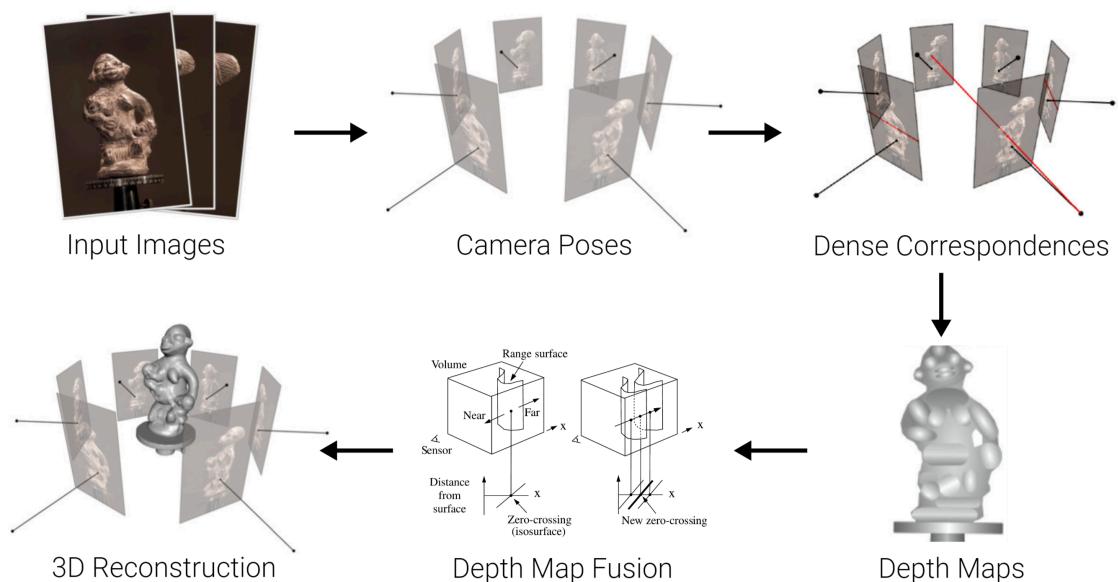


# Lecture 9 - Coordinate-Based Networks

这一章是 Geiger 教授非常期待的一章，关于“基于坐标的神经网络”，因为教授的小组已经在该领域做了非常多的工作，首先提出了 Occupancy 网络的概念，已经建立了一个完整的子领域，并在大约一两年内出现超过了 1000 个后续的工作（真的很惊人...我居然在听这样一位传说级别的教授的课...）

## 9.1 Implicit Neural Representations

- 首先，再回到传统的 3D 重建的流程图中



- 与之前提到的过程相同，从输入图像、评估相机姿态、稠密关联计算、生成深度图、进行深度图融合、最后重建出 3D 模型这个过程
- 到此为止，我们只在其中一些环节提到过使用 Learning 的方式来解决一些问题，但是我们是否可以直接从数据中学习并预测其对应的 3D 模型呢？

- 数据集问题

- 首先，谈到 Learning 问题，数据是一切的基础，我们需要有大量的数据，继续讨论方法才有意义——幸运的是，在过去十年以内，很多的 3D 数据集已经被各种团队制作并开源出来



[Newcombe et al., 2011]



[Choi et al., 2011]



[Dai et al., 2017]



[Wu et al., 2015]



[Chang et al., 2015]

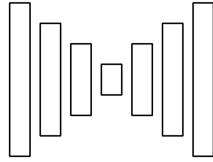


[Chang et al., 2017]

- ○ 包括使用主动可见光扫描（之前提到的结构光技术）、红外线扫描、激光扫描等等，也有使用 CAD 等人工渲染的 3D 模型数据集等；从内容上来说，既有物件本体的模型，也有场景的整体渲染
- 输出表示问题
  - 其次，我们考虑一下，如果我们想构建一个模型，从单个或多个输入图像中直接预测形状结构的模型，我们应该如何利用这些数据？



Input Images

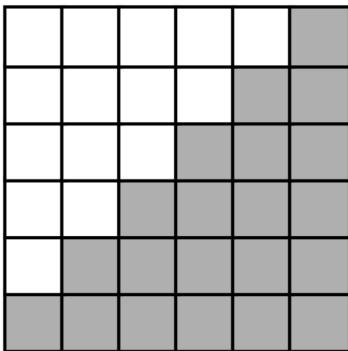


Neural Network

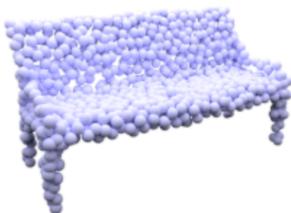
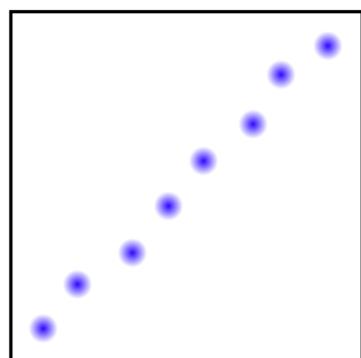


3D Reconstruction

- 输入数据是 2D 像素矩阵，Encoder 部分已经很清晰了——那么 Decoder 应该如何设计？或者说，我们应该如何设计输出的表示方式？
- 已经有很多团队尝试了很多种表示方式：
- **Voxel 表示** ([Maturana et al., IROS 2015](#))



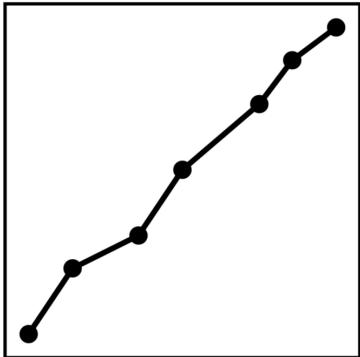
- 体素是 3D 空间的一种离散化表示，将连续的 3D 空间划分为规则的网格
- 由于其网络结构规则，非常容易就可以用神经网络建模并处理
- 空间复杂度为  $O(n^3)$ ，因此对于分辨率的限制很高——我们无法处理高分辨率的结果
- 曼哈顿假设，体素方法假设场景是由对齐的直线、平面构成，在真实事件中存在偏差
- Points 表示 ([Fan et al., CVPR 2017](#))



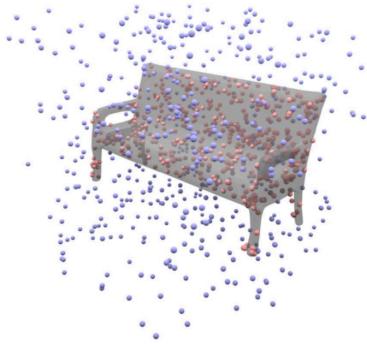
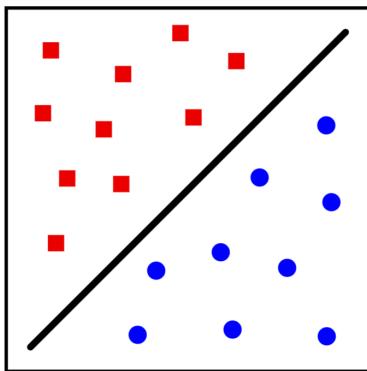
- 点云通过将 3D 物体的表面离散化为一组 3D 点来表示，每个点在空间中

## 具有坐标

- 并不建模点与点之间的连接性、拓扑结构
- 由于计算和存储的限制，点的数量通常是有限的，由此限制了模型的精度
- 点云适合描述粗糙的全局模型，但是在局部细节上无法很好的体现
- **Meshes 表示** ([Groueix et al., CVPR 2018](#))
- 



