

NeRF

NeRF

理论与论文学习

参考：

看完了，但是好像没怎么讲原理，不是很清晰：

【NeRF系列公开课01 | 基于NeRF的三维内容生成】 https://www.bilibili.com/video/BV1d34y1n7fn/?share_source=copy_web&vd_source=067de257d5f13e60e5b36da1a0ec151e

以下PPT来自：

【十分钟带你快速入门NeRF原理】 https://www.bilibili.com/video/BV1o34y1P7Md/?share_source=copy_web&vd_source=067de257d5f13e60e5b36da1a0ec151e

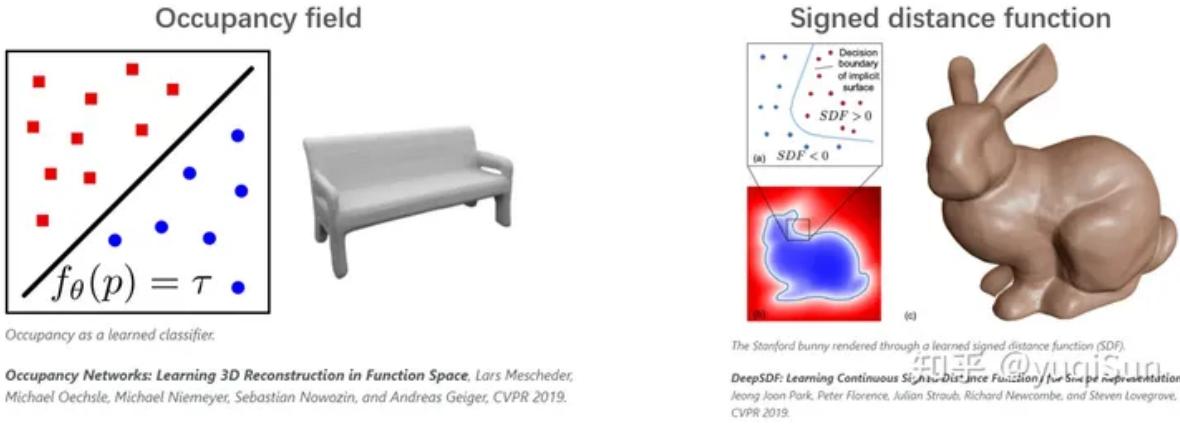
[NeRF简介哔哩哔哩bilibili](#)

1.1 背景

视角合成方法通常使用一个中间3D场景表征作为中介来生成高质量的虚拟视角。根据表示形式，3D场景表征可以分为“显式”和“隐式”表示。显式表示 (explicit representation)，包括Mesh, Point Cloud, Voxel, Volume等。显式表示的优点是能够对场景进行显式建模，从而合成照片级的虚拟视角。缺点是这种离散表示因为不够精细化会造成重叠等伪影，而且最重要的，它们对内存的消耗限制了高分辨率场景的应用。

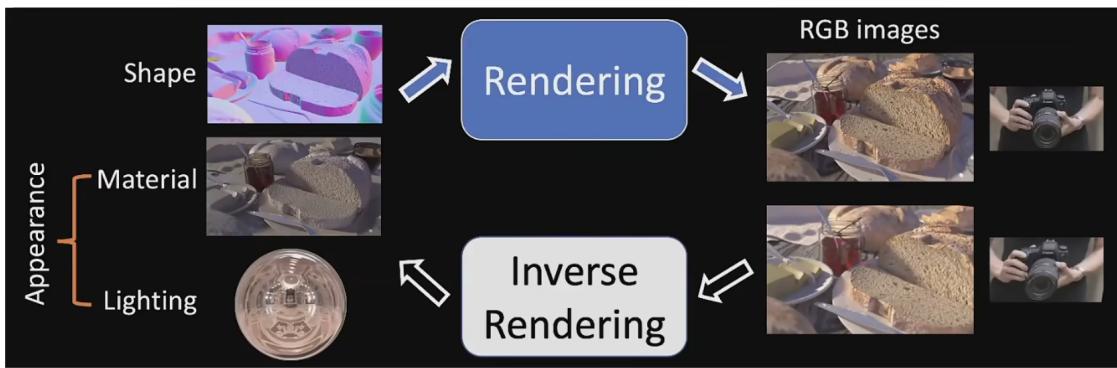


隐式表示 (implicit representation)，通常用一个函数来描述场景几何。隐式表示使用一个MLP模拟该函数，输入3D空间坐标，输出对应的几何信息。隐式表示的好处是它一种连续的表示，能够适用于大分辨率场景，而且通常不需要3D信号进行监督。在NeRF之前，它的缺点在于无法生成照片级的虚拟视角，如occupancy field、signed distance function (SDF)。



我们需要理解的是，无论是显式表示还是隐式表示，都是对3D场景进行表征。这种表征并不是凭空臆测或者天马行空的，而是根据现实中数据格式进行发展。例如现实中的3D数据主要有面数据、点数据、体数据，所以对应催生了一些Mesh、Point Cloud、Volume等中间表示。隐式表示则是借鉴了图形学中的一些表示形式，例如signed distance function。

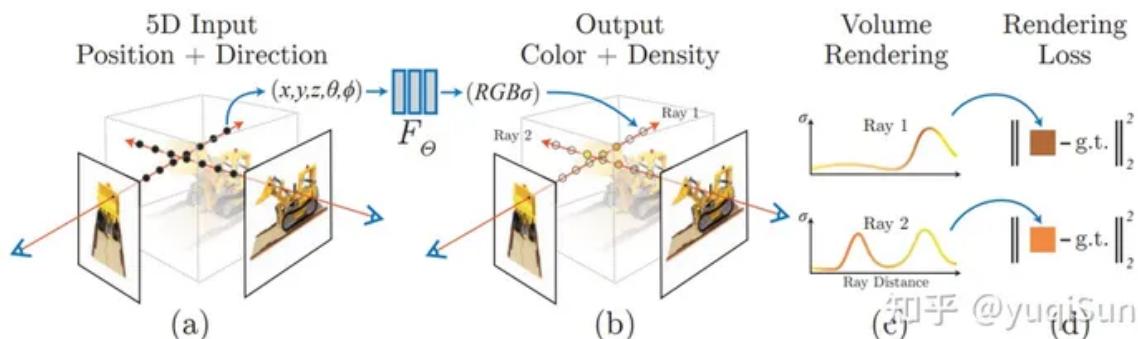
关于渲染与反渲染



1.2 NeRF方法

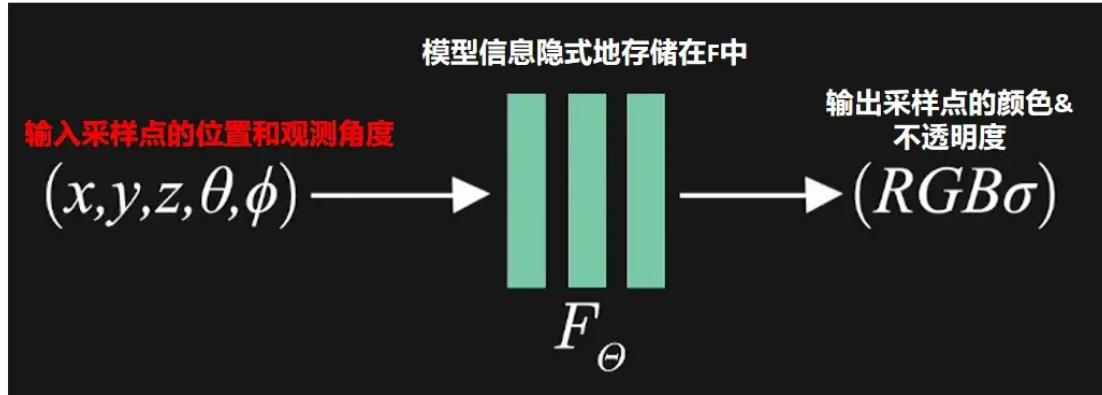
NeRF首次利用隐式表示实现了照片级的视角合成效果，与之前方法不同的是，它选择了Volume作为中间表示，尝试重建一个隐式的Volume。NeRF的主要贡献：

