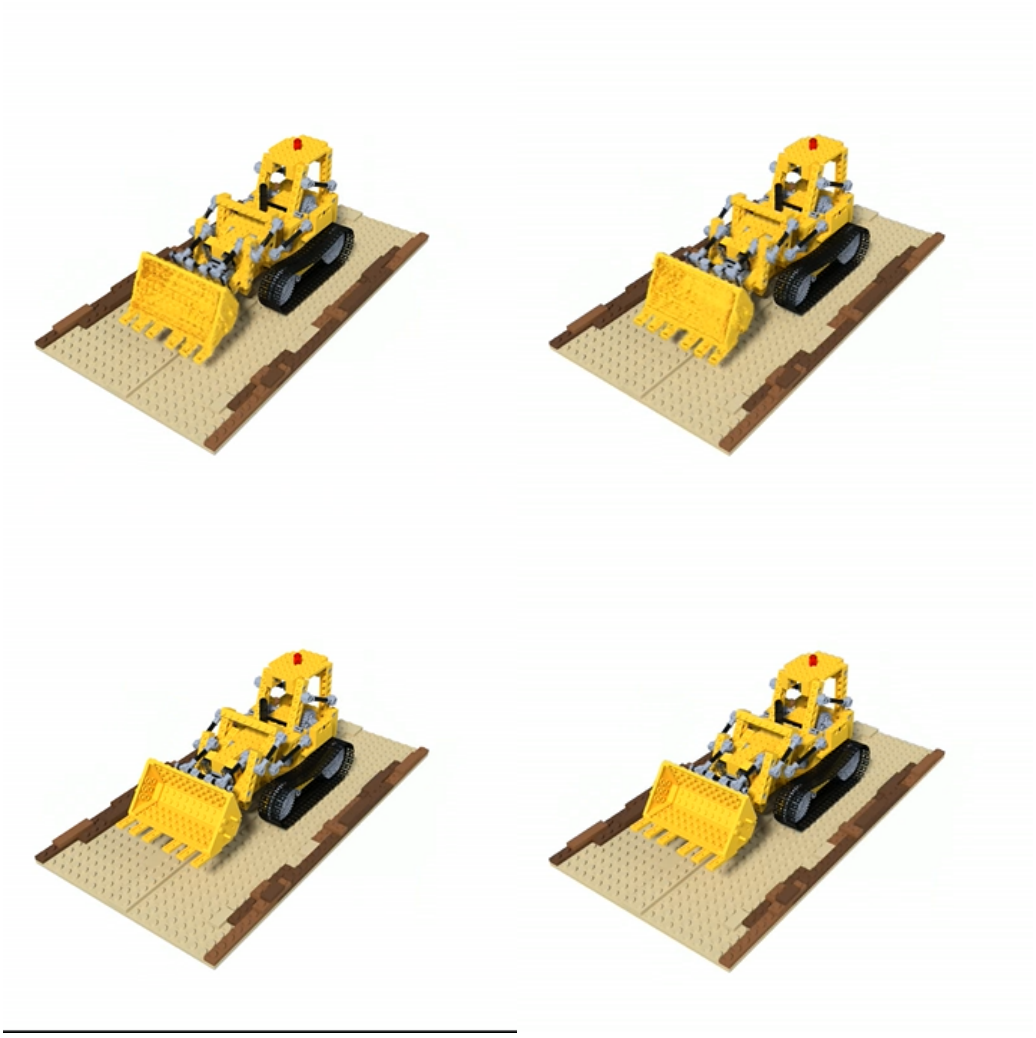
[2] Peng Gao et al. “Dynamic fusion with intra-and inter-modality attention flow for visual question answering”. In: *CVPR*. 2019, pp. 6639–6648.

[3] Ben Mildenhall et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”. In: *ECCV*. 2020.

[4] Tianye Li et al. “Neural 3d video synthesis from multi-view video”. In: *CVPR*. 2022, pp. 5521–5531.