- 网格表示法，将 3D 中的物体离散化为顶点和面的组合，定点描述物体的空间坐标，面连接顶点以形成表面
- 由于计算和存储的限制，顶点的数量通常有限，同样因此限制了模型的精度
- 需要类别特定的模板来建模，例如椅子、车等现成的 3D 模板
- 可能会导致自相交问题（由于几何结构的复杂性和优化限制，网格之间可能会相交，而这一点违反了现实中几何合理性）
- Geiger 教授的小组在这些的基础上，想要实现非离散化的一种隐式表示方式，非离散化+隐式的好处是，它可以表示任意的拓扑结构，并且理论上来说内存开销很低，并且它并不需要任何现成的特定模板（如 mesh 表示的情况）
-



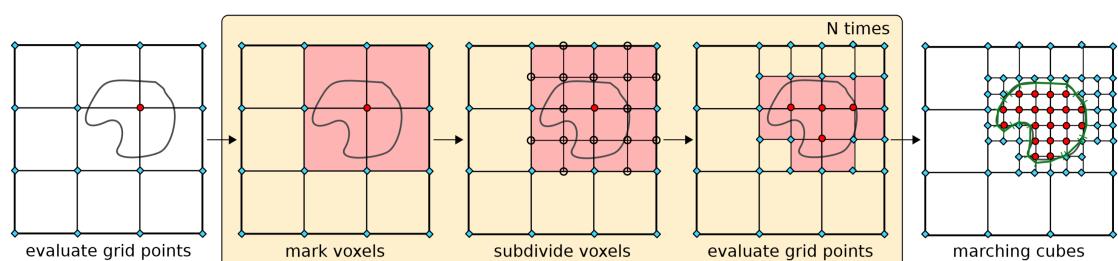
- 这种方法的关键思想包括：
- 不使用显式方式表达 3D 形状
- 考虑通过一个非线性分类器的决策边界来隐式地表示物体的表面  $f_{\theta}: \mathbb{R}^3 \times \mathcal{X} \rightarrow [0, 1]$

$\mathbb{R}^3$ : 3D 坐标 –  $\mathcal{X}$ : 条件参数 (例如图像) –

$$\begin{aligned} \mathcal{L}(\theta, \psi) = & \sum_{j=1}^K \textcolor{red}{\{K\}} \\ & \textcolor{blue}{\{\text{BCE}(f(\theta(p_{ij}), z_i), o_{ij})\}} + K \left[ \right. \\ & \textcolor{green}{\{q(\psi(z | p_{ij}, o_{ij}))_{j=1:K}\}} \parallel p_0(z) \\ & \left. \right] \end{aligned}$$

$K$ : 随机采样的 3D 点 ( $K = 2048$ ) – BCE: 交叉熵损失函数 –  $q_\psi$ :

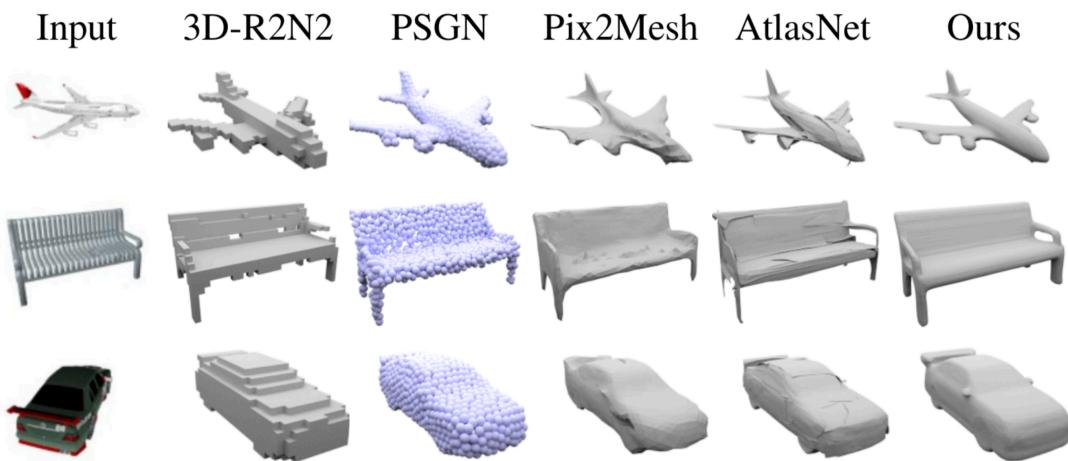
- 现在，得到一个非常好的分类器后，我们考虑如何基于这个分类器来提取网格



- ■ Multiresolution IsoSurface Extraction (MISE)

- 我们通过一种方法，称为**多分辨率等值面提取**，具体来说，我们先用一个比较粗糙的网格来采样，通过不断查询分类器来得知哪些网格属于物体内部、哪些网格属于物体外部
- 然后我们针对于属于物体内的这部分网格，继续细化采样的网格，然后再次不断查询分类器——我们将这个过程迭代 N 次，直到得到我们想要的细化粒度
- 这个过程大概需要 1-3 秒，取决于物体的大小，主要的开销在于不断查询分类器，但是我们并非全局查询，而是不断细化局部查询，因此并不会非常慢，但也不会非常快

- 结果展示

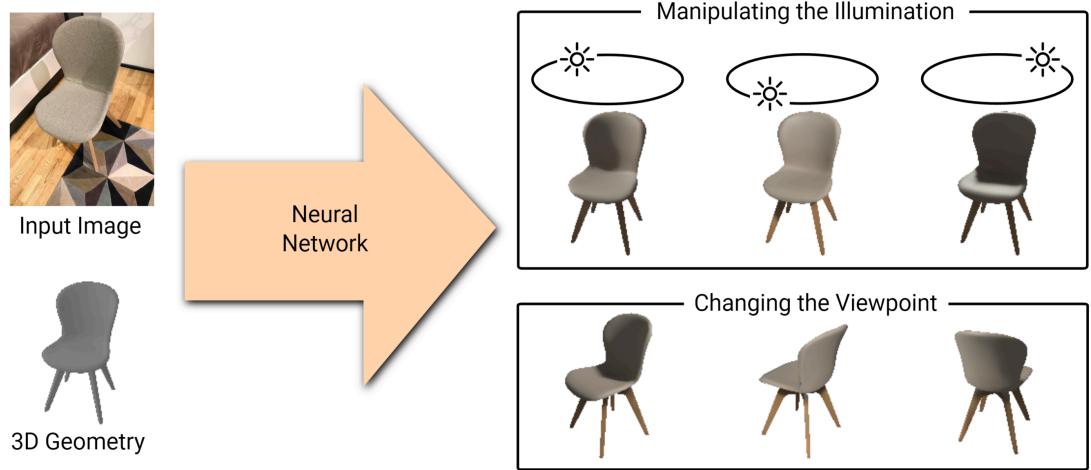


- ▪ 可以看到，这种方法的表示结果是非常细腻的；同时，我们还可以考虑将其应用在体素超分辨率任务上——通过现有的粗糙的 3D 模型训练分类器，然后用 MISE 来计算出细腻的 3D 模型

