Plenoxels笔记

> Plenoxels: Radiance Fields without Neural Networks

这一篇工作我认为对于NeRF的机制有着非常深刻的思考。

- 前置工作：PlenOctree

主要贡献：

- 无需使用神经网络，大大减少了训练时间
- 证明了可微渲染器才是关键
- 比传统NeRF快了2个数量级（100倍）
- 还可以拓展到360度无界场景

Method

在 PlenOctree 中，作者发现 Baking（通常指的是将某些复杂的计算，如光照、阴影或几何信息，预计算并“烘焙”到一种更简洁、更易于处理的数据结构中，从而在后续的渲染或推理过程中提高效率。） 得到的 Octree 也能利用体渲染进行相应的优化，并且优化速度还非常快，那么自然而然诞生就会出一个想法：我能不能一开始以 Baking 的表征（比如网格、八叉树）进行训练呢，训练过程就是 Baking 的过程。Plenoxel 就是这一一种解决方式。

作者发现 Baking 的主要作用反而不是 Raidance 部分的固定，而是 Geometry 部分的固定（思考一下，PlenOctree 的 fine-tune 过程是不改变八叉树结构的，而仅改变八叉树叶节点的值）。这也就意味着，如果我能在训练过程中实现表征的 Geometry 优化，那这个问题基本上就算解决了。

说白了，在 PlenOctree 中，Baking 之后，几何（Geometry）部分是固定的，不再发生改变，这意味着在后续的优化过程中唯一可以优化的是 辐射场部分（Radiance），即光照和颜色等内容。Baking 主要是在几何部分提供了一种有效的表征（八叉树），随后微调辐射部分的参数。而 Plenoxel 希望在训练过程中进行几何优化，不依赖于静态的八叉树结构，而八叉树并不是一个适合进行形状优化的表征，作者就把目光放到了更适合优化的稀疏体素网格--离散的八叉树叶节点，也就是系数网格 (sparse voxel grid) 身上。为了实现直接对稀疏网格进行优化，作者设置了一堆的组件，这也是整个 Plenoxel 的逻辑。

Plenoxel 是完全的 explicit 方法，没有用到任何 MLP，只用网格的顶点（其实网格和网格顶点的表述是能够相互转换的，理解就行）来存参数，存取的参数也沿用了 PlenOctree 中用到的σ和球谐函数的系数 (spherical harmonic coefficients)。

用稀疏体素网络存储σ和球谐系数，任意点的值是通过相邻体素进行三线插值得到的，后续使用loss直接优化网格内存储的值。

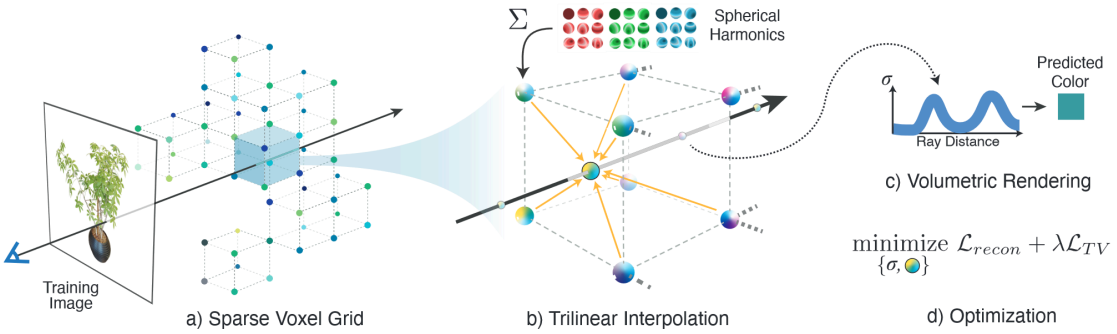

Figure 2. **Overview of our sparse Plenoxel model.** Given a set of images of an object or scene, we reconstruct a (a) sparse voxel ("Plenoxel") grid with density and spherical harmonic coefficients at each voxel. To render a ray, we (b) compute the color and opacity of each sample point via trilinear interpolation of the neighboring voxel coefficients. We integrate the color and opacity of these samples using (c) differentiable volume rendering, following the recent success of NeRF [26]. The voxel coefficients can then be (d) optimized using the standard MSE reconstruction loss relative to the training images, along with a total variation regularizer.