- 提出了一种5D neural radiance field 的方法来实现复杂场景的隐式表示。
- 基于经典的Volume rendering提出了一种可微渲染的流程，包括一个层级的采样策略
- 提出了一种位置编码 (positional encoding) 将5D坐标映射到高维空间。



NeRF基于MLP合成Volume的隐式表示，并通过Volume rendering渲染到2D图片计算损失

在理解NeRF之前，我们首先需要厘清两个概念，**神经场 (Neural field)** 与**体渲染 (Volume rendering)**。



我们可以简单地理解为：

我们使用一个NeRF神经网络采取体积雾的渲染方式，通过已知视角的图片进行训练，然后输入其它相机视角所对应一组采样点的参数，从而预测出未出现视角观测角度下采样点的图片。

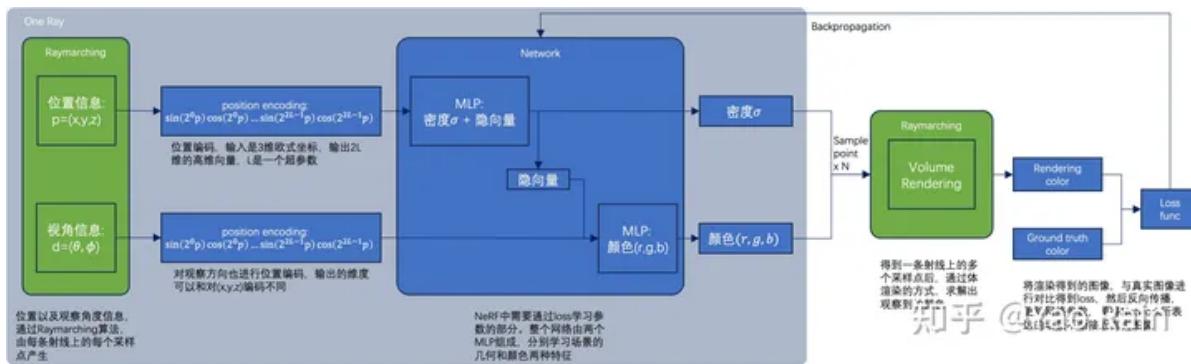
此时三维模型的信息就储存在了NeRF神经网络之中！所以这是一种“隐式”的表示方法，而不是像点云、体素、网格这种显式的表示方式！

总体流程：

NeRF，一种用于三维场景表达的隐函数

那么NeRF到底是什么？NeRF，全称为neural radiance field，翻译为神经辐射场。这里的辐射场，并不是新鲜玩意。任何结构，如果可以用空间位置信息查询到对应的一组颜色+密度信息，那么它就可以是这里所说的辐射场。对于有图形学背景的人来说，3D Texture就是一个很好的例子。只不过，NeRF使用了神经网络来表示一个辐射场。

话不多说，我们先从整体上直观的来认识NeRF。我做了一个图详细的描述了一下NeRF的整个pipeline：



NeRF pipeline

整体看下来NeRF的整体想法不算复杂：在forward用体渲染采样神经网络的辐射场，然后把结果和ground truth做对比，反向优化网络参数，不断迭代使得辐射场接近真实值。这么看来，入门最大的障碍，大概就是懂机器学习的不懂图形学，搞图形学的不懂机器学习吧（

于是为了方便理解，在上图中我把图形学和机器学习的部分，分别用不同颜色来表示，以供大家对症下药。绿色的部分，是属于图形学的组件，基本上只涉及体渲染。蓝色的部分，是涉及到机器学习的组件，也只是最基本的MLP而已。

之所以用神经网络来表示辐射场，一是比起类似3D Texture这种volume的表示，神经网络没有分辨率的限制，也不会随着表达精度的上升而光速增长体积。另一个原因，就是神经网络的参数是可学习的，神经网络中的计算/查询是可微的（后面讲到ngp，我们就会发现，只要满足这两个条件，三维表征也可以是其他形式的）。因此，NeRF才能通过不断更新参数，来使得表征更加接近真实值，完成高质量的新视角合成任务。

(--<https://zhuanlan.zhihu.com/p/631284285>)

1.3 神经场

神经场在Neural Fields in Visual Computing and Beyond[1]这篇文章中得到了非常详尽的阐述，简单来说：

场 (field) 是为所有 (连续) 空间和/或时间坐标定义的量 (标量)，如电磁场，重力场等。因此当我们在讨论场时，我们在讨论一个连续的概念，而且他是将一个高维的向量映射到一个标量。

神经场表示用神经网络进行全部或者部分参数化的场。

在视觉领域，我们可以认为神经场就是以空间坐标或者其他维度 (时间、相机位姿等) 作为输入，通过一个MLP网络模拟目标函数，生成一个目标标量 (颜色、深度等) 的过程。

1.4 体渲染

简单来说，体渲染就是模拟光线穿过一系列粒子，发生一些反射、散射、吸收，最后到达人眼的过程。没错，其实不管是你硬表面软表面，NeRF都是把各种物体都当做不同密度的粒子、烟雾一类的东西去建模的。

Volume Render部分

NeRF采用的是一种体积雾的渲染方式，在获取一定范围采样点的(r, g, b, a)之后需要再进行特定积分运算，最终得到对应像素最终的(r, g, b, a)，在训练时通过光线采样点积分的得到的像素值

代码在这一部分定义采样点实际采样的区域，并将这一组采样点进行积分的方式处理输出为rgb图像

需要的主要参数有：

H, W: 图片的高，宽 (像素为单位)

K: 相机的内参，即焦距

chunk: 同时处理的光线数量 (batch)

rays: 表示每个示例光线的起点和方向

c2w: 相机到世界的变换矩阵

near: 光线最近的距离

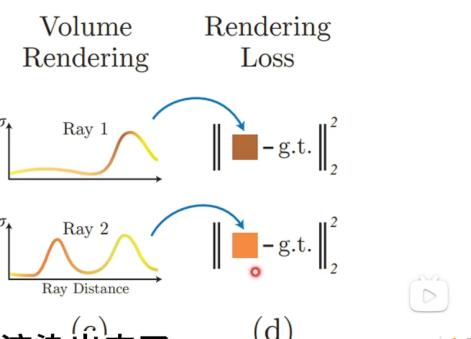
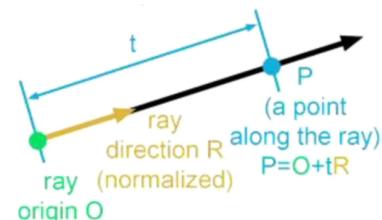
far: 光线最远的距离

通过特定积分的方式求得实际显示的颜色

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

$$\hat{C}_c(\mathbf{r}) = \sum_{i=1}^{N_c} w_i \mathbf{c}_i, \quad w_i = T_i (1 - \exp(-\sigma_i \delta_i))$$

