Learning Gaussian Splatting

在学习高斯之前，因为缺乏相关的图形学基础，需要恶补一下之前有关的三维重建和图形学的知识，所以做了这样一个文档

传统场景重建和渲染

Light fields (1996) -> Structure-from-Motion (SfM，2006) -> multi-view stereo (MVS，2007)

- MVS-based 方法会重投影（re-project）并融合（blend）输入图片到新相机视角，使用几何引导重投影。
- 当 MVS 生成不存在的几何时，算法通常无法从“未重建”或“过度重建”的区域中完全恢复。

深度学习

DeepStereo（2016）->DeepBlending (2018，根据输入图像权重进行新视角合成)-> Free View Synthesis (2020，纹理层次的新视角合成)

- 大多数这些方法使用基于 MVS 的几何结构是一个主要缺点; 此外，使用 CNN 进行最终渲染经常导致时间闪烁。

神经体积表示法

Soft 3D (2017，首次提出 Volumetric representations) ->将深度学习技术应用到 volumetric ray-marching（2019）-> NeRF (2020，提出 importance sampling 和 postional encoding，但是较大 MLP 影响了速度) -> MipNeRF 360（2022，sota 视觉效果，但训练和渲染时间太长）

- 但由于查询体积需要大量样本，使用体积光线行进（volumetric ray-marching）进行渲染的成本很高。

为了加速训练或渲染，现有的探索集中在三个方向：稀疏数据结构存储特征，不同的编码器和 MLP 容量。其中，值得一提的方法是 InstantNGP（2022），该方法使用 hash grid 和 occupancy grid 来加速计算，用一个较小的 MLP 表示密度和外观。Plenoxels（2022），使用稀疏体素网格，插值表示连续密度场，并且能够不使用 MLP。

- 两者都依赖于球面谐波：前者直接表示方向效果，后者对其输入进行编码到色彩网络。
- 尽管两者都提供了优异的结果，但这些方法仍然可能在一定程度上难以有效表示空间，这在某种程度上取决于场景/捕获类型。此外，图像质量在很大程度上受到用于加速的结构化网格的选择的限制，并且渲染速度受到在给定射线行进步骤中查询许多样本的需求的阻碍。

简单的来理解神经体式，因为nerf是连续的神经体积表达方式，所以造成了计算量大的问题，为了解决这个问题有人采用稀疏体素网格。也就是离散的神经体积。

神经体积和传统3D场景表示方法不同的地方在于，网格存储的信息不再是颜色等信息，而是特征向量，可以通过神经网络来对整个场景进行预测和复原。

基于点的渲染和辐射场

基于点的方法可以有效地渲染不连续和非结构化的几何样本（即点云）

点样本渲染（1998，对固定大小的非结构点进行光栅化）-> 椭圆、圆形点渲染（2005，将点扩展为圆形/椭圆）-> 可微分的基于点的渲染技术（2020，点被神经特征增强、使用 CNN）

- 但是他们仍然依赖 MVS 来获得初始几何形状，并因此继承了缺点，尤其是在特征较少/发亮区域或薄结构等硬性情况下过度或者不足的重建。
    - -blending 和体渲染（volumetric rendering）本质上是相同的成像模型。
- 体渲染：

$$C = \sum_{i=1}^{N} T_i (1 - \exp(-\sigma_i \delta_i)) \mathbf{c}_i \quad \text{with} \quad T_i = \exp\left(- \sum_{j=1}^{i-1} \sigma_j \delta_j \right)$$

$$C = \sum_{i=1}^{N} T_i \alpha_i \mathbf{c}_i,$$

- -blending：

  $$C = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)$$

  然而，渲染算法是非常不同的。
- 神经辐射场（NeRFs）是一种连续表示，隐含地表示空/占用空间；需要昂贵的随机抽样来找到样本，进而产生噪声和计算开销。
- 点是一种非结构化的离散表示，足够灵活，可以允许创建、销毁和移动类似于 NeRF 的几何体。这是通过优化不透明度和位置来实现的
    - 完整体积表示的缺点：**计算开销大、训练时间长、空间利用率高**
- 在《Neural Point Cloud Rendering via Multi-Plane Projection》中提到，
    - 该论文提出了一种基于多平面投影的神经点云渲染方法，能够从稀疏的点云中合成高质量的新视图。
    - 该方法将点云投影到多个平行的平面上，每个平面上的点使用一个小型的多层感知器（MLP）来预测颜色和透明度。然后，使用可微的光栅化（rasterization）和深度融合（depth fusion）来生成最终的图像。

- 该方法能够有效地利用点云的几何信息，避免了在空白空间中进行不必要的计算。同时，该方法能够通过优化点的位置和透明度，实现点云的动态更新，从而适应不同的场景和视角。此外，该方法能够处理视角依赖的外观变化，例如镜面反射和透明度。

3D 高斯

在人体性能捕捉领域，已经使用 3 D 高斯函数来表示捕捉的人体->最近它们已经与体积光线行进一起用于视觉任务->提出了神经体积原语。

- 它们专注于重建和渲染单个孤立对象（人体或面部）的特定情况，导致了具有小深度复杂性的场景。
- 相比之下，我们的各向异性协方差优化、交替优化/密度控制以及高效深度排序用于渲染，使我们能够处理完整的、复杂的场景，包括室内和室外的背景，并具有较大的深度复杂性。

各向异性的三维高斯函数的引入，作为辐射场的高质量、非结构化表示

我们将几何建模为一组不需要法线的 3 D 高斯函数。我们的高斯函数由在世界空间中定义的完整 3 D 协方差矩阵 Σ 和以点（均值）μ 为中心来定义：$G(x) = e^{-\frac{1}{2}(x)^T \Sigma^{-1}(x)}$