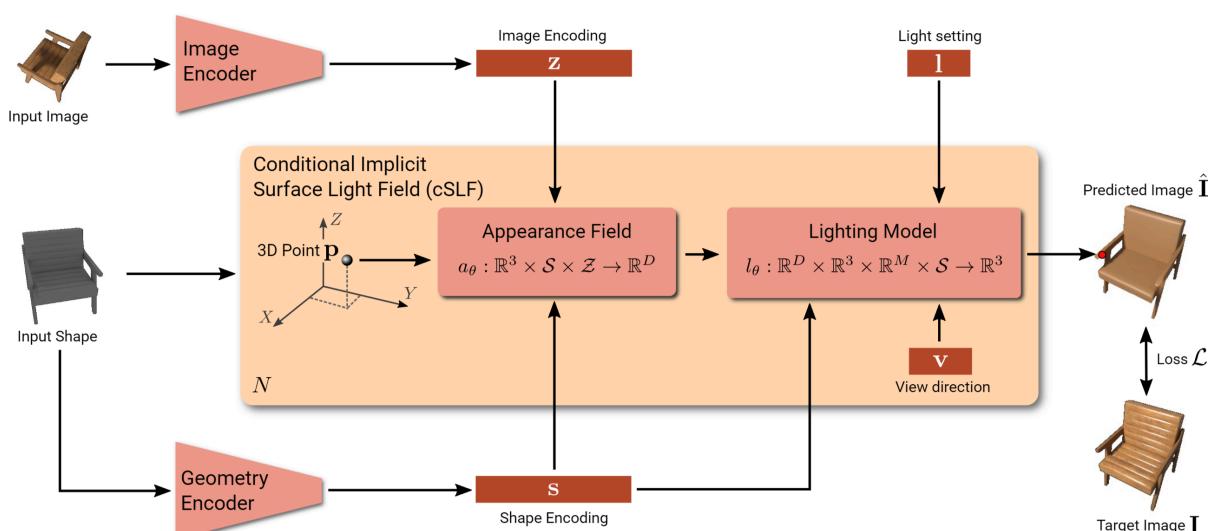
### 9.1.1 Representing Materials and Lighting

现在，我们已经知道了如何利用 Occupancy 网络来描述形状，既然它是一个隐式的表示方式，那么我们是否可以将这种想法应用于其它信息的描述，例如材料和光线？

- 问题定义 [Oechsle, Niemeyer, Mescheder, Strauss and Geiger: Learning Implicit Surface Light Fields. 3DV, 2020.](#)



- ■ 我们希望从输入的图像以及 3D 几何信息中，通过神经网络预测新的视角或光照条件下的渲染结果，包括移动光源、移动视角两种操作
- Conditional Surface Light Field (条件表面光场)
  - 渲染方程  $\int_{\Omega} \text{BRDF}(\mathbf{p}, \mathbf{v}, \mathbf{l}) \cdot \mathbf{l}(\mathbf{s}) \cdot \mathbf{n}^T d\mathbf{s}$
  - 上面是我们熟悉的渲染方程，在这里我们不直接使用这个渲染方程，而是构建一个条件表面光场 (cSLF)。
  - $L_{cSLF}(\mathbf{p}, \mathbf{v}, \mathbf{l}) = R^3 \times R^3 \times R^M \rightarrow R^3$
  - [!\[\[lec\\_09\\_coordinate\\_based\\_networks.pdf#page=22&rect=52, 26, 402, 118\]](#)



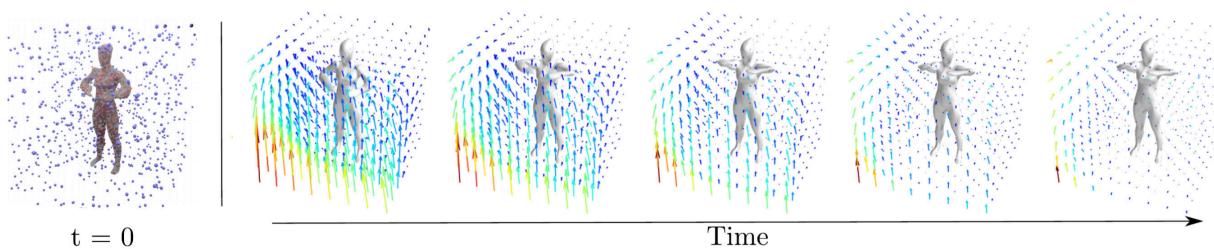
- ◦ 具体的过程，分别将图像和 3D 几何形状进行编码为特征向量  $z$  和  $s$ ，具体来说由外观场和光照模型组成，外观场捕捉该点的外观信息

(如颜色、纹理) , 然后与其它输入信息一起输入光照场, 最后输出 3D 点的最终 RGB 颜色值

### 9.1.2 Representing Motion

下面我们讨论一下, 尝试将这种想法应用于 Motion 的表示上

- Occupancy Flow [Niemeyer, Mescheder, Oechsle and Geiger: Occupancy Flow: 4D Reconstruction by Learning Particle Dynamics. ICCV, 2019.](#)



- 简单的想法是将 Occupancy 网络直接扩展至 4D, 因为 Motion 是一个在 3D 空间不连续, 随着时间维度变化的数据; 然而, 将  $f_\theta$  扩展为 4D 是一件非常困难的事
  - 因此, 我们考虑构建  $t = T$  时刻的 Motion, 从而只构建一个任意时刻通用的 3D Occupancy 网络
  - 利用一个时间和空间连续的向量场来表示 motion
  - 我们可以通过一个 ODE (常微分方程) 建立 3D 轨迹  $s(t)$  与速度  $v$  的关系  
\$\$ \frac{\partial \mathbf{s}(t)}{\partial t} = \mathbf{v}(\mathbf{s}(t), t)

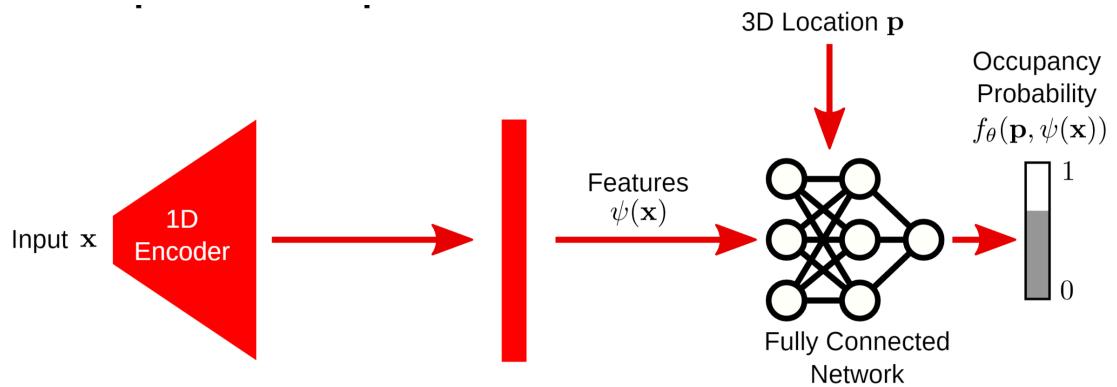
$$\frac{\partial \mathbf{s}(t)}{\partial t} = \mathbf{v}(\mathbf{s}(t), t)$$

- ![[lec\_09\_coordinate\_based\_networks.pdf#page=26&rect=61, 24, 391, 226]