其实它就能够把整个画面给渲染出来了



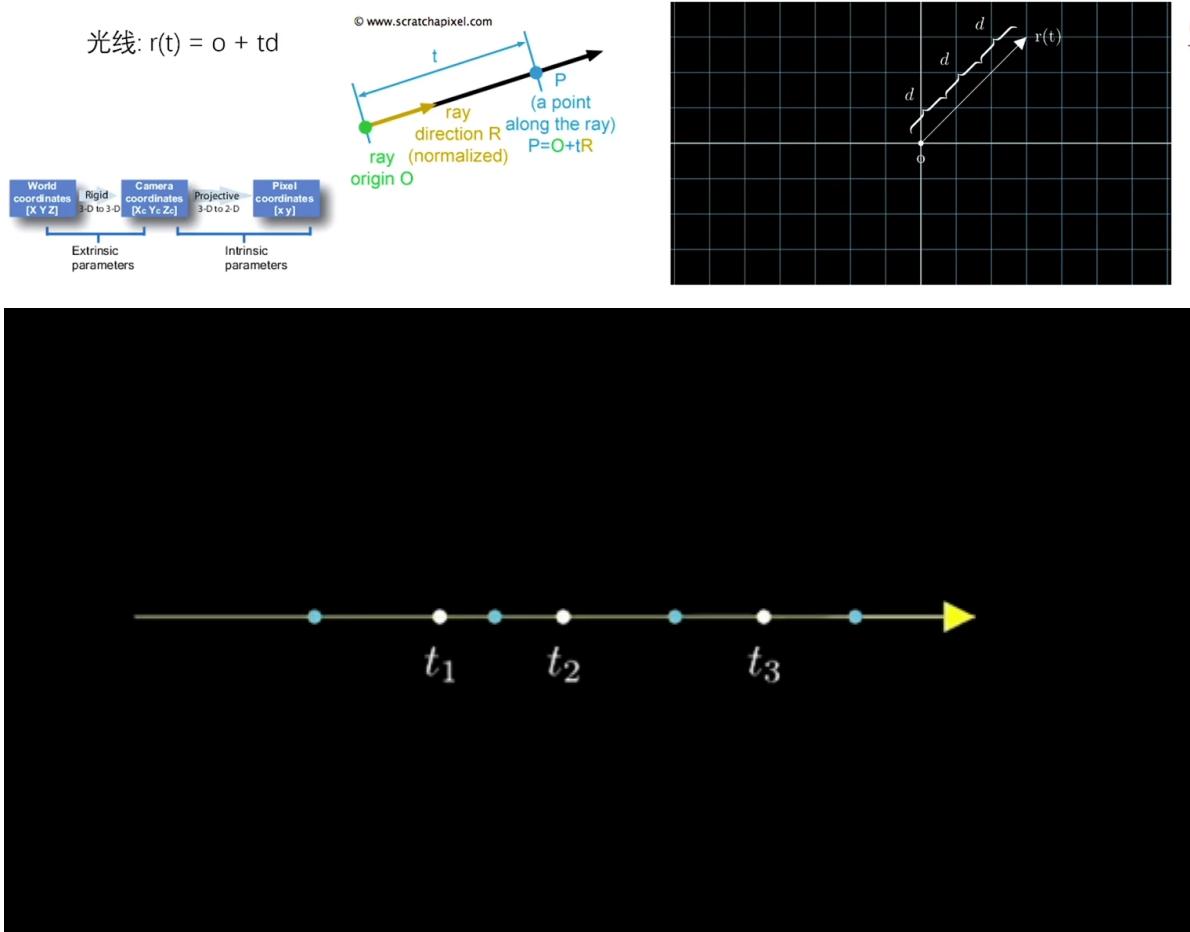
关于体渲染的理解推荐大家阅读State of the art on neural rendering[2]。体渲染简而言之是从体数据渲染得到2D图片的过程。

现实生活中，有一些领域采集的3D数据是以体数据格式存储的，例如医疗中的CT和MRI、地质信息、气象信息等，这些数据需要渲染到2D图像才能够被人类理解。除此之外体数据建模（Volume）相比于传统的Mesh、Point，更加适合模拟光照、烟雾、火焰等非刚体，因此也在图形学中有很多应用。

体数据的渲染主要是指通过追踪光线进入场景并对光线长度进行某种积分来生成图像或视频，具体实现的方法包括：Ray Casting, Ray Marching, Ray Tracing。

基于体渲染的研究在NeRF之前有很多，因为体渲染是一种可微渲染，非常适合与基于统计的深度学习相结合。目前可微渲染领域也有一些研究，是未来计算机视觉和计算图形学结合的一个重要方向。

具体：



不一定是完全均匀的 而是有随机间隔的(t_1 t_2 t_3)

其实就是在光线上取一段路径做积分。对计算机来说，则是在光线上产生一些step， t ，得到一系列采样点的位置 $r(t)$ 和观察方向 d ，然后把这些点作为神经网络的输入，得到对应的密度和颜色。最后再根据距离相机远近计算一个衰减权重 $T(t)$ ，把这些采样点做一个加权和就ok了。这与我们在pipeline图中的描述也是一致的。

1.4++ 体渲染数学公式推导

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t) \sigma(\mathbf{r}(t)) \mathbf{c}(\mathbf{r}(t), \mathbf{d}) dt, \text{ where } T(t) = \exp\left(-\int_{t_n}^t \sigma(\mathbf{r}(s)) ds\right)$$

【论文精读】NeRF中的数学公式推导

Yuhu Hu 已于 2022-04-24 14:09:55 修改

 NeRF 专栏收录该内容

27 订阅 8 篇文章 

这篇文章用于记录NeRF论文中数学公式的推导过程。

论文里的第一个公式就很硬核，展示了相机射线的期望颜色的计算方法。

5D 神经辐射场将场景表示为空间中任意点的体积密度和定向发射的辐射。文章使用经典体积渲染的原理，来渲染任何穿过场景的光线的颜色。体积密度 $\sigma(x)$ 可以解释为射线终止在位置 x 处无穷小粒子的微分概率。而期望的颜色 $C(r)$ （相机光线 $r(t) = o + td$ ，近处远界限为 t_n 和 t_f ）可以被表示为

$$C(r) = \int_{t_n}^{t_f} T(t) \sigma(r(t)) \mathbf{c}(r(t), d) dt, \text{ where } T(t) = \exp\left(-\int_{t_n}^t \sigma(r(s)) ds\right)$$

这个结论来源于1995年Max的一篇文章Optical models for direct volume rendering，是体渲染的开山之作。这篇文章将光线模型分为三类，我们逐个看一看。

光线吸收模型

简单来说就是吸收它们拦截的所有光，却不敢射或发射任何光。假设粒子是相同的球体，半径为 r ，投影面积 $A = \pi r^2$ ，设 ρ 为每单位体积的粒子数。这个模型的传递方程是

$$\frac{dI}{ds} = -\rho(s)AI(s) = -\tau(s)I(s)$$

其中 s 是沿光流方向的光线的长度， $I(s)$ 是距离 s 处的光强度， ρ 是光点密度。 $\tau(s) = \rho(s)A$ 称为消光系数，反映了光被遮挡的概率，整理并且等式两边同时积分：

$$\frac{1}{I(s)} \frac{dI}{ds} = -\tau(s)$$

$$\int_0^s \frac{1}{I(s)} ds = \int_0^s -\tau(t) dt$$

其中， I_0 是在 $s = 0$ 处的强度，而 $T(s) = \exp(-\int_0^s \tau(t) dt)$ 是介于0和 s 之间的介质的透明度。在体渲染中，消光系数 τ 通常称为不透明度。

这里的 $\tau(t)$ 相当于nerf的 σ

光线发射模型

介质还可以通过外部照明的发射或反射来增加光线。

如果粒子是透明的，但以每单位投影面积的强度 C 发光，这个 $I(s)$ 的微分方程为：

$$\frac{dI}{ds} = C(s)\rho(s)A = C(s)\tau(s) = g(s)$$

这个 $g(s)$ 被叫做源项。

我们把式子可以变为：

$$dI = g(s)ds$$

对两边进行积分：

$$\int_0^s dI = \int_0^s g(s)ds$$

这个方程的解是：

$$I(s) = I_0 + \int_0^s g(t)dt$$

其中 I_0 是 $s = 0$ 处的光强度。

表示 s 位置光发出的强度

吸收发射模型

实际上，空间中的粒子会遮挡入射光，并添加自己的光。因此，一个现实的微分方程应该包括源项 $g(s)$ 和衰减项 $\tau(s)$ 。我们只需要将前两种模型进行简单的数值加和（微分方程右侧加在一起），就可以得到这个模型的传递方程：