我们需要将我们的三维高斯函数投影到二维进行渲染。给定一个视图变换 W，协方差矩阵 Σ' 在摄像机坐标下可以表示如下：

$$\Sigma' = JW\Sigma W^T J^T$$

J 是射影变换的仿射逼近的雅可比矩阵。如果我们跳过 Σ' 的第三行和列，则我们将得到一个 2×2 的方差矩阵，具有与我们从具有法线的平面点开始时相同的结构和性质。（表示 X, Y 平面 3 D 高斯的分布）

其中，可以将协方差矩阵表示为

$$\Sigma = RSS^T R^T$$

一：基础公式

一开始在简单的了解高斯splatting的工作原理之后，简单的理解就是把高斯分布的椭球型直接扔到2D，但是为什么一定要高斯分布，以及如何得到不透明度，渲染依然是一个问题，所以打算从数学源头开始，把高斯splatting的原理弄明白。

1：高斯分布

$$p(x_1,\cdots,x_n)=\frac{1}{(2\pi)^{\frac{n}{2}}\left|\Sigma\right|^{\frac{1}{2}}}\cdot e^{-\frac{1}{2}\cdot[(\vec{X}-\vec{\mu})^\top\Sigma^{-1}(\vec{X}-\vec{\mu})]}$$

多元的高斯公式，其中μ是均值向量（每个维度均值所得）

一维高斯公式如下：

$$p(x)=\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-u)^2}{2\sigma^2}}$$

现在，我们考虑 n 个服从正态分布且互不相关的独立变量

现在，我们考虑 n 个服从正态分布且互不相关的独立变量 x=，其均值为 $x=[x_1,x_2,\ldots,x_n]^{\mathrm{T}}$ ，其均值为 $E(x)=[u_1,u_2,\ldots,u_n]^{\mathrm{T}}$, 其方差为 $\sigma(x)=[\sigma_1,\sigma_2,\ldots,\sigma_n]^{\mathrm{T}}$,

其概率密度函数为：

$$f(x)=p(x_1,x_2\ldots x_n)=p(x_1)p(x_2)\ldots p(x_n)$$
$$=\frac{1}{(\sqrt{2\pi})^n\sigma_1\sigma_2\cdots\sigma_n}e^{-\frac{(x_1-\mu_1)^2}{2\sigma_1^2}-\frac{(x_2-\mu_2)^2}{2\sigma_2^2}\cdots-\frac{(x_n-\mu_n)^2}{2\sigma_n^2}}$$

令 $z^2=\frac{(x_1-\mu_1)^2}{\sigma_1^2}+\frac{(x_2-\mu_2)^2}{\sigma_2^2}\cdots+\frac{(x_n-\mu_n)^2}{\sigma_n^2}$,$\sigma_z=\sigma_1\sigma_2\cdots\sigma_n$

简写上面的式子得到：

$$f(z)=\frac{1}{(\sqrt{2\pi})^n\sigma_z}e^{-\frac{z^2}{2}}$$

同时由于:

$$z^2 = z^{\mathrm{T}} z$$

$$= [x_1 - \mu_1, x_2 - \mu_2, \cdots, x_n - \mu_n] \begin{bmatrix} \frac{1}{\sigma_1^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2^2} & \cdots & 0 \\ \vdots & \cdots & \cdots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sigma_n^2} \end{bmatrix}$$

$$[x_1 - \mu_1, x_2 - \mu_2, \cdots, x_n - \mu_n]^{\mathrm{T}}$$

即：

$$[x - \mu_x]^{\mathrm{T}} \begin{bmatrix} \frac{1}{\sigma_1^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2^2} & \cdots & 0 \\ \vdots & \cdots & \cdots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sigma_n^2} \end{bmatrix} [x - \mu_x]$$

$$z^{\mathrm{T}} z = (x - \mu_x)^{\mathrm{T}} \sum{}^{-1} (x - \mu_x)$$

带入可以得到最终的结果：

$$f(z) = \frac{1}{(\sqrt{2\pi})^n \sigma_z} e^{-\frac{z^2}{2}} = \frac{1}{(\sqrt{2\pi})^n |\sum|^{\frac{1}{2}}} e^{-\frac{(x-\mu_x)^{\mathrm{T}} (\sum)^{-1} (x-\mu_x)}{2}}$$

去掉了指数部分前面的尺度系数（不影响椭球几何）；默认模型坐标中心在原点，方便旋转放缩，放入世界空间时再加上平移即可得到原式。

为了防止点渲染时不连续造成的空洞问题，需要对每个点初始化椭球。

但是，怎么对每个点初始化椭球是个随机的方法，而原文中自适应地生成新椭球是非常重要的一环，随机生成几个数填入协方差矩阵 Σ是不行的，每个3D高斯椭球都对应一个 Σ，但不是每个 Σ 都能对应一个椭球，需要 Σ 半正定。

由于 3 D 高斯和椭球是一一对应的，所以可以对球进行缩放成椭球，然后将其移动到点的位置从而将点云初始化成椭球，于是初始化方法是使用放缩变换 S 和旋转变换 R组合得到 Σ：

$\Sigma = RSS^T R^T$ 由于放缩变换都是沿着坐标轴，所以只需要一个3D 向量 S ，旋转则用四元数 q 表达。

同时机器学习通常使用梯度下降对参数进行优化，但直接优化 Σ 难以保证半正定，所以论文的方法是继续将梯度传递到 S, q 进行优化。

协方差矩阵

下面这个是协方差矩阵的公式

$$\Sigma_{ij} = \frac{1}{N-1}\sum_{k=1}^{N}(x_{ki}-\mu_i)(x_{kj}-\mu_j)$$

二维的情况，可以这样简化

$$\Sigma = \begin{pmatrix} \text{方差}(x_1) & \text{协方差}(x_1,x_2) \\ \text{协方差}(x_1,x_2) & \text{方差}(x_2) \end{pmatrix}$$