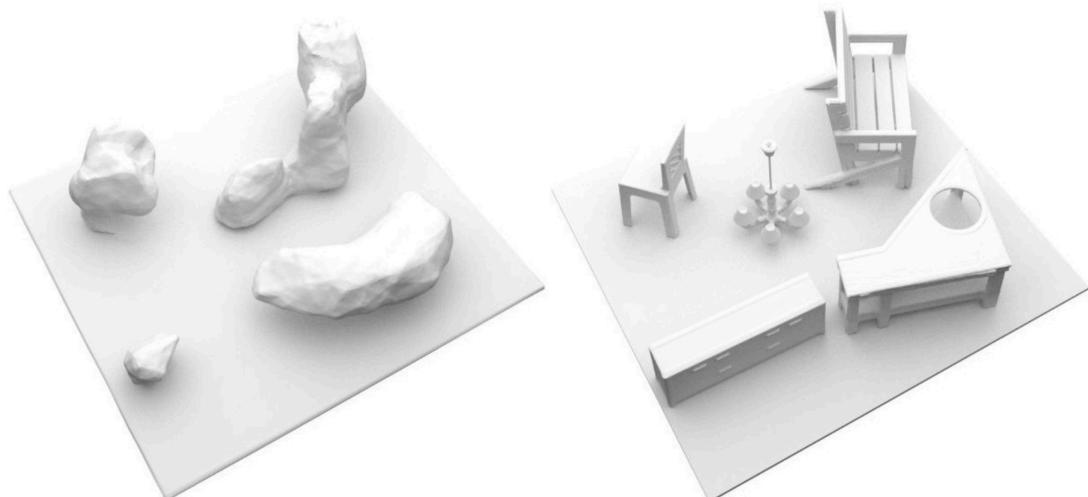
### 9.1.3 Representing Scenes

接下来我们考虑, 是否可以利用 Occupancy 网络来表示更加大尺度的对象, 例如整个场景

- 局限性



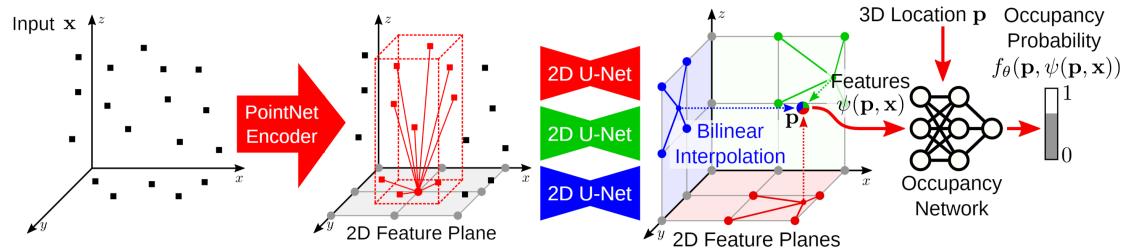
- ■ 到目前为止，我们会发现 Occupancy 网络存在的几个局限性——它总是对全局潜在特征进行编码，因此它会忽略局部信息，导致重建的几何形状出现过平滑，难以捕捉精细细节或复杂结构
- 它采用全连接结构，不具备平移等变性，无法很好地利用输入空间中的几何特征，对大规模或复杂场景的表现不佳



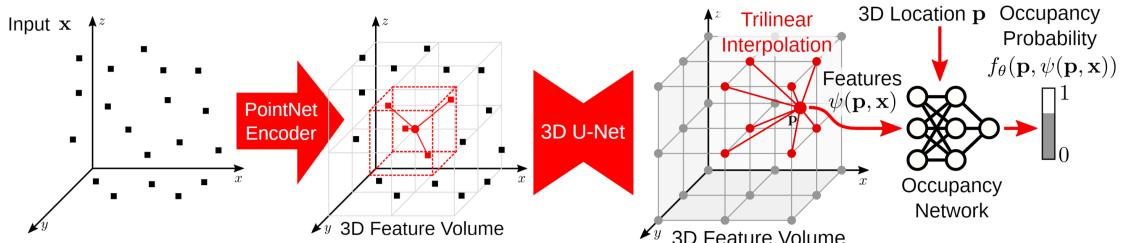
- ■ 隐式的模型表示对于简单的物体建模很好，但是对于像是上面复杂的场景建模效果很糟糕，左图是一个已经训练了很久的模型预测结果，可以看到物体的细节丢失很严重

因此，很自然可以想到，我们希望将 Occupancy 网络和之前的卷积网络的优点结合在一起

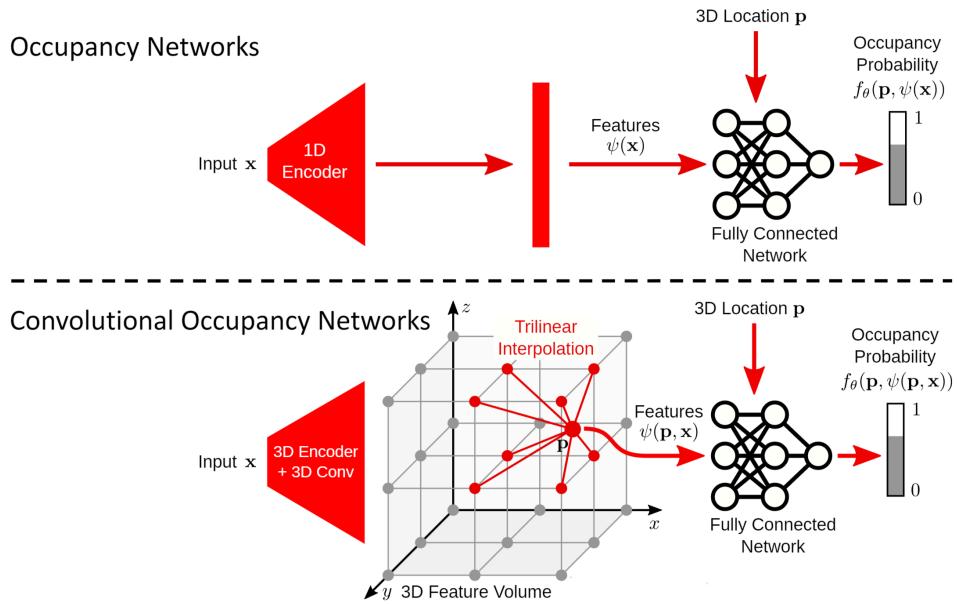
- Convolutional Occupancy 网络 [Peng, Niemeyer, Mescheder, Pollefeys and Geiger: Convolutional Occupancy Networks. ECCV, 2020.](#)



- ■ 2D 平面编码器
  - 首先，将 3D 点云的信息利用 PointNet 进行编码，提取局部特征并将点云投影到标准化的 2D 平面，输出为二维特征平面
- 2D 平面解码器
  - 通过 2D U-Net 对特征平面进行处理，学习更精细的几何特征
  - 查询点的特征，通过双线性插值从 2D 特征平面中提取
- Occupancy 概率预测
  - 最终通过一个比较浅的 Occupancy 网络  $f_\theta(\mathbf{p}, \psi(\mathbf{p}, \mathbf{x}))$  计算 3D 点  $\mathbf{p}$  的占据概率
- 同样的，我们也可以将这种思想应用于 3D 情况



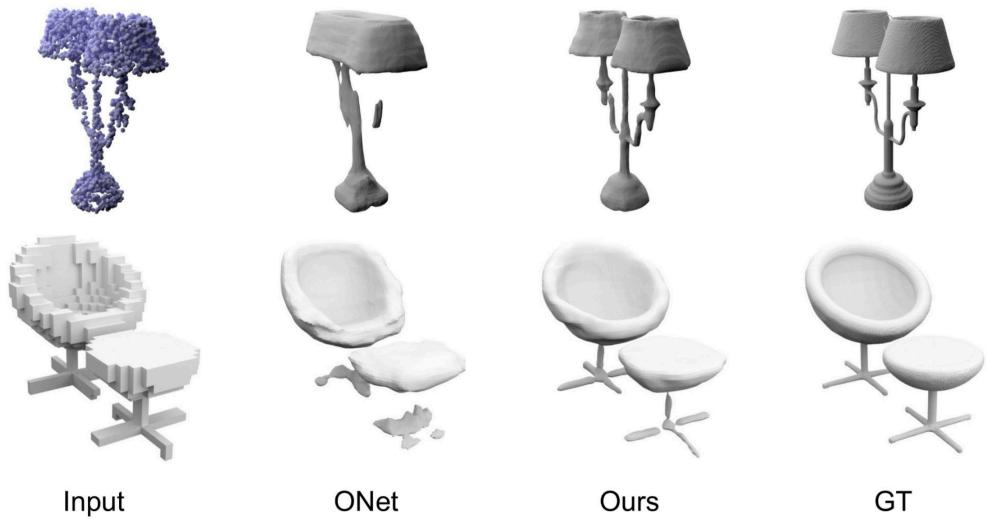
- 3D 体积编码器
  - 仍然输入 3D 点云数据，PointNet 提取点云的局部特征，并映射到标准化的 3D 体积空间（比较粗糙的体积空间）
- 3D 体积解码器
  - 使用 3D U-Net 对 3D 特征体积空间进行卷积处理，进一步提取全局和局部特征
  - 查询点的特征，通过三线性插值，从 3D 特征体积空间中提取
- Occupancy 概率预测
  - 最终仍然是通过一个比较浅的 Occupancy 网络来计算查询点  $\mathbf{p}$  的占据概率
- Occupancy 网络 VS 卷积 Occupancy 网络