程右侧加在一起，就可以得到这个模型的传递方程：

$$\begin{aligned} \frac{dI}{ds} &= g(s) - \tau(s)I(s) \\ \text{我们把 } \tau(s)I(s) \text{ 移到等式左边，然后都乘上 } \exp\left(\int_0^s \tau(t)dt\right), \text{ 得到：} \\ \left(\frac{dI}{ds} + \tau(s)I(s)\right)\exp\left(\int_0^s \tau(t)dt\right) &= g(s)\exp\left(\int_0^s \tau(t)dt\right) \\ \text{也可以表示为：} \\ \frac{d}{ds}\left(I(s)\exp\left(\int_0^s \tau(t)dt\right)\right) &= g(s)\exp\left(\int_0^s \tau(t)dt\right) \\ \text{从 volume 边缘的 } s = 0 \text{ 积分到眼睛的 } s = D, \text{ 我们得到：} \\ I(D)\exp\left(\int_0^D \tau(t)dt\right) - I_0 &= \int_0^D \left(g(s)\exp\left(\int_0^s \tau(t)dt\right)\right)ds \\ \text{把 } I_0 \text{ 移到等式右边，然后等式两边都乘上 } \exp\left(-\int_0^D \tau(t)dt\right), \text{ 我们可以得到 } I_D: \\ I(D) &= I_0 \exp\left(-\int_0^D \tau(t)dt\right) + \int_0^D \left(g(s)\exp\left(-\int_s^D \tau(t)dt\right)\right)ds \end{aligned}$$

第一项表示来自背景的光，乘以空间的透明度。第二项是源项 $g(s)$ 在每个位置 s 贡献的积分，乘以位置 s 到眼睛的透明度 $T'(s) = \exp(-\int_s^D \tau(x)dx)$ ，那么：

$$I(D) = I_0 T(D) + \int_0^D g(s) T'(s) ds$$

等式右侧第一项代表着从坐标0点出发经过0到D的介质入射到摄像机的光强，称之为背景光，在NeRF中，这一项考虑为0。所以在NeRF中，这个式子化简为：

$$I(D) = \int_0^D g(s) T'(s) ds = \int_0^D T'(t) \tau(t) c(t) dt$$

其中 $T'(t) = \exp(-\int_t^D \tau(x)dx)$ 。

转变为NeRF当中的形式

上面的式子和NeRF原文中仍然有差别，这是因为NeRF和Max的文章中使用的坐标不同。Max文章中的坐标是让相机在D坐标，而无穷远点在0坐标，这样前面的推导就是正确的。但是NeRF中的坐标，是让相机在坐标原点，无穷远坐标就是无穷远，这样就可以得到：

$$I(0) = \int_0^\infty g(s) T'(0, s) ds = \int_0^\infty T'(0, t) \tau(t) c(t) dt$$

其中 $T'(0, t) = \exp(-\int_0^t \tau(x)dx)$ 。

试想，其实在0到 ∞ 不是所有位置上都有介质，介质总有边界，我们就可以定义近平面和远平面 t_n 和 t_f ，那么上述的积分其实可以写成：

$$I(0) = \int_{t_n}^{t_f} T'(t_n, t) \tau(t) c(t) dt$$

NeRF中把消光系数（也就是不透明度） $\tau(t)$ 叫做体积密度 $\sigma(t)$ ，那么上式可以整理为：

$$I(0) = \int_{t_n}^{t_f} T'(t_n, t) \sigma(t) c(t) dt$$

如果记 $T(t) = \exp(-\int_{t_n}^t \sigma(s) ds)$ ，那么上式可以变成：

$$I(0) = \int_{t_n}^{t_f} T(t) \sigma(t) c(t) dt$$

而这一切讨论都是在固定射线的情况下，如果这个射线是动态的，我们还需要用 $r(t) = o + td$ 来表示的话，那么 $\tau(r(t))$ 其实可以表示在 r 这条射线上， t 位置的体积密度， $c(r(t), d)$ 就可以表示在 r 这条射线上， t 位置对 d 方向的光强。那么上式可以进一步变为：

$$I(0) = \int_{t_n}^{t_f} T(t) \sigma(r(t)) c(r(t), d) dt, \text{ where } T(t) = \exp(-\int_{t_n}^t \sigma(r(s)) ds)$$

因为我们研究的是 r 这条射线上的光，眼睛位置固定不变，所以可以隐去相机位置 o ，而添加参数 r 来强调射线也是个变量，再用 C 替换掉光强 I ，所以最终式子变为：

$$C(r) = \int_{t_n}^{t_f} T(t) \sigma(r(t)) c(r(t), d) dt, \text{ where } T(t) = \exp(-\int_{t_n}^t \sigma(r(s)) ds)$$

这就是原文当中的公式1。

把上式离散化，将近远平面区间等分为 N 份，在每个小区间内取样。

把积分符号变为求和， $T(t)$ 变为 T_i ， $c(r(t), d)$ 变为 c_i ， $\sigma(r(t))dt$ 变为 $\sigma_i \delta_i$ ，则得到最终的离散化公式。

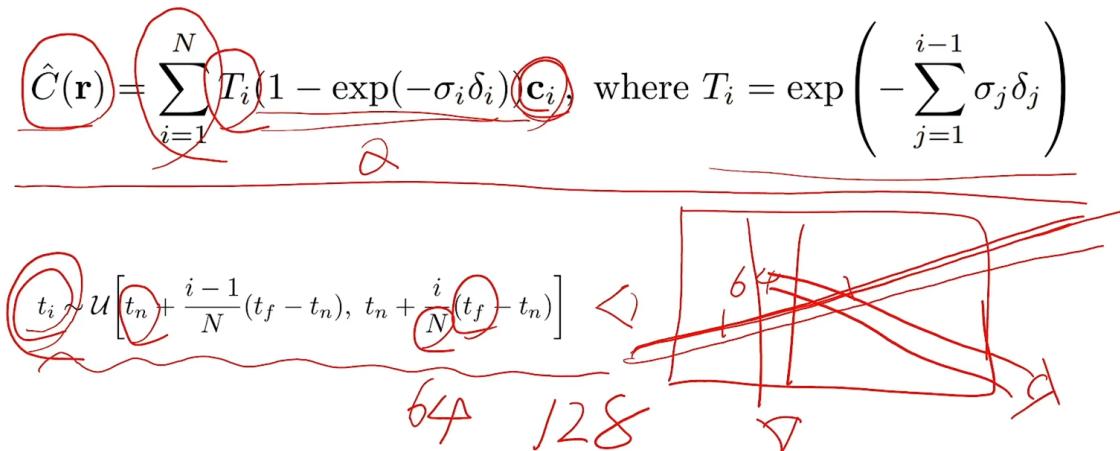
论文里给出的黎曼和的形式如下：

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

体渲染公式的黎曼和形式

总的来说，体渲染的过程并不复杂，重要的是其积分形式带来的可微性质，使得基于梯度的机器学习算法，能够迭代更新可学习参数，从而解决场景优化问题。具体而言，公式中 T_i 的黎曼和形式是可微的， C_i 是密度 $\sigma(r(t))$ 和颜色 $c(r(t), d)$ 相乘得到的，二者都是通过神经网络获取的，过程也是可微的。

c 本身的颜色值



相机位姿为基础的一组采样点位置—5D Input

Position(相机位置为起点的采样点位置): (x, y, z)
需要结合光线公式，采样点位置=出发点+距离*方向

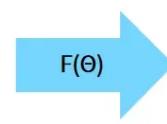
Direction(观测角度): (θ, ϕ) 这个是球坐标的表现方法，只需两个参数，方位角和俯仰角，代表的是相机位置与成像平面所对应像素连线发出的“光线”射线的方位，类似于Yaw, Pitch, Roll中的Yaw和Pitch，但不包含Roll注：

[1] θ 是点在xOy平面上的投影与原点的连线和x轴正方向所成夹角，也就是一般说的极坐标的 θ ，取值范围为 $[0, 2\pi]$ 或 $[0, 2\pi]$ 。

ϕ 是点与原点所成连线和z轴正半轴所成夹角，取值范围为 $[-\pi, \pi]$ (必须全闭，否则顶点取不到)。

[2]实际上在网络输入的仍然是三维向量，来表示射线方向 (x, y, z)