方差可以用来表示离散程度，协方差用来表示相关性

协方差是一种描述随机变量之间的相关性的矩阵，它的对角线元素表示了各个变量的方差，而非对角线元素表示了各个变量之间的协方差。方差是一种度量变量的离散程度的指标，而协方差是一种度量两个变量的线性相关程度的指标。各向异性协方差是指协方差矩阵不是对角阵或者单位阵乘以常数，而是有非零的非对角线元素，这意味着变量之间的相关性不是各个方向上相同的，而是有一定的方向性。各向异性协方差可以用来表示复杂的非线性关系，例如场景的辐射场

这种表示是指用3D 高斯函数来表示场景的几何和外观，每个高斯函数由一个中心点（均值）、一个协方差矩阵和一个透明度（opacity）参数来定义。这种表示适合优化，因为它可以通过梯度下降法来调整高斯函数的参数，从而使得渲染的图像与输入的图像尽可能接近。这种表示会产生紧凑的表示，因为它可以用少量的高斯函数来近似复杂的表面，而且可以利用协方差矩阵来描述表面的曲率和方向，从而避免了需要法线和半径等额外的信息。

> 高斯分布的协方差Σ 是正定矩阵，一定可以进行对角化，因此可以分解成如下形式
>
> $$\Sigma = RSS^TR^T$$
>
> R通过四元数表示，4个参数，S为对角矩阵，3个参数，所以协方差一共7个参数

协方差矩阵可以被对角化，也就是可以被特征值分解，因此可以得到

S是用来拉伸，R是用来旋转，形式一定要对称

为什么能联想到旋转和放缩矩阵呢？

由于协方差矩阵是是对称矩阵，所以进行特征值分解得到： $\Sigma = P \wedge P^T$ ，进而得到：$\Sigma = (P\Lambda^{1/2})(P\Lambda^{1/2})^T$,其中, P 是一个正交阵，可以理解为是一个旋转矩阵， $\Lambda^{1/2}$ 为一个全为正数的对角阵，可以理解为是一个缩放矩阵

（理解这里需要重新回想起一些线代的知识）

为什么 3 D 高斯可以与椭球相对应?

我们想像一下固定概率为 p 的三维高斯分布长什么样。

$$\frac{1}{(2\pi)^{\frac{N}{2}}|\Sigma|}e^{-\frac{1}{2}(x-u)^T\Sigma^{-1}(x-u)} = p$$

取对数得：

$$-ln(2\pi)^{\frac{N}{2}}|\Sigma| - \frac{1}{2}(x-u)^T\Sigma(x-u) = lnp$$

$$(x-u)^T\Sigma(x-u) = -2lnp - 2ln(2\pi)^{\frac{N}{2}}|\Sigma = C$$

$$\Sigma = P^T \wedge P$$

$$(x-u)^T P^T \wedge P(x-u) = (P(x-u))^T \wedge P(x-u) = C$$

令 $y = P(x-u)$ 得到椭球方程

$$y^T \wedge y = \sigma_1^2 y_1^2 + \sigma_2^2 y_2^2 + \sigma_3^2 y_3^2$$

原式子就是把标准椭球 x 先经过经过一个旋转变换 P^{-1}，再挪到 u 点得到

从几何的角度来理解：

对于一维的高斯来说，μ就是中心点，α控制了形状

对于二维的高斯来说，μ是圆心，他本身是一个圆，xy各自是一个高斯分布，也是用Σ来控制形状

那么对于三维的高斯也是同理的，μ代表了位置，Σ控制形状

2：三维投影到二维（渲染 splatting的过程）

$$y = p_c = W_w^c p_w + t_w^c$$

Pc是相机坐标

Pw是世界坐标

W是旋转，t是位移，将世界坐标位移到相机坐标（玩过blender的应该方便理解，我们渲染的都是那个摄像头的东西）

$$\Sigma_{p_c} = W \Sigma_{p_w} W^T$$

协方差矩阵的转化：（很多时候都需要加个转置）

相机坐标系压扁得到图像（二维的）：

从相机坐标系变换到像素坐标系：

$$z = \begin{bmatrix} z_x \\ z_y \\ 1 \end{bmatrix} = (1/p_c^z) \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_c^x \\ p_c^y \\ p_c^z \end{bmatrix}$$

这个变换是非线性的，需要进行一阶泰勒展开，$z = F(p_c) \approx F(\mu_y) + J(y - \mu_y)$,进而得到协方差变换为：

$$\Sigma_z = J \Sigma_{p_c} J^T = JW \Sigma_{p_w} W^T J^T$$

均值如下：

$$\mu_z = F(\mu^y) = F(W \mu_{p_w^c} + t)$$

这里之所以要展开，是因为在从相机坐标到二维坐标的时候，做了一个除以z的操作，这个操作并不是线性变化。高斯分布有一个很好的性质，对高斯变化进行线性变化依然是高斯分布：利用这个性质可以快速计算协方差矩阵和均值向量，因此我们想要用一节泰勒对细微处来展开，保持线性的性质

相机坐标的相关知识：Derivation of projection and transformation

接下来，我们要把 3 D 高斯投影到二维平面上完成渲染的过程，完成从3D 到2D 图像的光栅化过程，需要将3D 高斯投影到屏幕坐标系，这个过程需要世界到相机的变换矩阵 W，及相机到屏幕空间的映射矩阵 P。其中，W 矩阵是旋转和平移，是放射的，但是 P 投影不是仿射的，为了线性化表示，所以用 J 代替了 P。