- 相比于原始的 Occupancy 网络，卷积版本不再将输入编码为一个固定长度的特征向量，而是编码为可以被卷积网络处理的 3D 特征体积空间，然后输入一个较浅的 Occupancy 网络来预测占据概率

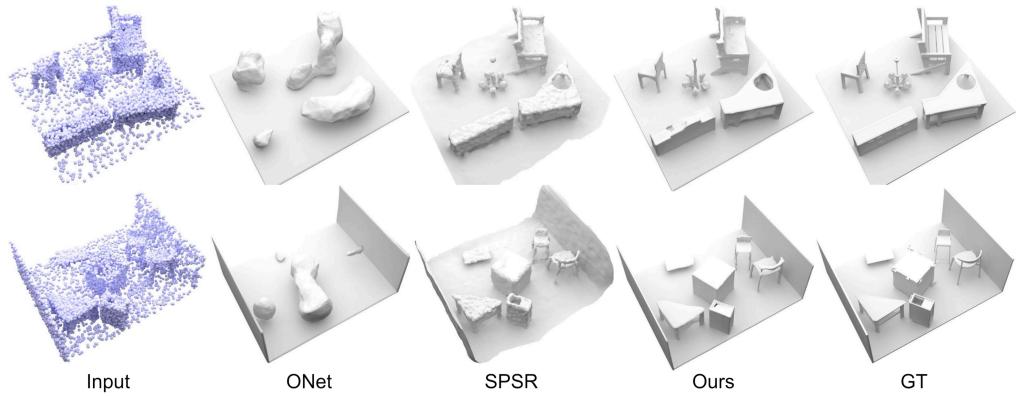
## • 结果展示

- 对象物体的重建效果

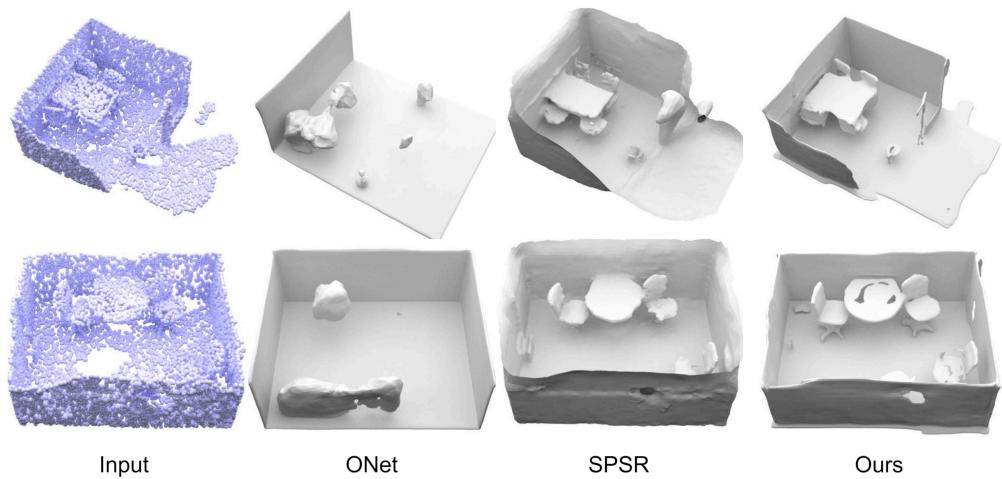


- ○ 相比原始的 Occupancy 网络，卷积版本有更高的精度和更快的收敛速度，可以看到它更好的保留了物体的局部细节

- 场景级别的重建效果

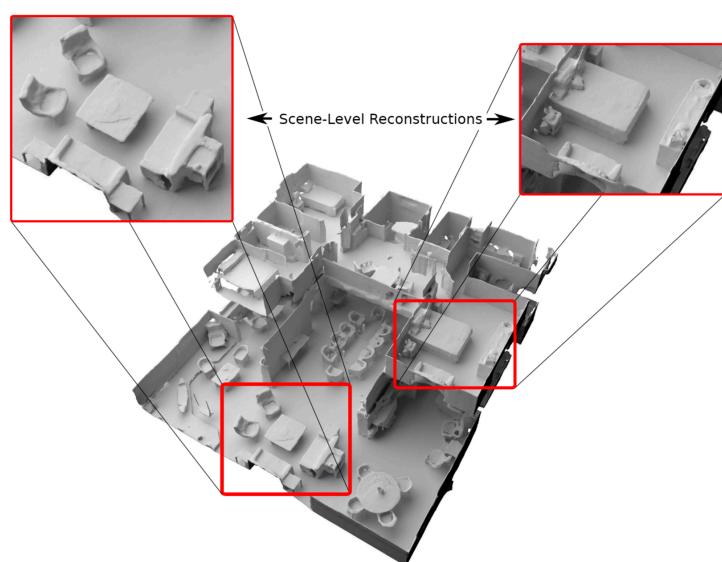


- 除此以外，卷积版本的 Occupancy 网络还可以在人工构造的房间数据集上训练，用来预测真实的房间结构——上面的模型预测结果基于一个人工构造的数据集



- 这里的结果基于 ScanNet，一个扫描真实房间得到的数据集，可以看到，它至少比原始的 Occupancy 网络的预测结果可信的多
- 大尺度重建

- (Matterport3D 数据集上的结果)



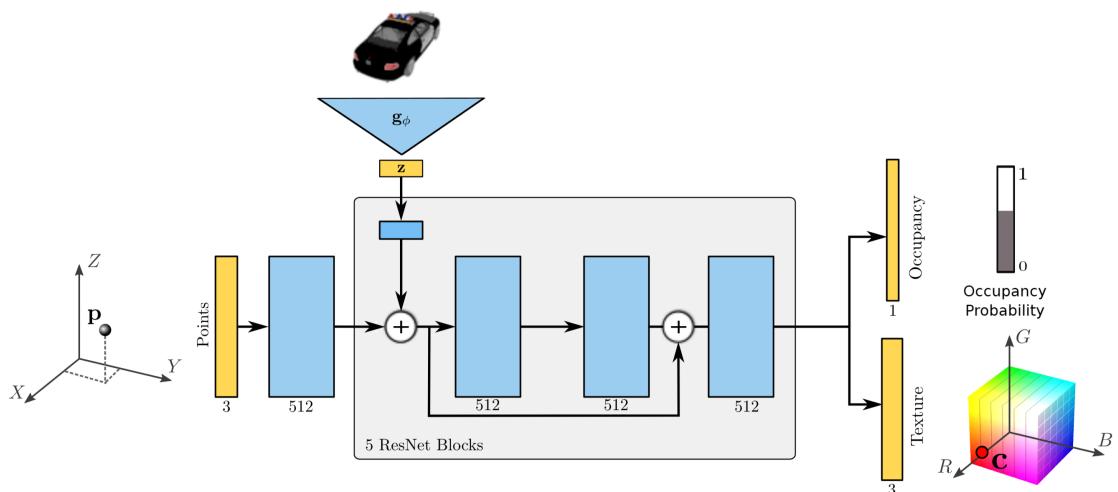
- 对于非常大尺度的重建，我们也可以将该想法扩展为一个完全卷积的模型，在人造的数据集上进行训练
- 由于尺度很大，因此它不适合全部载入 GPU 内存中，我们可以采用滑动窗口的推理方式，分别对每个窗口部分进行推理
- 这样的想法可以将其缩放为任何的场景尺寸