一个像素点对应的射线和一组采样点位置，即为一个训练资料~



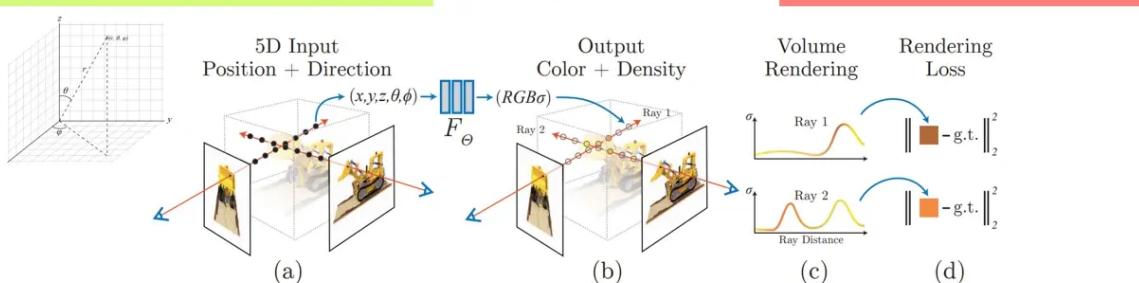
NERF三维重建 隐式的神经表示

一根射线上一组采样点的属性—4D Output

像素点对应射线上的一组采样点的颜色值 (r, g, b) 和不透明度 (σ)

射线方向上对采样渲染点进行积分，在第一次出现波峰对该像素点的着色影响最大

注：
Density(密度值/不透明度): σ
该值与观测角度无关



1.5 位置编码

(提升神经辐射场的方法之一)

位置编码 $\gamma(x, y, z)$

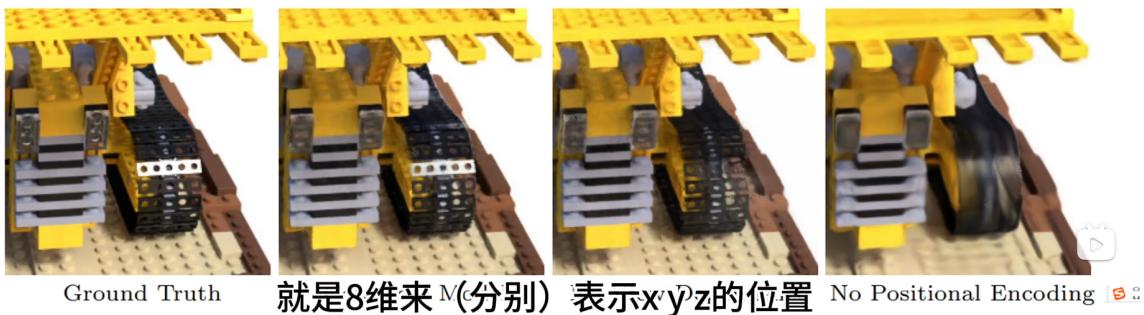
$$F_\Theta = F'_\Theta \circ \gamma$$

$$\gamma(p) = (\sin(2^0 \pi p), \cos(2^0 \pi p), \dots, \sin(2^{L-1} \pi p), \cos(2^{L-1} \pi p)).$$

sin和cos两个为一项

空间坐标中用10项，即20维来分别表示(x,y,z)位置编码

相机方向用4项，即8维来分别表示(x,y,z)位置编码



因为神经网络倾向于学习低频信息，所以我们给输入映射到高频空间，这样能更容易学习高频信息

上图可以看出没加入位置编码，细节就会比较缺失

chatgpt:

为什么这个位置编码能让他高频信息更容易获得呢

位置编码中的正弦 (sin) 和余弦 (cos) 函数之所以能够帮助模型更容易获取高频信息，是因为它们引入了不同频率的周期性振荡模式，这有助于模型理解和捕捉场景中的细微变化和细节。以下是解释这一点的原因：

- 周期性模式：**正弦和余弦函数都是周期性函数。通过在位置编码中使用这些函数，不同位置的编码将在不同频率的周期性模式下振荡。这意味着编码的不同维度将对场景中的不同频率的特征具有更敏感的响应。**高频信息**通常以**快速振荡**的形式存在，因此**正弦和余弦**函数有助于模型更好地捕捉这些高频特征。
- 多尺度表示：**正弦和余弦函数具有不同的频率。通过在位置编码中使用不同频率的正弦和余弦函数，模型可以同时考虑多个尺度的特征。这意味着模型可以同时捕捉高频和低频信息，从而更全面地理解场景的细节。
- 模型可学习性：**正弦和余弦函数的频率是可学习的参数。在训练中，神经网络可以自动调整这些频率以适应不同的场景和任务。这意味着模型可以根据需要调整编码的频率，以更好地捕捉高频信息。

综合来说，位置编码中的正弦和余弦函数提供了一种有效的机制，使神经网络更容易获取高频信息，从而提高了模型对细节的感知和渲染的质量。这对于处理复杂的三维场景重建和渲染任务非常有帮助。

关于使用位置编码的理由，也是基于过去大家对MLP的观察。人们发现MLP的参数在学习的时候，会倾向于学习低频信息，所以直接对网络使用5D输入，效果会很糊。使用位置编码后，原本在低维接近的点，在高维会离得很远。换句话说，MLP在高维学到的低频信息，往往对低维来说不是低频的。

1.6 分层采样

Hierarchical sampling

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

$$\hat{C}_c(\mathbf{r}) = \sum_{i=1}^{N_c} w_i \mathbf{c}_i, \quad w_i = T_i (1 - \exp(-\sigma_i \delta_i))$$

Coarse 64 Output

Fine $64 + 128$

(提升神经辐射场的方法之一)

由于如果是俊宇64个采样点这样的话，很可能有很多采样点在空气中，浪费了。因此采用2层网络。

粗网络 均匀采样64个点

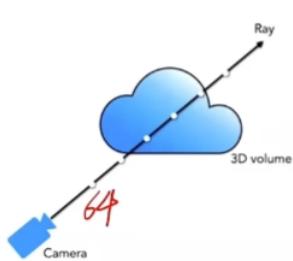
精细网络 使用粗网络的输出结果，当做权重采样

Hierarchical sampling

$$\hat{C}_c(\mathbf{r}) = \sum_{i=1}^{N_c} w_i \mathbf{c}_i, \quad w_i = T_i (1 - \exp(-\sigma_i \delta_i)) \quad \hat{w}_i = \frac{w_i}{\sum_{j=1}^{N_c} w_j}$$

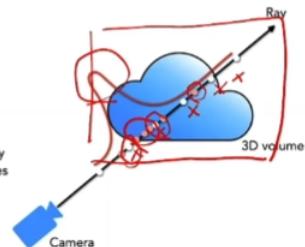
权重可以看成沿着射线的分段常数概率密度函数 (Piecewise-constant PDF)

$$128 + 64 = 192$$



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

treat weights as probability distribution for new samples

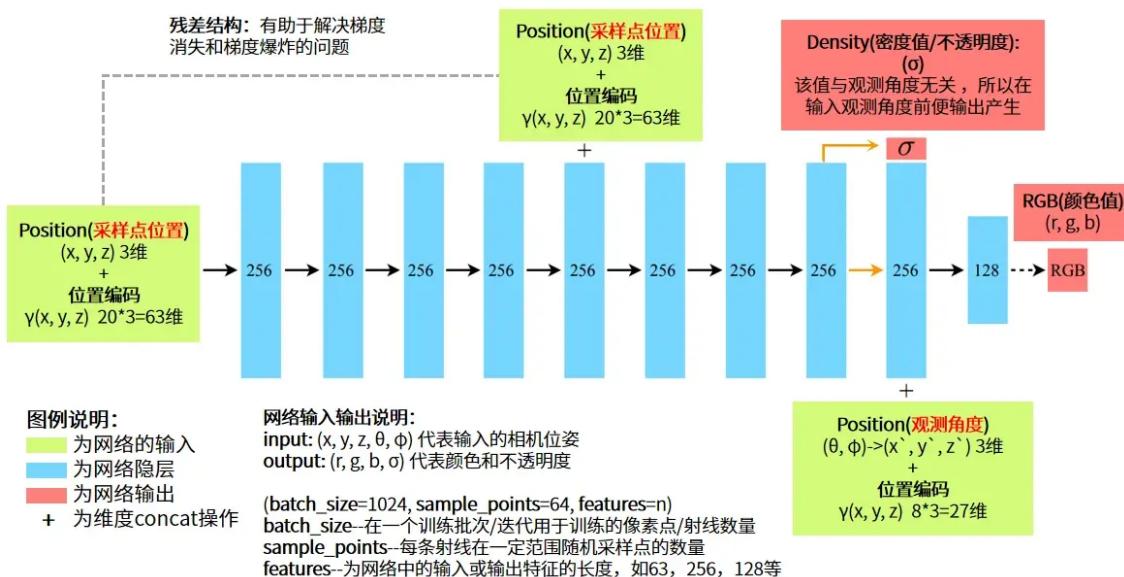


先是用cross网络使用较少的采样点，粗采样一遍。之后按照粗采样到的位置的密度值，在相应区间生成不同数量的采样点。原则上重点照顾拥有更高密度的位置，密度越高，使用越多的采样点进行精细采样。

