NeRF从入门到放弃3: EmerNeRF

https://github.com/NVlabs/EmerNeRF
该方法是Nvidia提出的，其亮点是不需要额外的2D、3Dbox先验，可以自动解耦动静field。
核心思想：
1. 动、静filed都用hash grid编码，动态filed比静态多了时间t，静态的hash编码输入是(x,y,z)，动态是(x,y,z,t)。
2. **使用flow融合多帧的特征，预测当前时刻的点的前向和后向的flow，最后的动态Feature是0.25pre+0.5+0.25next**
3. 用3个head分别预测正常物体、天空和阴影。

3.1 SCENE REPRESENTATIONS

1 Scene decomposition

为了实现高效的场景解耦，把4D场景分解为静态场和动态场，**两者都分别由可学习的hash grid(instant NGP**) Hs和hd表示。（注，下标s和d分别表示static和dynamic，下文所有表示都是此含义）
这种解耦为与时间无关的特征 hs = Hs(x) 和时变特征 hd = Hd(x, t) 提供了一种灵活紧凑的 4D 场景表示，其中 x = (x, y, z)

是查询点的 3D 位置，t 表示其时间步长。这些特征通过轻量级MLP进一步转换为动态和静态的feature(gs和gd)，和用于预测每个点的密度 (σs 和 σd)。

Scene decomposition. To enable efficient scene decomposition, we design EmerNeRF to be a hybrid spatial-temporal representation. It decomposes a 4D scene into a static field $\mathcal{S}$ and a dynamic field $\mathcal{D}$, both of which are parameterized by learnable hash grids (Müller et al., 2022) $\mathcal{H}_s$ and $\mathcal{H}_d$, respectively. This decoupling offers a flexible and compact 4D scene representation for time-independent features $\mathbf{h}_s = \mathcal{H}_s(\mathbf{x})$ and time-varying features $\mathbf{h}_d = \mathcal{H}_d(\mathbf{x}, t)$, where $\mathbf{x} = (x, y, z)$ is the 3D location of a query point and t denotes its timestep. These features are further transformed into $\mathbf{g}_s$ and $\mathbf{g}_d$ by lightweight MLPs (g_s and g_d) and used to predict per-point density σ_s and σ_d:

$$\mathbf{g}_s, \sigma_s = g_s(\mathcal{H}_s(\mathbf{x})) \qquad \mathbf{g}_d, \sigma_d = g_d(\mathcal{H}_d(\mathbf{x}, t)) \tag{1}$$

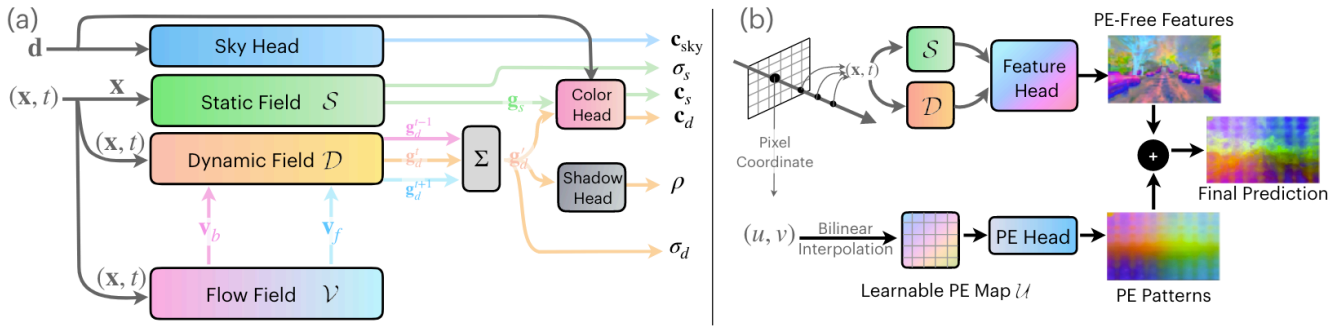

Figure 2: **EmerNeRF Overview.** (a) EmerNeRF consists of a static, dynamic, and flow field $(\mathcal{S}, \mathcal{D}, \mathcal{V})$. These fields take as input either a spatial query $\mathbf{x}$ or spatial-temporal query $(\mathbf{x}, t)$ to generate a static (feature $\mathbf{g}_s$, density σ_s) pair or a dynamic (feature $\mathbf{g}'_d$, density σ_d) pair. Of note, we use the forward and backward flows ($\mathbf{v}_f$ and $\mathbf{v}_b$) to generate temporally-aggregated features $\mathbf{g}'_d$ from nearby temporal features $\mathbf{g}_d^{t-1}$, $\mathbf{g}_d^t$, and $\mathbf{g}_d^{t+1}$ (a slight abuse of notation w.r.t. Eq. (8)). These features (along with the view direction $\mathbf{d}$) are consumed by the shared color head which independently predicts the static and dynamic colors $\mathbf{c}_s$ and $\mathbf{c}_d$. The shadow head predicts a shadow ratio ρ from the dynamic features. The sky head decodes a per-ray color $\mathbf{c}_{\text{sky}}$ for sky pixels from the view direction $\mathbf{d}$. (b) EmerNeRF renders the aggregated features to 2D and removes undesired positional encoding patterns (via a learnable PE map followed by a lightweight PE head).