## 9.2 Differentiable Volumetric Rendering

到目前为止，我们讨论的都是将 3D 点转换为一种隐式的 3D 表示方式，我们需要已知这些 3D 点是在物体内还是物体外；然而这些已知信息在现实中是非常难以获取的——因此，我们考虑，是否可以将一组 2D 的 RGB 信息直接转换为这种 3D 的隐式表示方式？

### 9.2.1 Learning from Images

- DVR: Differentiable Volumetric Rendering

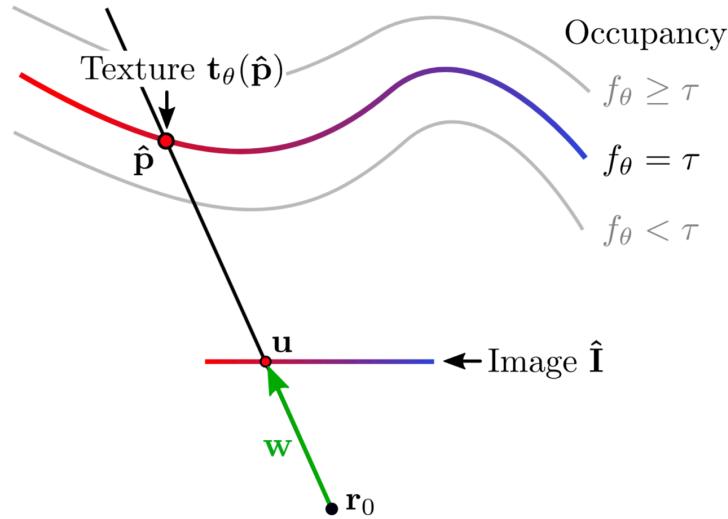


- ■ 首先，将物体通过一个全局编码器得到一个全局的特征向量  $z$ ，同时输入一组 3D 点，通过 PointNet 编码出特征，然后输入一个 5 个块组成的残差网络；和之前不同的是，我们有两个较浅的头网络，分别用于预测 Occupancy 概率和纹理特征——就像我们在 Lecture 6 里面讲到的，如果我们想要从多个视角还原 3D 模型，我们就需要同时还原出形状特征以及纹理特征

如果我们希望该网络正常运作——正常的前向传播和反向传播，我们就需要为该模型定义渲染操作，也即该模型到底如何生成最后的可视化的渲染后的图像

## 9.2.2 Forward Pass (Rendering)

- DVR: Differentiable Volumetric Rendering

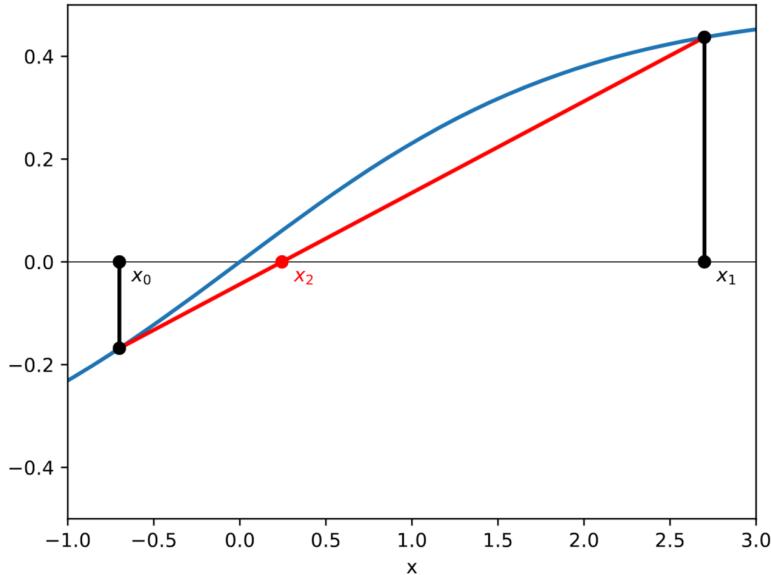


- 对于相机中心  $r_0$ , 图像平面  $\hat{I}$  上的某一点像素  $u$ , 我们通过连接  $r_0$  和  $u$  得到一个向量  $w$ , 延长这个射线, 按照一定步长来不断查询目标点的 Occupancy 概率——当我们发现查询点的 Occupancy 概率已经大于一个阈值  $\tau$  时, 停止算法
- 我们对所有的像素都进行上面这个过程, 找到满足阈值  $\tau$  的点  $\hat{p}$
- 在找到的表面点  $\hat{p}$  上, 计算对应的纹理值  $t_\theta(\hat{p})$ , 也即表面颜色; 然后将该值插入像素  $u$ , 生成最终渲染的图像

这是一个非常简单的过程, 其中按照一定步长来寻找表面点  $\hat{p}$  的方法, 我们可以采用另一种方法快速求解——它称为割线法 (Secant Method)

- Secant Method

- 割线法是一种很古老的数学方法, 它不要求解梯度就可以求出函数  $f_\theta(\mathbf{p}) = \tau$  的根



- 例如这种情况，我们选择两个点  $x_0$  和  $x_1$ ，通过线性插值来计算两个点  $(x_0, f(x_0)), (x_1, f(x_1))$  之间的割线 \$\$

$$y_{\{2\}} = \frac{f(x_{\{1\}}) - f(x_{\{0\}})}{x_{\{1\}} - x_{\{0\}}} (x_{\{2\}} - x_{\{1\}}) + f(x_{\{1\}})$$

- 当  $y_2 = 0$  时，我们将对应的  $x_2$  代入，即可得到新根的估计值  $x_2 = x_1 -$

- 然后我们将  $x_{\{2\}}$  更新为新的  $x_{\{1\}}$ ，然后原来的  $x_{\{1\}}$  更新为  $x_{\{0\}}$ ，然后继续寻找新的  $x_{\{2\}}$ ，正如下面两幅图的过程所示

- !

```
[[lec_09_coordinate_based_networks.pdf#page=44&rect=248,52,441,200|lec_09_coordinate_based_networks, p. 36|400]] !
```

```
[[lec_09_coordinate_based_networks.pdf#page=45&rect=247,50,440,198|lec_09_coordinate_based_networks, p. 36|400]]
```

### 9.2.3 Backward Pass (Differentiation)

首先简单介绍一下偏导数和全导数的概念

- 假设函数  $f(x, y)$  中，变量  $y$  同时也是  $x$  的函数
  - 偏导数定义为 \$\$

$$\frac{\partial f(x,y)}{\partial x} = \frac{\partial f}{\partial x}$$

- 例如，对于  $f(x,y) = xy$ ，则其偏导数  $\frac{\partial f(x,y)}{\partial x} = y$  - 全导数定义为

$$\frac{df(x,y)}{dx} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial x}$$

- 例如， $f(x,y) = xy, y = x$ ，则  $\frac{df}{dx} = y + x = 2x$  > 偏导数表示固定其它

- 隐函数与隐式微分

- 隐函数定义为  $f(x,y)=0$

- 其中  $y$  仅通过隐式关系定义，不能直接写成显式函数  $y = y(x)$ ；对于隐式

- 举个例子，对于圆的方程

- $x^2 + y^2 = 1$ ，对两侧求导结果  $2x + 2y \frac{dy}{dx} = 0$ ，解出  $\frac{dy}{dx} = -\frac{x}{y}$

- !

