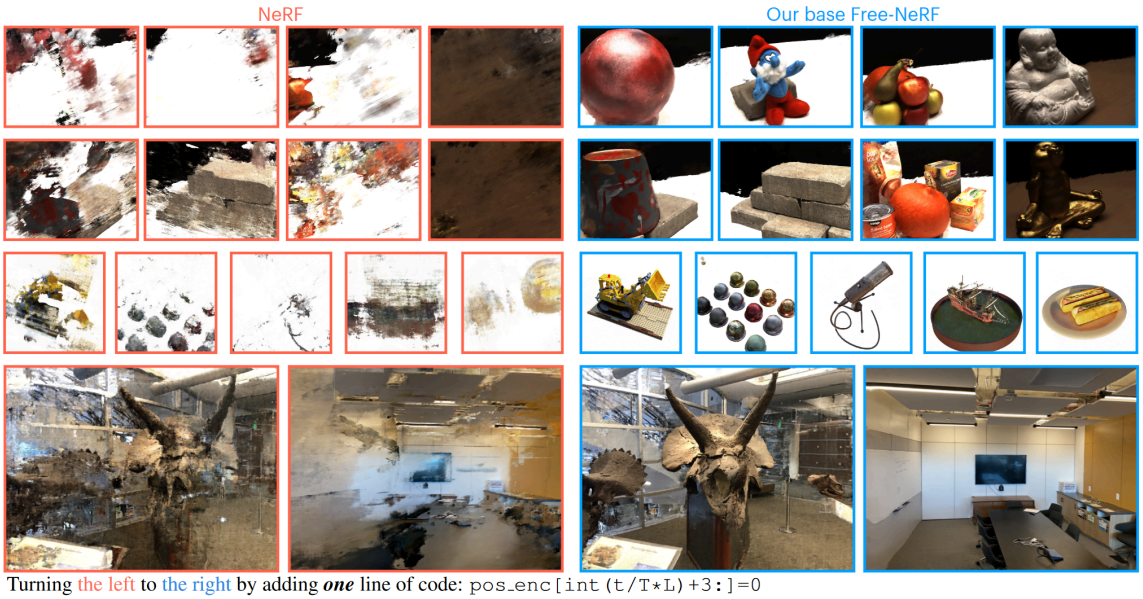
FreeNeRF

> j



Turning the left to the right by adding ***one*** line of code: `pos_enc[int(t/T*L)+3:]=0`

Figure 1. **Example novel view synthesis results from sparse inputs.** The only difference between NeRF (left) and FreeNeRF (right) is the use of our frequency regularization, which can be implemented as few as, approximately, *one* line of code (bottom, where t and T denote the current training iteration and regularization duration, respectively; L is the length of the input positional encoding).

我感觉好像就是加了俩正则化。

Frequency Regularization

Building on this empirical finding, we propose a frequency regularization method. Given a positional encoding of length $L + 3$ (Eq. (2)), we use a linearly increasing frequency mask $\boldsymbol{\alpha}$ to regulate the visible frequency spectrum based on the training time steps, as follows:

$$\gamma'_L(t, T; \mathbf{x}) = \gamma_L(\mathbf{x}) \odot \boldsymbol{\alpha}(t, T, L), \tag{4}$$

$$\text{with } \boldsymbol{\alpha}_i(t, T, L) = \begin{cases} 1 & \text{if } i \leq \frac{t \cdot L}{T} + 3 \\ \frac{t \cdot L}{T} - \lfloor \frac{t \cdot L}{T} \rfloor & \text{if } \frac{t \cdot L}{T} + 3 < i \leq \frac{t \cdot L}{T} + 6 \\ 0 & \text{if } i > \frac{t \cdot L}{T} + 6 \end{cases} \tag{5}$$

where $\boldsymbol{\alpha}_i(t, T, L)$ denotes the i-th bit value of $\boldsymbol{\alpha}(t, T, L)$; t and T are the current training iteration and the final iteration of frequency regularization, respectively. Concretely, we start with raw inputs without positional encoding and linearly increase the visible frequency by 3-bit each time as training progresses. This schedule can also be simplified as one line of code, as shown in Figure 1. Our frequency regularization circumvents the unstable and susceptible high-frequency signals at the beginning of training and gradually provides NeRF high-frequency information to avoid over-smoothness.