这里的投影采用了跟之前 NDC 坐标有点类似的方法，使用“挤压”的方式，但是不是透视投影或者正交投影。 相机坐标(t_1, t_2, t_3)压缩到(x_1, x_2, x_3)上。

$$x_1 = \frac{t_1}{t_3}$$

$$x_2 = \frac{t_2}{t_3}$$

$$x_3 = \sqrt{t_1^2 + t_2^2 + t_3^2}$$

这里 设计成是因为在基于点的着色时，需要考虑一个射线出去的点的前后顺序，所以需要对相应位置进行保留。但是比较遗憾的是，这并不是一个线性变换，也就是无法通过求解一个变换矩阵把这个变

换求解出来。但是我们可以使用一个线性变换对这个非线性变换进行近似。

这里就需要啊用到对应的雅可比矩阵

雅可比矩阵就可以简单的理解为高维的泰勒展开，也就是非线性化到线性化的过程，让我们可以得到对应的线性变化矩阵：

这里 J 雅可比矩阵，可以看来一阶导的多维形式，假设我们输入为 ：$x_1, x_2, ...x_n$,输出为 $y_1, y_2, \cdots, y_n$

$$J = \begin{bmatrix} \frac{\partial y_1}{x_1} & \frac{\partial y_1}{x_2} & \cdots & \frac{\partial y_1}{x_m} \\ \frac{\partial y_2}{x_1} & \frac{\partial y_2}{x_2} & \cdots & \frac{\partial y_2}{x_m} \\ \cdots & & & \\ \frac{\partial y_n}{x_1} & \frac{\partial y_n}{x_2} & \cdots & \frac{\partial y_n}{x_m} \end{bmatrix}$$

前面的非线性变换，使用这种方式进行近似的话，其雅可比矩阵为：

$$\begin{bmatrix} \frac{1}{t_3} & 0 & -\frac{t_1}{t_3^2} \\ 0 & \frac{1}{t_3} & -\frac{t_2}{t_3^2} \\ \frac{t_1}{\sqrt{t_1^2+t_2^2+t_3^2}} & \frac{t_2}{\sqrt{t_1^2+t_2^2+t_3^2}} & \frac{t_3}{\sqrt{t_1^2+t_2^2+t_3^2}} \end{bmatrix}$$

3：球谐函数

在高斯 splatting 或 NeRF（神经辐射场）这样的技术中，球谐函数常用来表示光照（比如从不同方向照射的光）或物体的颜色变化。球谐函数被用来表示颜色 C(θ, ϕ)，θ 和 ϕ 是球面上的角度（就像经度和纬度）。