- $x^2 + y^2 = 1$ ，对两侧求导结果  $2x + 2y \frac{dy}{dx} = 0$ ，解出  $\frac{dy}{dx} = -\frac{x}{y}$

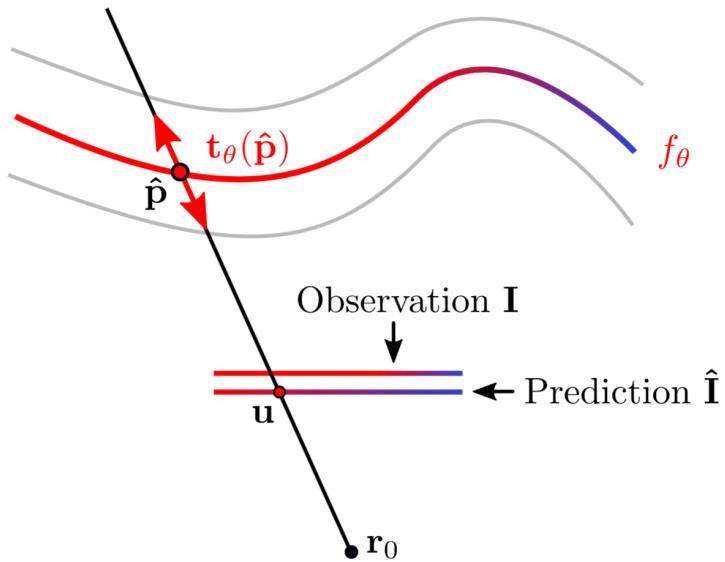
- 注意到，这里的求导结果中还是包含  $y$  项的，也就是说它不是一个函数，而是隐式地表示了一个曲线——表示隐函数  $x^2 + y^2 = 1$  的切线斜率

---

- 然后我们再回到反向传播的具体过程中

- Backward Pass

-



- 定义损失函数  $\mathcal{L}(\hat{I}, I)$  为渲染图像  $\hat{I}$  与目标图像  $I$  的像素差  $\$$   

$$\mathcal{L}(\hat{I}, I) = \sum \limits_{\textbf{u}} |\hat{I}_{\textbf{u}} - I_{\textbf{u}}|$$

```

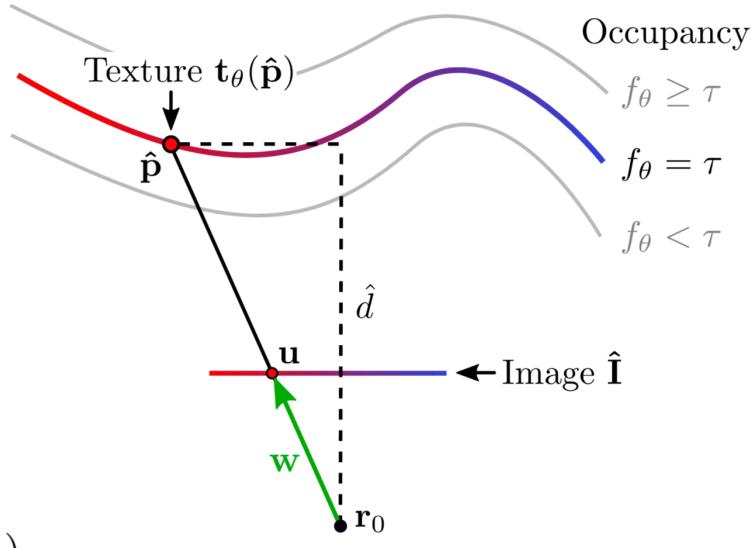
\begin{aligned}
& \frac{\partial \mathcal{L}}{\partial \theta} = \sum_u \\
& \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{l}}_u} \cdot \dots \\
& \frac{\partial \hat{\mathbf{l}}_u}{\partial \theta} = \textcolor{red}{\frac{\partial \theta(\hat{\mathbf{p}})}{\partial \theta}} + \textcolor{red}{\frac{\partial \hat{\mathbf{p}}}{\partial \theta}} \\
& \cdot \textcolor{red}{\frac{\partial \hat{\mathbf{p}}}{\partial \theta}}
\end{aligned}

```

—其中，由于 $t_\theta(\hat{p})$ 是 $\theta$ 的函数， $\hat{p}$ 也是 $\theta$ 的函数，因此求这里的全导数时

- 至此，如果我们能够将后面这个式子解出来，那么一切都会很完美——我们定义了损失函数，并且能够求得损失函数对于参数的梯度；并且由上文可以看出，我们求得的是一个解析解；接下来我们只需要考虑这个式子该如何解出来

这里有全导数的一个应用，因为函数  $t_\theta$  本身包含因子  $\theta$ ，与此同时它包含的因子  $\hat{\mathbf{p}}$  也包含  $\theta$ （这里把 depends on 译为包含...因子）



- 我们考虑射线  $\hat{\mathbf{p}} = \mathbf{r}_0 + \hat{\mathbf{d}}\mathbf{w}$

- 我们知道， $\hat{\mathbf{p}}$  点必须满足  $f_\theta(\hat{\mathbf{p}}) = \tau$ ，相对于  $\theta$ ，我们可以得到

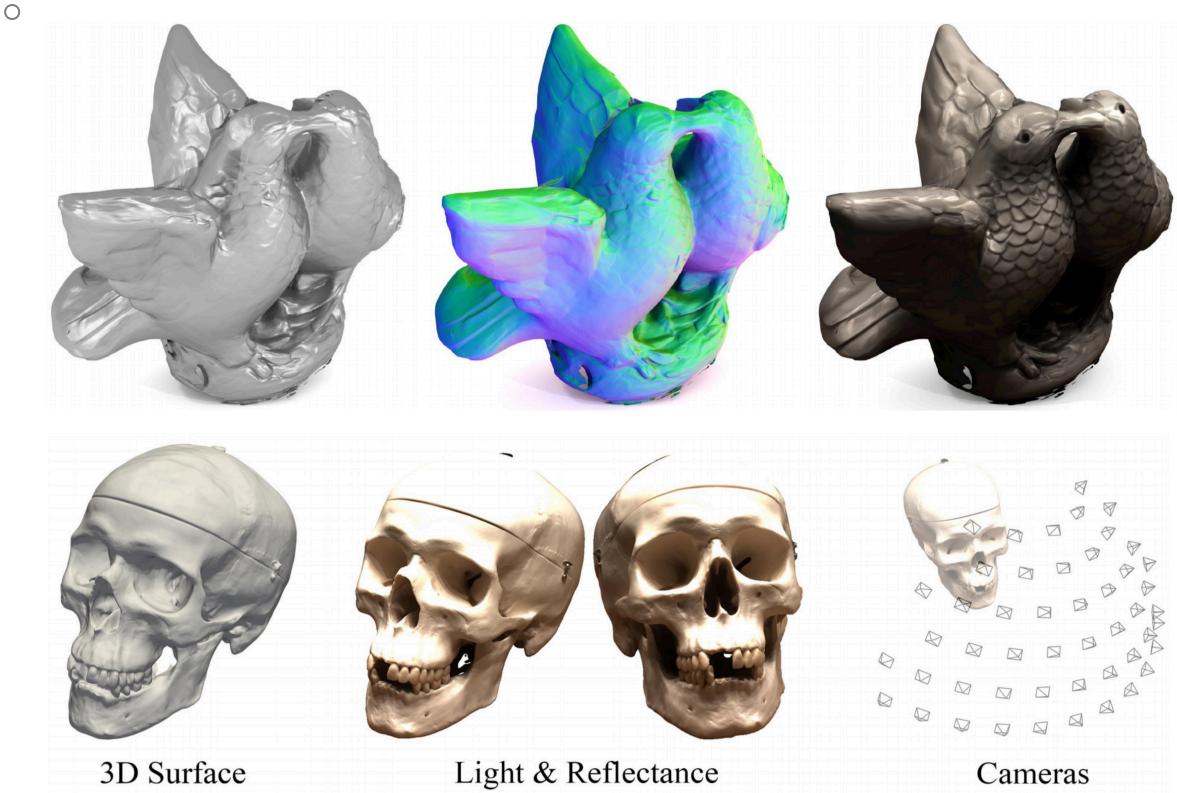
$$\begin{aligned} \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \theta} + \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \cdot \frac{\partial \hat{\mathbf{p}}}{\partial \theta} &= 0 \quad \text{iff} \\ \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \theta} + \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \cdot \frac{\partial \hat{\mathbf{p}}}{\partial \theta} &= 0 \end{aligned}$$