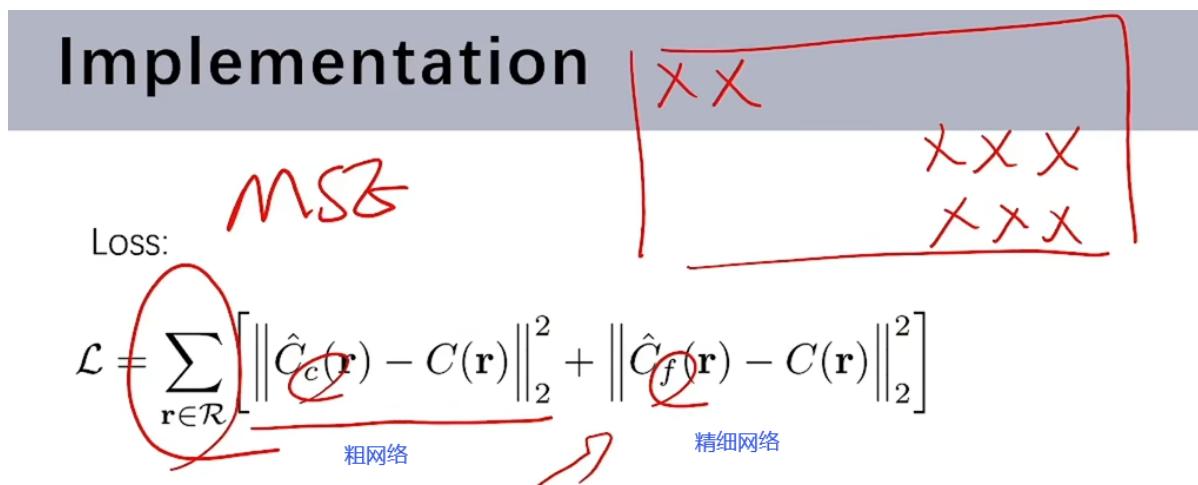
- 使用两层网络，第一次的计算为粗网络模型，第二次的计算为精细网络模型
- 粗网络模型的采样点位为64个，精细网络模型的采样点位数为 $\underline{64+128}$
- 一条光线的总点位数量为 $\underline{64+64+128=256}$

(很像包围盒思想)

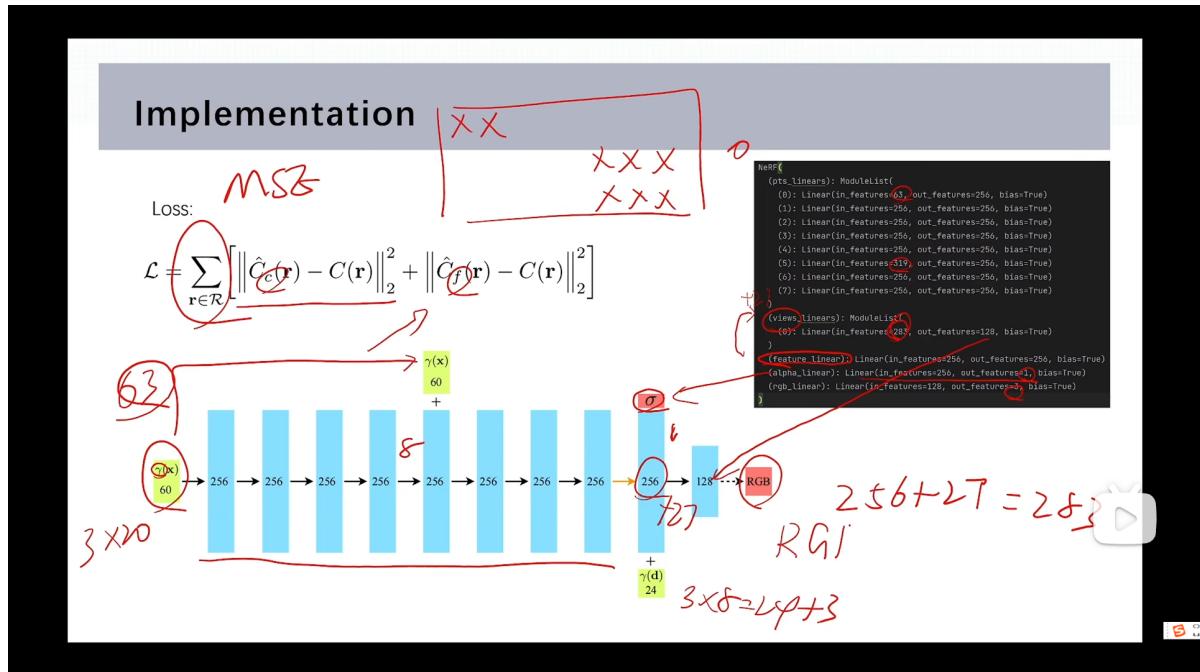
1.7 具体实现细节：



损失函数



均方误差MSE 计算渲染出的点位与真实gt (ground truth) 的差值



1.8 评价指标

评价指标-PSNR

PSNR: Peak Signal to Noise Ratio 峰值信噪比

$$\text{PSNR} = 10 \times \lg \left(\frac{\text{MaxValue}^2}{\text{MSE}} \right)$$

$255^2 \uparrow$

值越大越好

MaxValue 为像素值的最大取值，为255

1.9 劣势

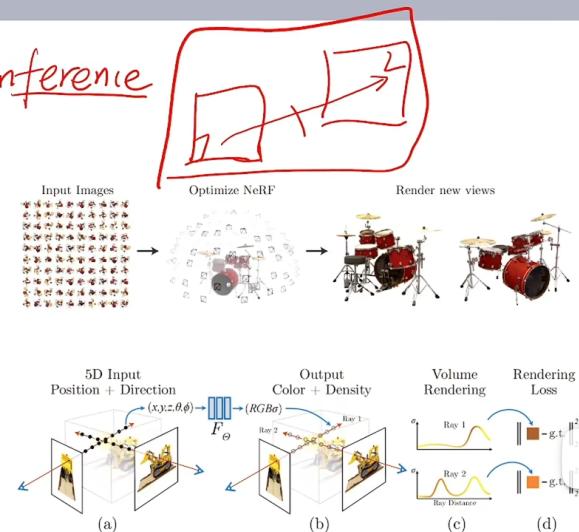
NeRF 劣势

- 4
5
1. 它很慢，训练和推理都很慢
2. 它只能表示静态的场景
3. 对光照处理的不好
4. 训练的模型都仅能代表一个场景，没有泛化能力



train

inference



1.10 NeRF总结

从我个人的观点来看，我举得NeRF最大的贡献，是实现了神经场（Neural Field）与图形学组件Volume rendering的有效结合。NeRF本身的方法实现是非常简洁的，简洁而有效，说明这种组合是合理的。这也启发我们探索更多视觉和图形学的交叉领域，事实上这一方面的探索还比较少。