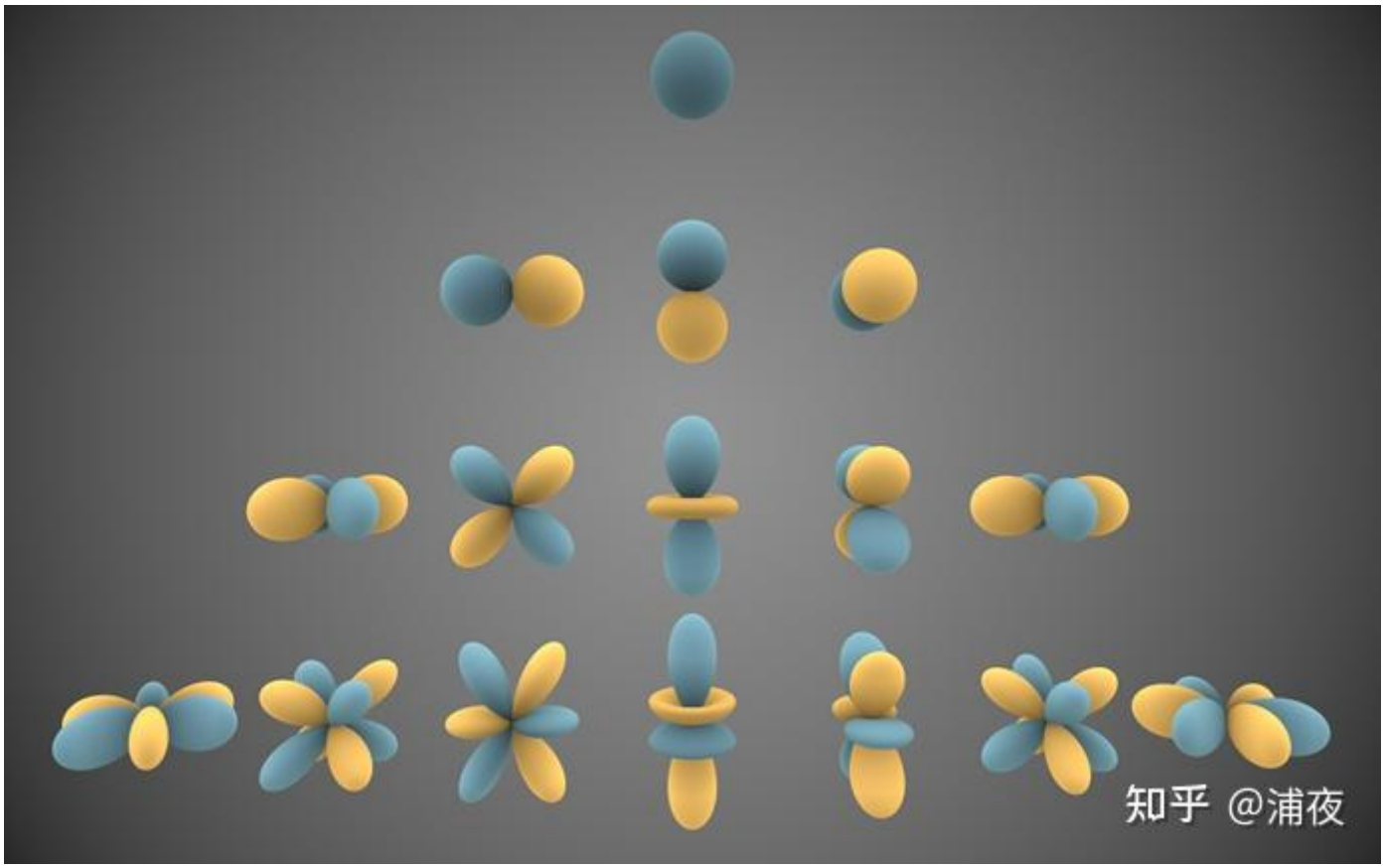
注意！！！！NERF是利用MLP来得到球谐函数

光照函数C (θ , ϑ)可以表示为球谐函数的加权线性组合，如下，某一个位置高斯球的函数，输入为角度，输出为这个角度的颜色

$$C(\theta, \phi) = \sum_{j=0}^{J} \sum_{m=-j}^{j} c_j^m Y_j^m(\theta, \phi)$$

球谐函数，Spherical Harmonics 本质上可以理解为在球坐标系下的基函数（有点类似基向量），这样只需要球谐函数，就可以用来表示所有的球坐标系下的函数，来描述光照的关系

他们对应的系数，就是球谐系数

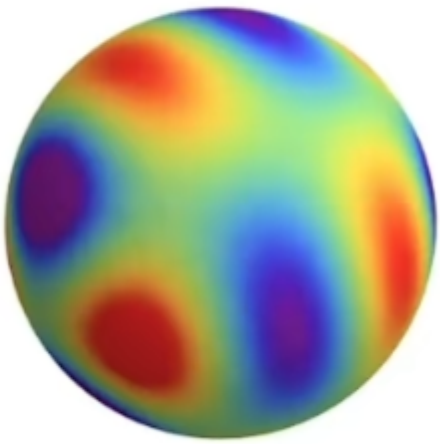


具体的公式我还没有看懂，先粗略的理解下

球谐函数与颜色：

球谐函数来表达RGB，一种颜色用一个球谐函数的加权来得到的，球谐函数的阶数可以理解成直线，二次函数，三次函数一样的阶数，当球谐函数为四阶的时候，从上图可以看到，我们总共需要16种不同的球谐函数，RGB每个颜色一个，因此需要48个参数来确定。

有了球谐函数，可以帮助我们从不同的角度看得到不同的颜色，因为是一个球体，每个方向都有着色，这是求谐函数的结果



优化的过程，很大的一部分就是在优化这个

InstantNGP中的球谐函数

InstantNGP 使用了一个哈希网格和一个占据网格来加速计算，以及一个较小的多层感知器（MLP）来表示密度和外观。它直接使用球形谐波来表示每个哈希网格单元中的方向效果，即每个网格单元存储一个球形谐波的系数，用于描述该位置的辐射值随方向的变化。这样，它可以直接从球形谐波中采样出任意方向的辐射值，而无需额外的网络或插值。

4 splatting and blending

Splatting：把 3D 高斯“云”投影到 2D 屏幕上，生成像素（像把 3D 物体“拍扁”成 2D 图像）。

α blending：把多个 2D 像素层叠加（混合），生成最终的画面（像叠加多张透明胶片）。

$$I_{result} = I_1 * \alpha_1 + I_{BK} * (1 - \alpha_1)$$

这个有点像图像融合，就是直接利用透明度来对图像融合，阿尔法对对应的权重

多图的时候可以扩展到这个公式：α就是不透明度

不透明度的获取还需要了解

$$\begin{aligned} I_{result} &= I_1 \times \alpha_1 + (1-\alpha_1)(I_2 \times \alpha_2 + (1-\alpha_2)(I_3 \times \alpha_3 + I_{BK} \times (1-\alpha_3))) \\ &= \alpha_1 I_1 + (1-\alpha_1)\alpha_2 I_2 + (1-\alpha_1)(1-\alpha_2)\alpha_3 I_3 + (1-\alpha_1)(1-\alpha_2)(1-\alpha_3) I_{BK} \end{aligned}$$

如果Ibk=0，公式可以在被简化

$$C = \sum_{i \in N} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)$$

传统 α blending：之前我们讲的，把多层 2D 图像按透明度叠加（像叠透明胶片）。

NeRF 体渲染：NeRF（神经辐射场）用体视显微镜的思路，直接计算 3D 空间中的光线积分，得到颜色。

$$C = \sum_{i=1}^{N} T_i (1 - \exp(-\sigma_i \delta_i)) \mathbf{c}_i \quad \text{with} \quad T_i = \exp\left(- \sum_{j=1}^{i-1} \sigma_j \delta_j \right)$$

$$C = \sum_{i=1}^{N} T_i \alpha_i \mathbf{c}_i \qquad \alpha_i = (1 - \exp(-\sigma_i \delta_i)) \text{ and } T_i = \prod_{j=1}^{i-1} (1 - \alpha_i)$$

nerf可以理解为计算一条光纤上N层

总结：高斯分布决定了 splat的形状，而球谐函数决定了splat的颜色，两者最后投影为2d的过程， 就是splatting。

而且球谐函数会随着时机哦啊发生变化

二：实验流程

这是论文中的高斯整体流程

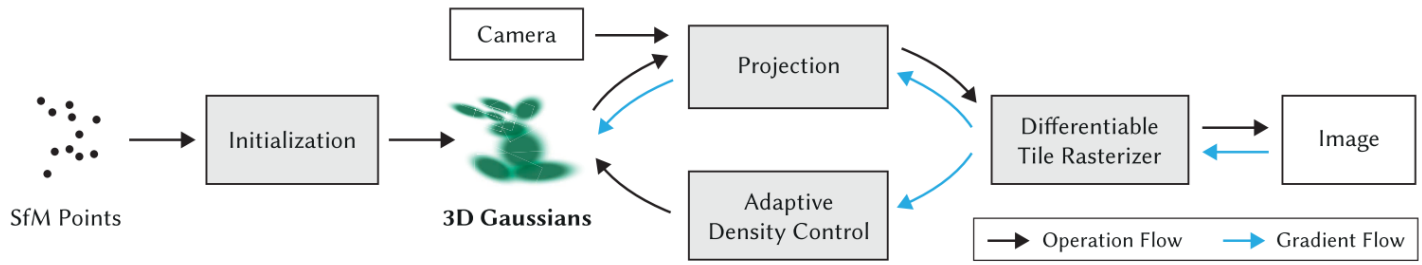


Fig. 2. Optimization starts with the sparse SfM point cloud and creates a set of 3D Gaussians. We then optimize and adaptively control the density of this set of Gaussians. During optimization we use our fast tile-based renderer, allowing competitive training times compared to SOTA fast radiance field methods. Once trained, our renderer allows real-time navigation for a wide variety of scenes.