给定长度为 (L + 3) 的位置编码，通过一个线性增长的频率掩码 (\alpha) 来控制在不同训练时间步长下的频率可见性。公式（4）给出了这一调节方式：

$\gamma'_L(t, T; \mathbf{x}) = \gamma_L(\mathbf{x}) \odot \alpha(t, T, L)$

其中，$\gamma_L(\mathbf{x})$是输入数据的位置编码，$\alpha(t, T, L)$是一个频率掩码，随着训练时间步长逐步调整频率范围。具体地，$\alpha_i(t, T, L)$ 的定义如下：

$$\alpha_i(t, T, L) = \begin{cases} 1 & \text{if } i \leq \frac{t \cdot L}{T} + 3 \\ \frac{t \cdot L}{T} - \lfloor \frac{t \cdot L}{T} \rfloor & \text{if } \frac{t \cdot L}{T} + 3 < i \leq \frac{t \cdot L}{T} + 6 \\ 0 & \text{if } i > \frac{t \cdot L}{T} + 6 \end{cases}$$

其中 (i) 表示第 (i) 位的频率分量，(t) 是当前训练迭代次数，(T) 是频率正则化的最终迭代次数。随着训练的进行，开始时不使用位置编码，逐步增加可见的频率，每次增加 3 位频率信号，从而稳定训练初期不稳定的高频信息，同时在训练后期逐步引入高频信息，以避免过度平滑。

这个方法的目标是通过在训练初期抑制高频信息来提升模型的稳定性，随着训练的深入，再逐步引入高频特征，从而帮助模型学习到更多细节信息。这样可以在避免过早过拟合的同时，确保模型在训练结束时能够有效捕捉到高频特征。

Occlusion Regularization

As discussed above, the presence of floaters and walls in novel views is caused by the imperfect training views, and thus can be addressed directly at training time without the need for novel-pose sampling [22, 11, 37]. To this end, we propose a simple yet effective "occlusion" regularization that penalizes the dense fields near the camera. We define:

$$\mathcal{L}_{occ} = \frac{\boldsymbol{\sigma}_K^\intercal \cdot \mathbf{m}_K}{K} = \frac{1}{K} \sum_K \sigma_k \cdot m_k, \tag{6}$$

where $\mathbf{m}_k$ is a binary mask vector that determines whether a point will be penalized, and $\boldsymbol{\sigma}_K$ denotes the density values of the K points sampled along the ray in the order of proximity to the origin (near to far). To reduce solid floaters near the camera, we set the values of $\mathbf{m}_k$ up to index M, termed as regularization range, to 1 and the rest to 0. The occlusion regularization loss is easy to implement and compute.

公式中的符号解释：

- σ_K: 这是一个长度为 K 的密度值向量，表示在光线沿途上采样的 K 个点的密度值。通常，这些密度值用于表示光线穿过不同位置时的透明度或不透明度。
- $\mathbf{m}_k$: 这是一个二进制掩码（mask）向量，长度也是 K。它的作用是决定哪些点会被正则化惩罚。向量中的元素 m_k 取值为 0 或 1：
  - $m_k = 1$ 表示该点需要被正则化，即施加惩罚。
  - $m_k = 0$ 表示该点不需要正则化。
- σ_k: 这是密度值向量 σ_K 中第 k 个点的密度值。
- m_k: 是二进制掩码向量 $\mathbf{m}_K$ 中第 k 个点的掩码值。
- K: 光线沿途采样的点数。

公式的作用和含义：

- **损失函数的计算**:

$$\mathcal{L}_{\text{occ}} = \frac{\sigma_K^T \cdot \mathbf{m}_K}{K} = \frac{1}{K} \sum_{k=1}^{K} \sigma_k \cdot m_k$$

是通过对每个点的密度值 σ_k 与对应的掩码值 m_k 相乘后求和，再除以 K 得到的。这相当于对光线上的所有点施加了一定的正则化惩罚，具体施加到哪些点取决于掩码 $\mathbf{m}_K$ 的值。

- **掩码的作用**: $\mathbf{m}_k$ 作为二进制掩码向量，用来指定哪些点会被加入到遮挡正则化的惩罚范围。根据设置，靠近摄像机的点通常更容易产生浮点现象，因此通过设置靠近摄像机的点的 $\mathbf{m}_K$ 为 1，可以对这些点的密度施加惩罚，减小其密度值，从而减少靠近摄像机的虚假物体。
- **减少浮点（floaters）的效果**: 这种正则化方法旨在减少密度场中靠近摄像机的高密度浮点，使得场景更加平滑和合理。通过将靠近摄像机的密度值降下来，可以减少模型生成的无意义的“浮动物体”或“墙壁”，从而提高生成视图的质量。

直观理解：

该正则化损失在训练过程中，通过在距离摄像机较近的点上施加惩罚，强制模型减少这些点的密度值，使得模型生成的3D场景更符合真实的物理情况。这样可以更有效地避免在渲染过程中产生不必要的虚假物体，提升模型的表现。