另一方面，NeRF的简洁也说明了他本身存在很多问题。接下来我们将结合NeRF的问题以及NeRF的应用，概述一下NeRF的发展。

参考

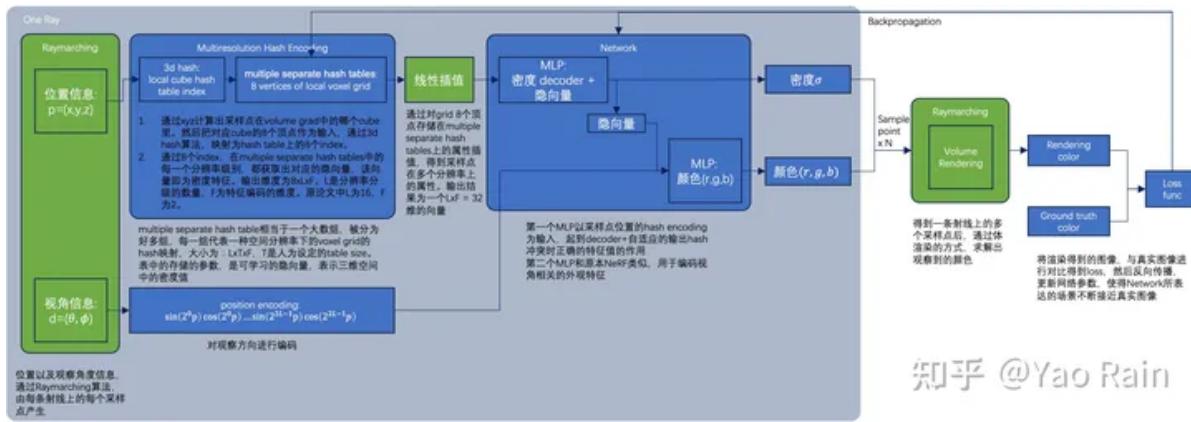
- <https://github.com/yenchenlin/awesome-NeRF>
- <https://arxiv.org/abs/2101.05204> (survey)
- <https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-generating-camera-rays/definition-ray>
- <https://blog.csdn.net/zhuoqingjoking97298/article/details/122161124> 相机
- <https://keras.io/examples/vision/nerf/>
- <https://blog.csdn.net/YuhshiHu/article/details/124318473> (NeRF的数学公式推导)

Instant-NGP

<https://zhuanlan.zhihu.com/p/631284285>

用参数化显式表达加速pipeline

先直观的感受一下Instant-NGP的pipeline：



知乎 @Yao Rain

Instant-NGP pipeline

流程一眼看上去和NeRF差不多，但是密密麻麻的小字直觉上就让人觉得不详（

这里为先直接给出Instant-NGP与NeRF的异同：

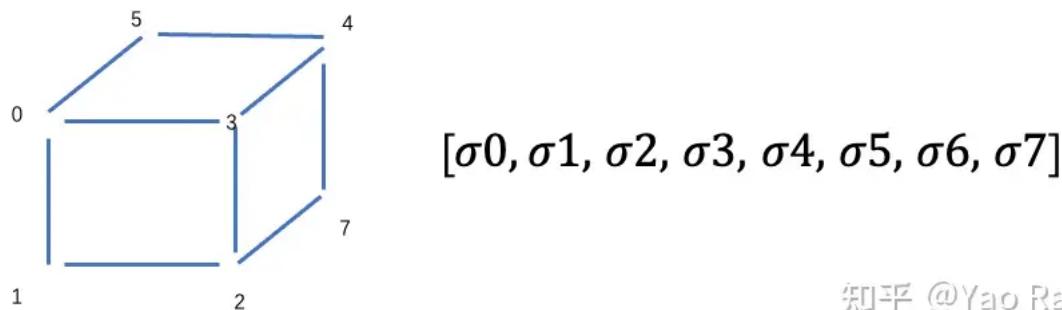
1. 同样基于体渲染
2. 不同于NeRF的MLP， NGP使用稀疏的参数化的voxel grid作为场景表达；
3. 基于梯度，同时优化场景和MLP（其中一个MLP用作decoder）。

可以看出，大的框架还是一样的，最重要的不同，是NGP选取了参数化的voxel grid作为场景表达。通过学习，让voxel中保存的参数成为场景密度的形状。

MLP最大的问题就是慢。为了能高质量重建场景，往往需要一个比较大的网络，每个采样点过一遍网络就会耗费大量时间。而在grid内插值就快的多。但是grid要表达高精度的场景，就需要高密度的voxel，会造成极高的内存占用。考虑到场景中有很多地方是空白的，所以NVIDIA就提出了一种稀疏的结构来表达场景。

Hashing of voxel grid

先简单介绍一下voxel grid和它的hash table形式。voxel，可以认为是空间中的一个小立方体。voxel grid，就可以想象成空间中一系列共用顶点的小正方体，这些小正方体的每个顶点，就是一个voxel vertex。这个voxel可以保存任何信息，一般我们拿来保存一些只和位置有关的信息，比如，**密度**和**漫反射颜色**。我们可以用一个三维数组来表示一个voxel grid: $[N, N, N]$ ，同样也可以用一个 N^3 的一维表来表示：



知乎 @Yao Rain

grid和它的table

如上图左边就是一个 $N=2$ 的voxel grid。如果我们按顺序给每一个顶点一个index，然后把其中的密度值放入一个一维表对应的index中，就能得到一个voxel grid的table形式。如果顶点所对应的table中的index是通过hash算法得到的，那这个一维表就是一个hash grid。具体用到的 spatial hash function，原论文和知乎上有很多讲解，这里不做说明。

一般情况下，我们需要查询的点，并不会和grid的顶点重合。所以查询的时候，需要先找到采样点在哪个小立方体内，然后再对立方体的八个顶点做插值来求解。这一过程在NGP pipeline图中也有说明。

Multiresolution hash encoding [多分辨率哈希编码]

如果我们要表达一个精细的场景，使用hash grid的时候，哈希表的大小往往是小于顶点数量的。这样就可能造成哈希冲突，表中的密度不知道应该表达哪个顶点的才好。为了避免这种不好的事情发生，NGP提出了multiresolution hash encoding的方法，翻译为多分辨率哈希编码。

具体而言，我们把相同空间，用不同大小的grid表达，比如从 $16 \times 16 \times 16$ 到 $512 \times 512 \times 512$ 的M个分辨率的grid。然后我们把hash table的大小T设为固定值，比如 $64 \times 64 \times 64$ 。这样一来，我们就得到了M个大小为T的hash table，这些表放在一起就是原文中的multiple separate hash tables。当grid小于64的时候，总是不会发生冲突的。超过64的grid，冲突就冲突了，反正最后的密度，是以不同权重混合每一个分辨率的grid的密度得到的 ($m = w_1 * md_1 + w_2 * md_2 + w_3 * md_3$ md:密度)。至于权重谁高谁低，则是MLP通过loss学到的。

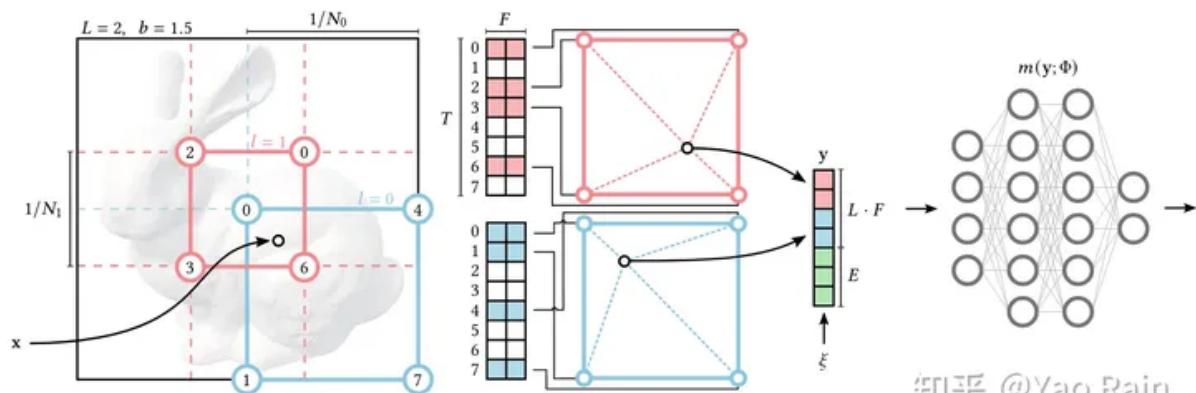
从输入输出的角度而言，Multiresolution hash encoding最后会把一个 (x, y, z) 的位置信息，转变为该位置上的multiple separate hash tables中的密度信息。输出一个 $L \times F$ 维的向量，作为网络的输入。（这是一个针对输入的再赋权）