1：输入数据 初始化：

把点云输入进去，可以通过colmap得到（也就是你输一组照片，可以用colmap得到点云），在点云的位置初始化3d gaussian

当然随机初始化感觉也是可以的，吃屎colmap可以快速重建相机的位置，帮助收敛

2： projection

要把这些高斯球全部投影到2d上面，同时根据camera来进行调整，这是投影的公式（前面有提到过）

$$G(x) = e^{-\frac{1}{2}(x)^T \Sigma^{-1}(x)}$$

$$\Sigma = RSS^T R^T$$

$$\Sigma' = JW\Sigma W^T J^T$$

3：differentiable tile rasterizer

简单的来说就是光栅化，在投影到2d平面之后，利用前面讲到的blending来得到投影

- 将屏幕划分为 16 x 16 块，一次性预排序所有 primitives，避免针对每个像素排序的开销
- 仅保留有 99%置信度的每个 tile 视锥内可视的 Gaussians（用人话来理解，就是判读这个高斯点是不是在相机的视野里面啊，我们在考虑是否需要进行渲染）
- 用保护带（guard band）拒绝极端位置的高斯，例如：距离平面太近，或在视锥体之外太远的高斯；（因为需要考虑一个点是否是在边缘地带产生的不好的效果，简单的来说，就是有一个缓冲地带）
    - 当一个高斯斑点的均值接近近平面时，它的投影2D 协方差会变得非常大，因为它会被放大到屏幕上。这会导致计算不稳定，可能出现数值溢出或者奇异矩阵的情况。当一个高斯斑点的均值远离视锥体时，它的投影2D 协方差会变得非常小，因为它会被缩小到屏幕上。这也会导致计算不稳定，可能出现数值下溢或者奇异矩阵的情况。因此，这些极端位置的高斯斑点需要被剔除，以避免不必要的计算和错误。
- 将 3D 高斯实例化为高斯对象，包括视觉空间深度和块 ID。并按对象中包含的深度信息排序。（实例的高斯就是包含了高斯的信息）
- 在光栅化过程中，每个块使用一个线程去处理，每个线程首先将 Gaussians 加载到 shared memory。对于给定像素点，从前往后遍历列表，根据 α 加权求和颜色。当 α 达到目标饱和值时，该进程停止，得到该像素点的颜色。（总共有16 *16的块，单独对每个块使用一个线程，根据每个高斯点的深度进行加权求和计算）
- 我们不限制接受梯度更新的混合基元的数量。我们强制执行这一特性，是为了让我们的方法能够处理具有任意不同深度复杂度的场景 （混合基元可以理解为有颜色的，有作用的点）
- 梯度回传：从最后一个点，从后往前遍历列表，选择对应的 Gaussian 反传梯度。
- 我们可以只存储前向传递结束时的总累积不透明度，而不是在后向传递中遍历一个逐渐缩小的不透明度的显式列表，从而恢复这些中间不透明度。具体来说，每个点在前向过程中存储最终累积的不透明度 α；我们在后向遍历中将其除以每个点的 α，以获得梯度计算所需的系数。

4：与gt来计算loss，更新高斯球的参数

这里需要提到损失函数：

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}}$$

损失函数里面有两个新的参数：

首先，基于像素来比较差异

$$L^{l_1} = \frac{1}{N}\sum_{p \in P} |x(p) - y(p)|$$

还有一个D-SSIMloss

5：梯度更新

根据前面说的，来更新高斯球

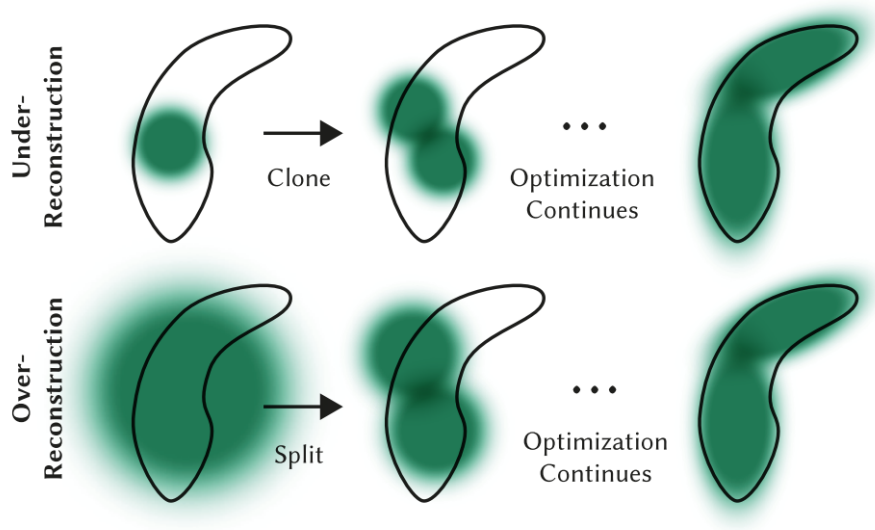


Fig. 4. Our adaptive Gaussian densification scheme. *Top row (under-reconstruction)*: When small-scale geometry (black outline) is insufficiently covered, we clone the respective Gaussian. *Bottom row (over-reconstruction)*: If small-scale geometry is represented by one large splat, we split it in two.