所以这一步得到每个3D点的feature和密度。

2 Multi-head prediction

用三个head分别预测 color sky 和shadow，动态和静态共享共一个color mlp。
该color head以 (gs, d) 和 (gd, d) 作为输入，并为每个点都输出一个静态和动态的颜色；由于天空的深度定义不明确，所以单独加一个head预测天空的深度；添加一个影子的head去表述动态物体的阴影，输出动态对象0-1的标量，调整静态场预测的颜色强度。

由此图可看出，MLP_color的输入分别是动态feature和朝向，shadow head的输入是动态feature，sky head的输入只是朝向（为什么要这么做，因为没有深度信息，不知道采样多少个点）。

$$\mathbf{c}_s = \mathrm{MLP}_{\mathrm{color}}(\mathbf{g}_s, \gamma(\mathbf{d})) \qquad \mathbf{c}_d = \mathrm{MLP}_{\mathrm{color}}(\mathbf{g}_d, \gamma(\mathbf{d})) \tag{2}$$

$$\mathbf{c}_{\mathrm{sky}} = \mathrm{MLP}_{\mathrm{color\_sky}}(\gamma(\mathbf{d})) \qquad \rho = \mathrm{MLP}_{\mathrm{shadow}}(\mathbf{g}_d) \tag{3}$$

3.2 EMERGENT SCENE FLOW

1 场景流估计（Scene flow estimation）

用flow的head对当前时刻的query点，预测前向和后向的流。
最后的动态Feature是0.25pre+0.5+0.25next**

该特征聚合模块实现了三个目标：1）它将流场与场景重建损失（例如 RGB 损失）连接起来进行监督，2）它巩固特征、去噪时间属性以进行准确预测，以及 3）每个点通过其时间链接特征的共享梯度来丰富，通过共享知识提高单个点的质量

Hv和Hd应该是一样的。

$$\mathbf{v} = \mathrm{MLP}_v(\mathcal{H}_v(\mathbf{x}, t)) \qquad \mathbf{x}' = \mathbf{x} + \mathbf{v} \tag{7}$$

flow部分代码：MLP的最后一层的输出是6维，前3维表示forward flow，后3维表示backwark flow。**注意，最后一层mlp是没有激活函数的，以为要预测前后项的flow值，理论上有正负的，所以不能加激活函数。**

```
# ======== Flow Field ======== #
self.flow_xyz_encoder = None
if self.cfg.enable_flow_branch:
    self.flow_xyz_encoder =
HashEncoder(self.cfg.flow_xyz_encoder)
    self.flow_mlp = nn.Sequential(
        nn.Linear(
            self.flow_xyz_encoder.n_output_dims,
            self.cfg.base_mlp_layer_width,
        ),
        nn.ReLU(),
        nn.Linear(self.cfg.base_mlp_layer_width,
self.cfg.base_mlp_layer_width),
        nn.ReLU(),
```

```
        nn.Linear(self.cfg.base_mlp_layer_width,
6),  # 3 for forward, 3 for backward
        # no activation function for flow
    )
```

2 特征聚合模块(Multi-frame feature integration)

预测出forward 和backwark flow后，加到原本的位置，即得到上一阵和下一帧的位置，把上一阵和下一帧的位置都送到动态的mlp网络中。

$$\mathbf{g}'_d = 0.25 \cdot g_d(\mathcal{H}_d(\mathbf{x} + \mathbf{v}_b, t-1)) + 0.5 \cdot g_d(\mathcal{H}_d(\mathbf{x}, t)) + 0.25 \cdot g_d(\mathcal{H}_d(\mathbf{x} + \mathbf{v}_f, t+1)) \quad (8)$$

上图公式中，gd是动态的mlp，Hd是hash编码，也就是说当前帧点的坐标加上前后相的光流偏移量(Δx,Δy,Δz)后，和上一帧的时间t，再次进行hash编码，然后都送到动态的mlp网络中得到上一帧和下一帧的动态feature，再和当前帧的feature加权平均。

把flow和场景重建的loss损失结合起来进行监督；增强了动态部分的特征，去噪时间属性以进行准确的预测；每个点通过其时间链接特征的共享梯度来丰富，通过共享知识提高单个点的质量。

没有用显式的监督，这种能力来自于时间聚合步骤，同时优化场景重建损失。我们的假设是，只有时间一致的特征受益于多帧特征集成，这种集成间接地将场景流场驱动到最优解——预测所有点的正确流。

3 消融实验

消融研究证实了这一点：当禁用时间聚合或停止这些附近特征的梯度时，flow无法学习有意义的结果，加入flow, psnr+1

实验细节

只用了3个相机，图片resize成640x960。25K迭代，8196。静态场景把flow和dynamic分支去掉。静态场景不加feature，加feature40分钟，动态场景不加feature2小时，加feature2.25小时。