NeRF体渲染公式不足

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

$$\text{where} \qquad T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$$

- 积分离散化肯定会有损失
- 只考虑了单散射，因为在实际物理世界中应该还会有吸收、放射、外散射、内散射。（详情见另外一篇文章"体渲染"）

Grid Representation

这里和PlenOctree的不同，为了实现三线插值，作者没有使用八叉树结构，而是将带有指针的密集 3D 索引数组存储到仅包含占用体素值的单独数据数组中。说人话，这个密集数组并不直接存储所有体素的数据。相反，它指向一个独立的数据数组，这个数据数组只包含那些被占据的体素的数据。也就是说，网格中的体素指针指向一个包含实际信息的稀疏数据数组。

同时，球谐函数依然发挥着很重要的作用。

- 低次谐波（即低频分量）主要用于捕捉颜色的平滑变化。它们表示光照场景中的低频信息，即较为均匀、平滑的变化，类似于朗伯体反射（Lambertian）表面。
- 作用：编码的是光线方向上的平滑、缓慢变化的光照和颜色。这些变化通常出现在没有强烈反射的表面上，比如哑光表面，光线在这些表面上扩散得非常均匀。
- 高次谐波（即高频分量）用于编码高频信息，比如镜面反射（specular effects）等场景中的光照变化。这类变化通常非常快速和局部，比如光线在光滑表面上的反射会导致明亮的高光。
- 作用：捕捉光线方向上快速变化的反射和细节，比如高光或反射。这些特性在有光滑表面或者镜面反射的物体上更为明显。

本文使用的球谐函数仅仅有2阶，因为作者发现再提高没啥性能提升。一个颜色通道要1+3+5=9个系数，RGB三个颜色通道总共27个系数。

Plenoxel 网格使用三线性插值来定义整个体积的连续全光函数。这与 PlenOctrees 形成对比，PlenOctrees 假设每个体素内的不透明度和球谐系数保持恒定。事实证明，这种差异是成功优化体积的一个重要因素，正如下面讨论的。所有系数（针对不透明度和球谐函数）均直接优化，无需任何特殊的初始化或神经网络预训练。

Interpolation插值

作者通过实验发现三线插值远比最近邻插值好。因为插值通过表示颜色和不透明度的子体素变化来提高有效分辨率，并且插值产生连续函数近似，说白了就是提高函数连续性，实现可微，这对于成功优化至关重要。

Coarse to Fine

此外，为了更好的性能，作者提出了网格的 Coarse to Fine 策略，即训练到一定阶段时，对网格进行上采样（所有网格一分八，分辨率从 25632563 变成了 51235123 ，插值出来的网格的参数用三线性插值的方式得到），然后再进行优化，绝对步长也进行了缩小（光线上的样本点增多了）。

同时在上采样之后，Plenoxel 会对网格根据w或者σ进行剪枝操作，仅关注非空的网格，所以这个在上采样后的存储会更小，训练速度也会更快。

同时由于三线性插值依赖样本点附近八个网格的参数可靠，所以剪枝过程会做一个膨胀处理避免影响外面一圈的样本点的插值效果，以便仅在体素本身及其邻居都被视为未被占用时才对体素进行修剪。

OPtimization

$$\mathcal{L} = \mathcal{L}_{recon} + \lambda_{TV}\, \mathcal{L}_{TV} \tag{3}$$

Where the MSE reconstruction loss $\mathcal{L}_{recon}$ and the total variation regularizer $\mathcal{L}_{TV}$ are:

$$\mathcal{L}_{recon} = \frac{1}{|\mathcal{R}|} \sum_{\mathbf{r}\in\mathcal{R}} \|C(\mathbf{r}) - \hat{C}(\mathbf{r})\|_2^2$$

$$\mathcal{L}_{TV} = \frac{1}{|\mathcal{V}|} \sum_{\substack{\mathbf{v}\in\mathcal{V} \\ d\in[D]}} \sqrt{\Delta_x^2(\mathbf{v},d) + \Delta_y^2(\mathbf{v},d) + \Delta_z^2(\mathbf{v},d)}$$

由于 Plenoxel 是完全的 explicit 方法，没有用到任何 MLP，这意味着网络点存放的参数的优化是完全独立的。
而这很可能会导致训练过程中因为视角分布导致失真问题。想象一下，我某部分网格存了较大（这里的较大不是单纯数值上的大，理解一下）的参数就能够很好的应付训练视角的渲染，而另一部分网格由于前面的这些网格承担了大部分的渲染任务，使得它只存了较少的参数。让我进行新视角合成的时候，这两部分网格参数的割裂感就会以失真的形式展示出来。