6：实验结果

- **warm-up**。开始优化时使用的图像分辨率是原来的 4 倍，并在迭代 250 次和 500 次后进行两次上采样。
- **SH 系数优化**对于缺乏角度信息非常敏感（即，非 object-center 的场景）。因此优化时，首先仅优化零阶分量，然后每 1000 次迭代引入一组 SH 频带，直到表示所有 4组 SH 频带。
    - SH 系数优化的一个问题，就是它对于角度信息的缺乏很敏感。如果场景的拍摄角度是均匀分布的，比如在一个物体周围的半球上拍摄，那么优化效果很好。但是如果场景的拍摄角度有缺失的区域，比如只拍摄了场景的一个角落，或者使用了“内外”的方法，那么优化就会得到错误的结果，特别是对于 SH 的零阶分量，也就是场景的基本颜色或者漫反射颜色。
- 分割大高斯对于实现背景的良好重建非常重要。而克隆小高斯而不是分割它们可以实现更好更快的收敛，尤其是当场景中出现薄结构时。

三：论文实验复现

环境部署

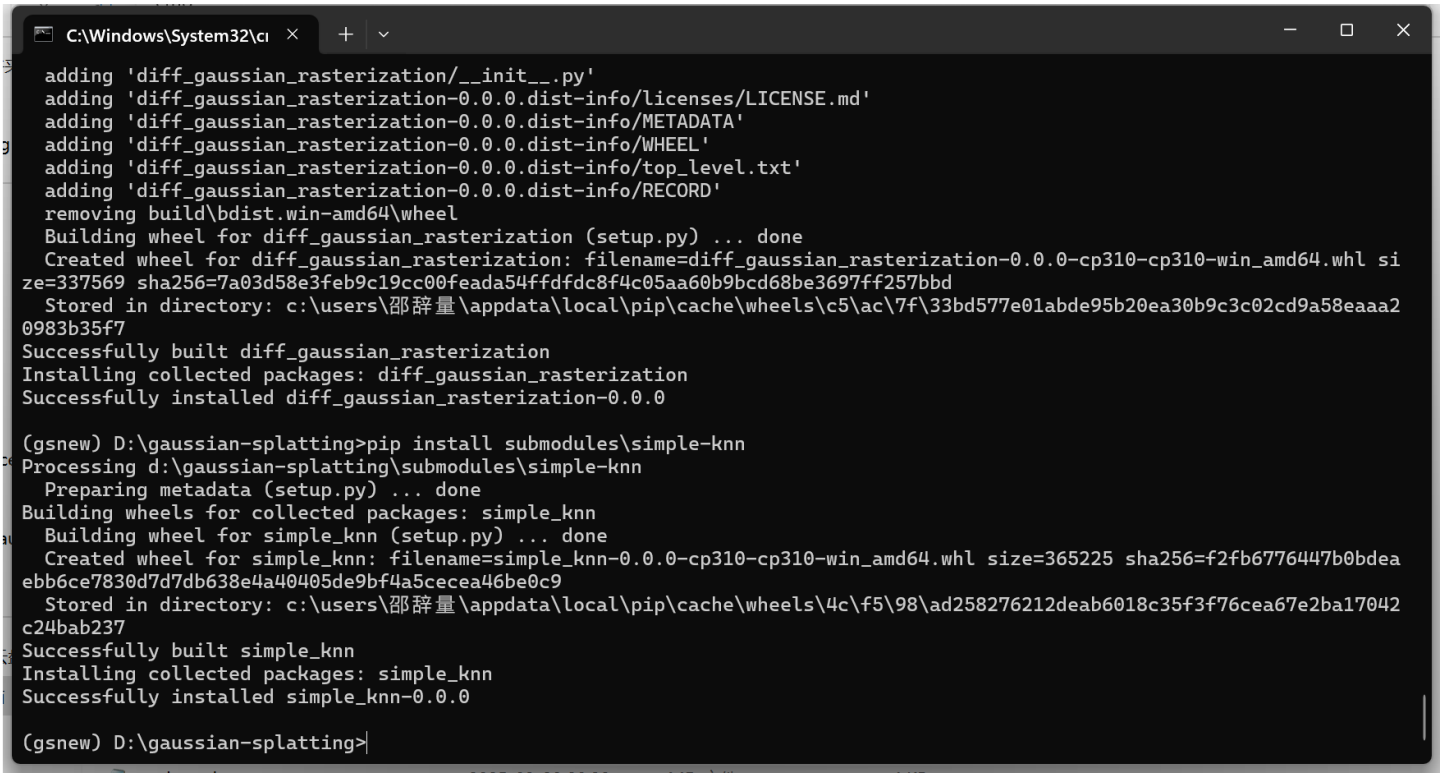
环境部署花了大概两天左右的时间，遇到了几个比较大的问题记录一下：

首先就是cuda和pytorch的版本不对应的问题，这个创建虚拟环境直接解决了（需要上官网查对应的型号）

其次就是visual code studio的2019社区版本已经了，通过查issue找到了 professional 版本解决了问题

最后这些都解决了，但还是依然报错，这个问题查了很久，重复了各种各样的方法，最后发现是wheel在编译的时候被放进了一个中文的路径，导致无法识别到。

解决方法就是创建了一个临时文件来存放，识别到了就可以把所有的库安装成功了



四：代码

具体细节建这个文档：3D高斯代码

参考文献：

- 论文链接：https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/
- 论文解读资料：
  - https://blog.csdn.net/m0_37604894/article/details/137864723
  - https://zhuanlan.zhihu.com/p/680669616
  - https://space.bilibili.com/3494380627299296/lists/2116610?type=season
  - https://blog.csdn.net/weixin_42973508/article/details/141782282
  - https://blog.csdn.net/chien__/article/details/144124323