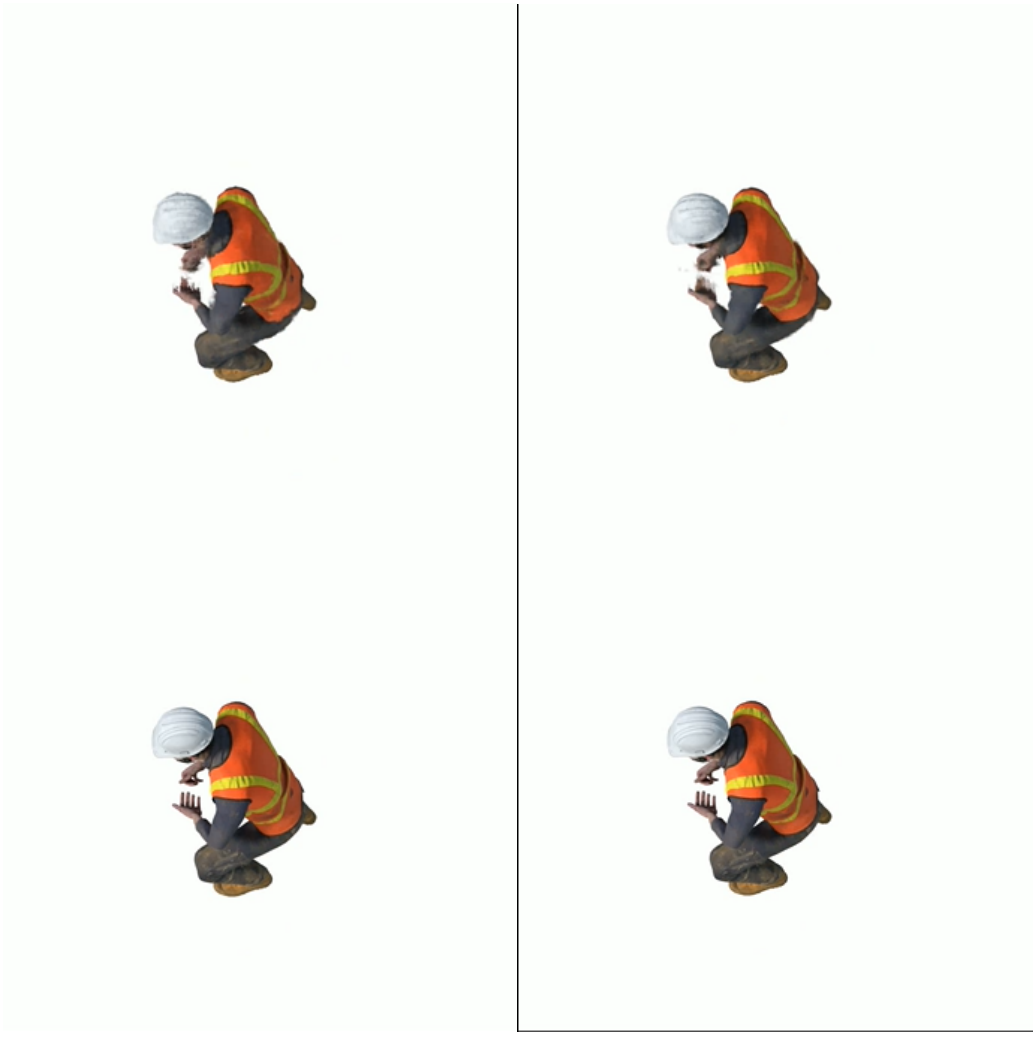
[5] Jiemin Fang et al. “Fast dynamic radiance fields with time-aware neural voxels”. In: *SIGGRAPH Asia 2022*. 2022, pp. 1–9.

[6] Ruizhi Shao et al. “Tensor4d: Efficient neural 4d decomposition for high-fidelity dynamic reconstruction and rendering”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 16632–16642.



(a) K-Planes (b) Ours

图 4. D-NeRF Lego 场景其中一张测试集可视化结果，上图为预测值，下图为真实值



(a) K-Planes (b) Ours

图 5. D-NeRF StandUp 场景其中一张测试集可视化结果，上图为预测值，下图为真实值

[7] Albert Pumarola et al. “D-NeRF: Neural Radiance Fields for Dynamic Scenes”. In: *CVPR* (2021).

[8] Keunhong Park et al. “Nerfies: Deformable neural radiance fields”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5865–5874.

[9] Edgar Tretschk et al. “Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 12959–12970.

[10] Zhengqi Li et al. “Neural scene flow fields for space-time view synthesis of dynamic scenes”. In: *CVPR*. 2021, pp. 6498–6508.

[11] Wenqi Xian et al. “Space-time neural irradiance fields for free-viewpoint video”. In: *CVPR*. 2021, pp. 9421–9431.

[12] Keunhong Park et al. “HyperNeRF: A Higher-Dimensional Representation for Topologically Varying Neural Radiance Fields”. In: *ACM Trans. Graph.* 40.6 (Dec. 2021).

[13] Sara Fridovich-Keil et al. “Plenoxels: Radiance fields without neural networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 5501–5510.

[14] Cheng Sun, Min Sun, and Hwann-Tzong Chen. “Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction”. In: *CVPR*. 2022.

[15] Thomas Müller et al. “Instant Neural Graphics Primitives with a Multiresolution Hash Encoding”. In: *ACM Trans. Graph.* 41.4 (July 2022), 102:1–102:15. DOI: 10.1145/3528223.3530127. URL: https://doi.org/10.1145/3528223.3530127.

[16] Xiang Guo et al. “Neural deformable voxel grid for fast optimization of dynamic view synthesis”. In: *ACCV*. 2022, pp. 3757–3775.

[17] Chenxi Lola Deng and Enzo Tartaglione. “Compressing explicit voxel grid representations: fast nerfs become also small”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2023, pp. 1236–1245.

[18] Anpei Chen et al. “Tensorf: Tensorial radiance fields”. In: *ECCV*. 2022, pp. 333–350.

[19] Jiaxiang Tang et al. “Compressible-composable nerf via rank-residual decomposition”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 14798–14809.

[20] Allen Gersho and Robert M Gray. *Vector quantization and signal compression*. Vol. 159. Springer Science & Business Media, 2012.

[21] Towaki Takikawa et al. “Variable bitrate neural fields”. In: *ACM SIGGRAPH 2022 Conference Proceedings*. 2022, pp. 1–9.

[22] Lingzhi Li et al. “Compressing volumetric radiance fields to 1 mb”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 4222–4231.

[23] Binbin Huang et al. “PREF: Phasorial Embedding Fields for Compact Neural Representations”. In: *arXiv preprint arXiv:2205.13524* (2022).

[24] Daniel Rho et al. “Masked Wavelet Representation for Compact Neural Radiance Fields”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2023, pp. 20680–20690.

[25] Sara Fridovich-Keil and Giacomo Meanti et al. “K-Planes: Explicit Radiance Fields in Space, Time, and Appearance”. In: *CVPR*. 2023.