L是总共有多少种不同的分辨率，（L对应的是特定分辨率体素对应的编码层，按照之前表格给出的超参数，层数被设置为16，即体素分辨率变化分为16级，）

F是每个顶点保存的密度特征编码的维度。

之后MLP为该采样点每个分辨率上的密度分配权重，并做特征decode，得到最终的密度值。

这里放个原文中的图表给各位做参考：



Instant-NGP原文中的算法流程图

-哈希：根据内容就能直接定位数据位置

Instant-ngp不仅拥有可训练的权重值 Φ ，同时还拥有可训练的编码权重 θ 。

Instant-NGP中的MLP

NGP中同样有两个MLP，一个是类似NeRF的外观网络，另一个则大为不同。NGP中和密度相关的MLP，功能是把multiresolution hash encoding输出的，采样点所对应的多个分辨率上的密度特征编码，按照不同的权重混合，同时做decode得到真正的密度值。相比起NeRF中MLP 60维的输入，NGP原文的MLP只有32维输入，同时网络的层数也少很多，是一个小型的MLP，跑起来要比NeRF的网络快很多。

Instant-NGP小结

可以看出，NGP改进了图形学中已有的结构并应用到体渲染框架中，来加速从二维到三维的重建。并且即使和NeRF一样使用了MLP，功能目的却完全不同。

实践：

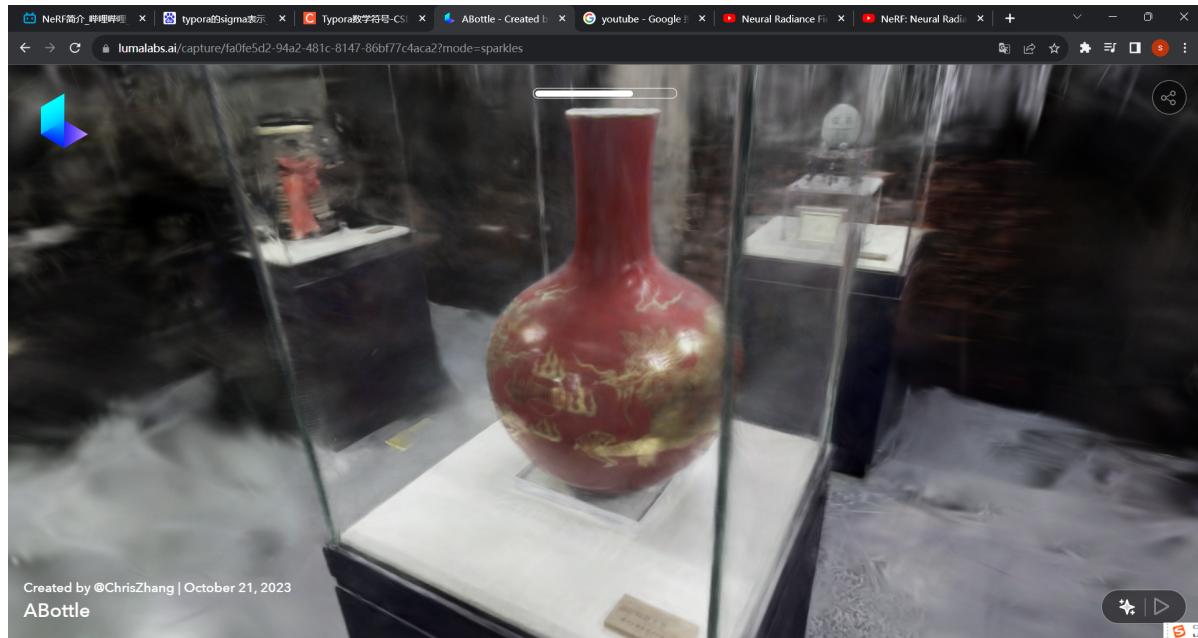
1、

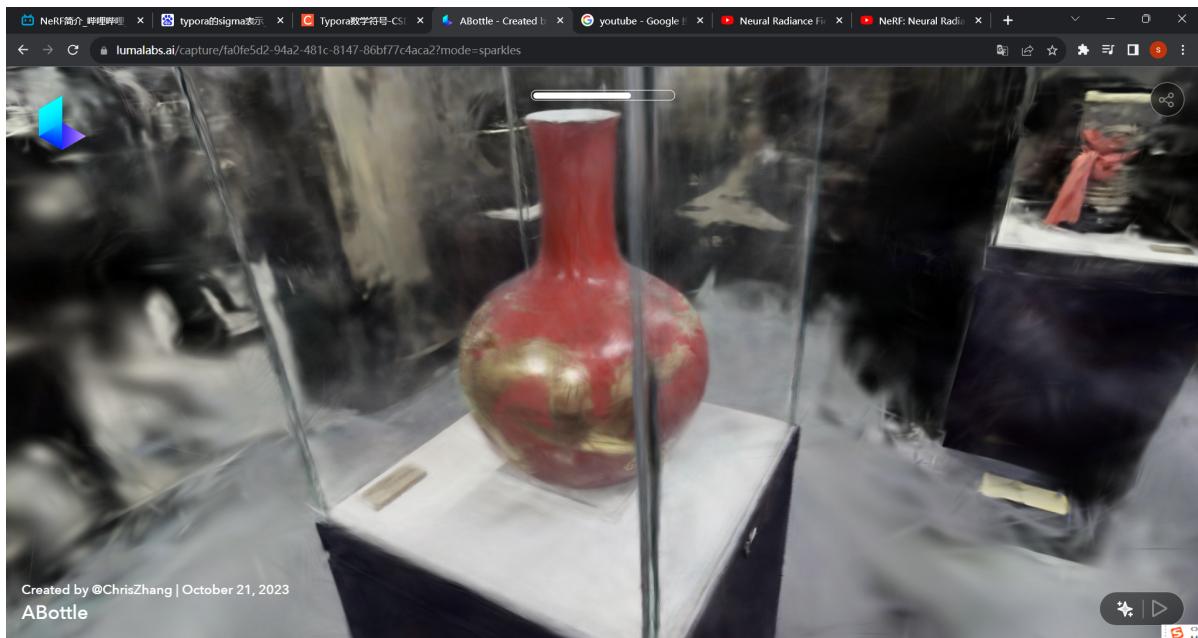
https://www.bilibili.com/video/BV1q84y1U7Qf/?spm_id_from=333.999.0.0&vd_source=f2def4aba42c7ed69fc648e1a2029c7b

2、

[AI \(Nerf\) 扫描3d场景并导入虚幻5！Luma AI教程](#)哔哩哔哩bilibili

使用Luma AI(<https://lumalabs.ai/>)将扫描现实场景并导入虚幻，使用了Nerf技术。导入虚幻插件[http://docs.lumalabs.ai/9DdnisfQaLN1sn](https://docs.lumalabs.ai/9DdnisfQaLN1sn)。导入到UE5后不仅可以随意改变摄像机，还可以随意打光或者把场景里放入其他物体。





原文

<https://github.com/NVlabs/instant-ngp>

windows

<https://github.com/bycloudai/instant-ngp-Windows>

<https://www.youtube.com/watch?v=kq9xlvz73Rg>

LINUX

<https://www.jianshu.com/p/02c3d3cce99b>