对于上述问题，Plenoxel 提出了相应的 smooth prior 来处理，通过计算 TV loss (total varitation)来使相邻网格的值变得平滑。

$$\mathcal{L}_{TV} = \frac{1}{|V|} \sum_{\mathbf{v} \in V} \sum_{d \in [D]} \sqrt{\Delta_x^2(\mathbf{v}, d) + \Delta_y^2(\mathbf{v}, d) + \Delta_z^2(\mathbf{v}, d)}$$

Plenoxel 对于 σ 和球谐函数的系数分别采用了不同权重的 TV loss 进行约束。

PS：这个啥TV（总变差正则）其实就是这个东西。

- 1. 离散情况下的 TV 正则化

设 x 为一个待优化的图像或体素值的集合，$x_{i,j,k}$ 表示体素网格中某个体素的值，那么二维图像的总变差可以表示为：

$$TV(x) = \sum_{i,j} \sqrt{(x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2}$$

对于三维的体素网格数据（如 Plenoxel 中的稀疏体素），总变差可以推广为：

$$TV(x) = \sum_{i,j,k} \sqrt{(x_{i+1,j,k} - x_{i,j,k})^2 + (x_{i,j+1,k} - x_{i,j,k})^2 + (x_{i,j,k+1} - x_{i,j,k})^2}$$

其中，$x_{i+1,j,k}$ 表示网格中与 $x_{i,j,k}$ 相邻的体素值。

- 2. Plenoxel 中的 TV 正则化

在 Plenoxel 中，TV 正则化的公式为：

$$\mathcal{L}_{TV} = \frac{1}{|V|} \sum_{\mathbf{v} \in V} \sum_{d \in [D]} \sqrt{\Delta_x^2(\mathbf{v}, d) + \Delta_y^2(\mathbf{v}, d) + \Delta_z^2(\mathbf{v}, d)}$$

这里的 $\mathbf{v} = (i, j, k)$ 表示体素坐标，$\Delta_x^2(\mathbf{v}, d), \Delta_y^2(\mathbf{v}, d), \Delta_z^2(\mathbf{v}, d)$ 分别表示体素网格在不同方向上值的差异。也就是说，它通过约束体素在 x, y, z 三个方向上的值的变化，使得相邻体素之间的参数更加平滑。

Unbounded Scenes

比较惊喜的是 Plenoxel 也处理了 Unbounded Scene 的情况，它的处理方法和 NeRF++ 较为相似，用两个 sparse grid 分别处理内和外的情况；处理外部区域的 grid 利用等距常数投影 (Equirectangular Projection) 对应到了单位球上，而处理内部区域的 grid 本身在初始化的过程中就只关注单位球内的区域，这时候 Plenoxel 和 NeRF++ 的空间划分方式就一致了。这使得空间从欧式空间（Euclidean space）转化为了视差空间（Parallax space），有效处理了远距离场景中的光线变化。

对于外部区域则利用了多球体图像 (Multi-Sphere Images) 实现，直观来说就是将外部区域分成了多层（64层）球壳，越往外的球壳越大，最外一层球壳接近无穷（取倒数起到的作用，欧式空间转到了视差空间），不同层之间可以进行插值，从而实现了外部的三线性插值。对于外部区域的 appearance，Plenoxel 仅保留了 RGB 值，也就是背景部分的 RGB 不再是 view dependent,这意味着背景部分的颜色变化不随着观察角度的变化而变化，类似于传统的贴图（texture mapping）。这简化了外部区域的处理，因为背景在大多数情况下可以认为是静态的。同时作者在处理背景的时候，还考虑了引入 beta 分布来正则 σ 的优化：

$$\mathcal{L}_{\beta} = \lambda_{\beta} \sum_{r} (\log(T_{FG}(r)) + \log(1 - T_{FG}(r)))$$

其中 r 是训练的光线，$T_{FG}(r)$是光线r积累的前景透射率。很直观，即一条光线要么只看到前景，要么只看到背景。正则化项确保了这种二元性，减少了光线在前景和背景之间的模糊不确定性。

Others

在实际的使用中，Plenoxel 可能并不是很好用。一方面，explicit 设计实现不了一种全局的优化，很容易陷入到局部最优解（网格顶点间的特征是孤立的），产生撕裂效果的失真。与之相比，大家还是更倾向于 Hybrid 方案，用离散的特征存储结合小型 MLP 的方式，实现存储和速度还有效果的权衡。