深度学习

- Flynn et al. [2016]是一篇关于利用深度学习技术进行新视角合成的论文 。它的主要贡献是提出了一种基于深度图的新视角合成方法，称为 DeepStereo。DeepStereo 使用一个卷积神经网络（CNN）来预测给定一组输入图像和一个目标视角时，目标视角下的深度图和颜色图。它的优点是可以处理复杂的场景，包括遮挡、反射和透明度，并且可以生成高质量的新视角图像。它的缺点是需要大量的训练数据，以及较高的计算成本。

- Hedman et al. [2018]是一篇关于利用 CNN 来估计混合权重的论文 。它的主要贡献是提出了一种基于 CNN 的新视角合成方法，称为 DeepBlending。DeepBlending 使用一个 CNN 来预测给定一组输入图像和一个目标视角时，每个输入图像在目标视角下的混合权重。然后，它使用这些混合权重来对输入图像进行加权平均，从而生成目标视角下的新图像。它的优点是可以处理动态的场景，以及具有不同曝光和白平衡的输入图像，并且可以生成连续和自然的新视角图像。它的缺点是需要一个粗略的几何模型作为先验，以及对于一些极端的视角变化，可能无法生成合理的结果。

神经体积表示法

- [Penner and Zhang 2017]是一篇关于利用 CNN 进行体积表示的论文 。它的主要贡献是提出了一种基于体积表示的新视角合成方法，称为 Soft 3 D Reconstruction。Soft 3 D Reconstruction 使用一个 CNN 来预测给定一组输入图像和一个目标视角时，目标视角下的体积密度和颜色。然后，它使用一个可微的体积渲染层来对体积进行光线追踪，从而生成目标视角下的新图像。它的优点是可以处理任意分布的输入视角，以及具有复杂几何和外观的场景，并且可以生成连续和自然的新视角图像。它的缺点是需要一个大量的训练数据，以及对于一些具有遮挡或反射的场景，可能无法生成准确的结果。
- Henzler et al. [2019]是一篇关于利用深度学习技术和体积光线追踪的论文 。它的主要贡献是提出了一种基于体积光线追踪的新视角合成方法，称为 PlatonicGAN。PlatonicGAN 使用一个生成对抗网络（GAN）来预测给定一组输入图像和一个目标视角时，目标视角下的体积密度和颜色。然后，它使用一个可微的体积光线追踪层来对体积进行光线追踪，从而生成目标视角下的新图像。它的优点是可以处理任意分布的输入视角，以及具有复杂几何和外观的场景，并且可以生成高质量的新视角图像。它的缺点是需要一个大量的训练数据，以及对于一些具有遮挡或反射的场景，可能无法生成准确的结果。

快速光栅化

- [Lassner and Zollhofer 2021] 是一篇论文的引用，论文的全称是《Pulsar: Learning to Render 3D Scenes with Points and Spheres》。该论文提出了一种使用点和球体来表示和渲染三维场景的学习方法，称为 Pulsar。Pulsar 使用了一种快速的球体光栅化算法，可以在 GPU 上实现高效的前向和反向传播，同时保持高质量的视觉效果。
- [Rückert et al. 2022] 也是一篇论文的引用，论文的全称是《ADOP: Adaptive Differentiable Object Primitives for 3 D Reconstruction and Synthesis》。该论文提出了一种使用可微分的对象原始体（Differentiable Object Primitives，简称 ADOP）来表示和重建三维场景的方法。ADOP 使用了一种自适应的密度控制策略，可以动态地调整对象原始体的数量和形状，以适应不同的场景复杂度和细节。

基于点的渲染和辐射场

- **神经点折射（Neural Point Catacaustics）[Kopanas et al. 2022]**：这是一种基于点的渲染方法，可以实现复杂的光线传输效果，如折射和反射。该方法使用一个 MLP 来表示场景的密度和颜色，以及一个 SH 系数来表示方向性外观。该方法的特点是可以处理镜面效果，但是需要 MVS 几何作为输入，即需要通过多视图立体视觉（Multi-View Stereo）技术重建场景的三维结构。

- **[Zhang et al. 2022]**：这是一种最新的基于点的渲染方法，不需要MVS几何作为输入，也使用SH系数来表示方向性外观。该方法的特点是可以处理单个物体的场景，但是需要用掩码（mask）来初始化场景的点云。该方法在分辨率和点数较低时速度较快，但是不清楚它是否能够扩展到典型的数据集[Barron et al. 2022; Hedman et al. 2018; Knapitsch et al. 2017]的场景。

3 D 高斯

- [Rhodin et al. 2015]：这是一种用于捕捉人体运动的方法，它使用3D 高斯来表示人体的表面和外观，然后用体积光线行进来渲染人体。这种方法可以从多个视角的视频中重建出人体的三维模型，并且可以在不同的光照和背景下合成新的视角。
- [Stoll et al. 2011]：这是一种用于捕捉多人互动的方法，它使用3D高斯来表示人体的表面和外观，然后用抛雪球（splatting）来渲染人体。这种方法可以从多个视角的视频中重建出多个人体的三维模型，并且可以在不同的场景下合成新的视角。
- [Wang et al. 2023]：这是一种用于视觉任务的方法，它使用3D高斯来表示场景的密度和颜色，然后用体积光线行进来渲染场景。这种方法可以从单个视角的图片中重建出场景的三维模型，并且可以在不同的视角下合成新的图片。
- [Lombardi et al. 2021]：这是一种用于重建和渲染单个物体的方法，它使用神经网络来生成一组体积原始体（volumetric primitives），然后用体积光线行进来渲染物体。这种方法可以从多个视角的图片中重建出物体的三维模型，并且可以在不同的光照和视角下合成新的图片。