- 将  $\hat{\mathbf{p}}$  的定义代入，得到第二个式子 - 解出深度的梯度，得到

$$\begin{aligned} \frac{\partial \hat{\mathbf{p}}}{\partial \theta} &= \mathbf{w} \\ \frac{\partial \hat{\mathbf{d}}}{\partial \theta} &= -\mathbf{w} \left( \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \right)^{-1} \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \theta} \end{aligned}$$

- 至此，这些式子都很容易计算，因此我们可以以解析解完成整个前向传播和

- 结果展示



- [Lipman 小组](#)对这个模型还进行了扩展——考虑了依赖视图外观的调节，这允许基于外观的光滑程度与光反射率进行调节，而不是仅仅依赖于 Lambertian 假设

### 9.3 Neural Radiance Fields

利用 Occupancy 网络表示形状和外观这一想法最受欢迎的后续工作之一——称为神经网络辐射场(Neural Radiance Fields)

- NeRF 是一种新颖的视图合成方法 [Mildenhall, Srinivasan, Tancik, Barron, Ramamoorthi and Ng: NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. ECCV, 2020.](#)

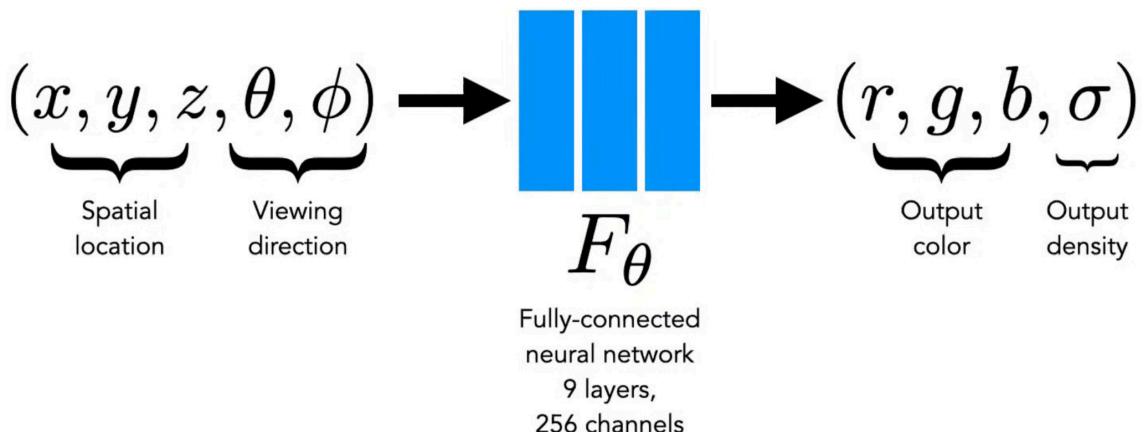


Inputs: sparsely sampled images of scene

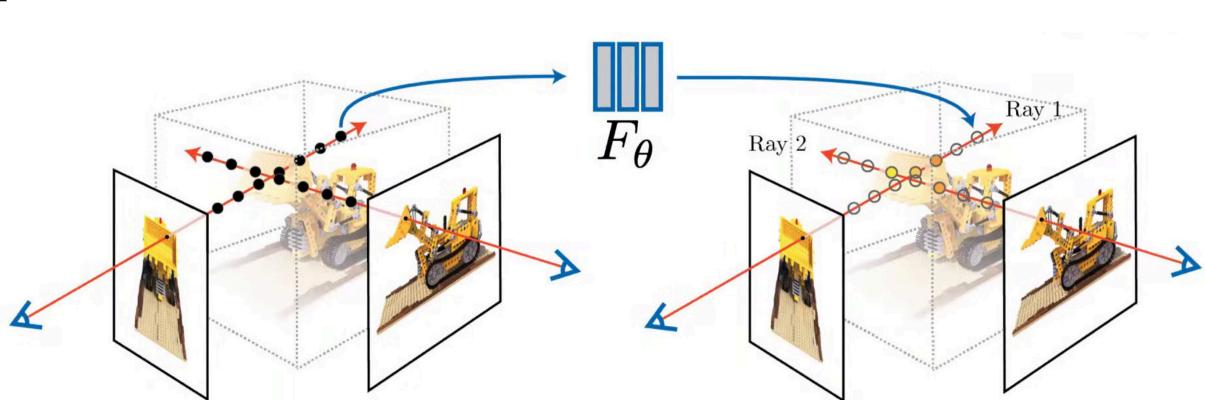
Outputs: new views of same scene

- 它不像是传统的重建方法，而是考虑如何从一组稀疏的多视角输入图像来重建出物体的新的视角——重建物体的 3D 模型并不是最终目标，目标是渲染的新视角的质量

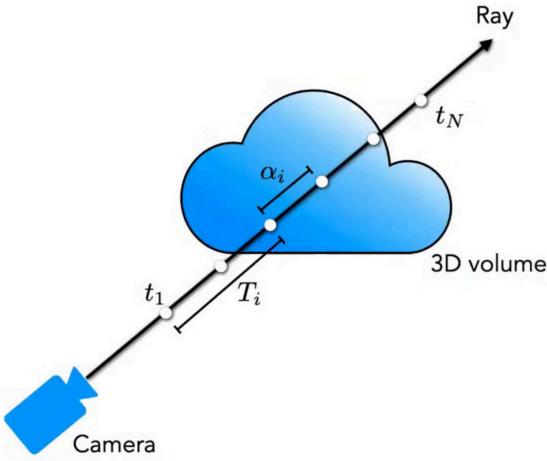
- 这种思想的好处在于，它适用于一般场景——不像传统的很多 3D 重建方法，必须要有一些假设或者前提条件



- ■ 它的模型结构采用 ReLU MLP，共 9 层 MLP，每层有 256 个通道
  - 输入空间坐标位置、观察视角方向（俯角+仰角）
  - 输出 RGB 颜色值，以及一个密度值  $\sigma$ 
    - 密度值  $\sigma$  描述 3D 点的透明度（或者说实体率），因此它也允许 NeRF 对如雾、玻璃等半透明物体建模
  - 输入观察方向，允许模型对与视角相关的场景特性有更好的建模效果
  - 实际操作中，观察视角方向会被作为一个标准化的 3D 向量  $d$  表示
- Volume Rendering



- NeRF 同样使用体积渲染技术，但是与之前查找表面点  $\hat{p}$  的策略不同，它使用一种称为  $\alpha$  合成的方法，很类似于传统 CV 领域中的光线追踪技术
- 沿着射入的光线采样若干点，查询辐射场来获得颜色、密度值，然后应用  $\alpha$  合成来计算该光线源头像素的颜色



- 对于光线  $r(t) = o + td$ , 有 \$\$

$$C \approx \sum \limits_{i=1}^N \underbrace{T_i}_{\text{weights}} \underbrace{\alpha_i}_{\text{alpha}} \underbrace{c_i}_{\text{colors}}$$

$$\underbrace{T_i}_{\text{weights}} = \prod \limits_{j=1}^{i-1} (1 - \alpha_j)$$

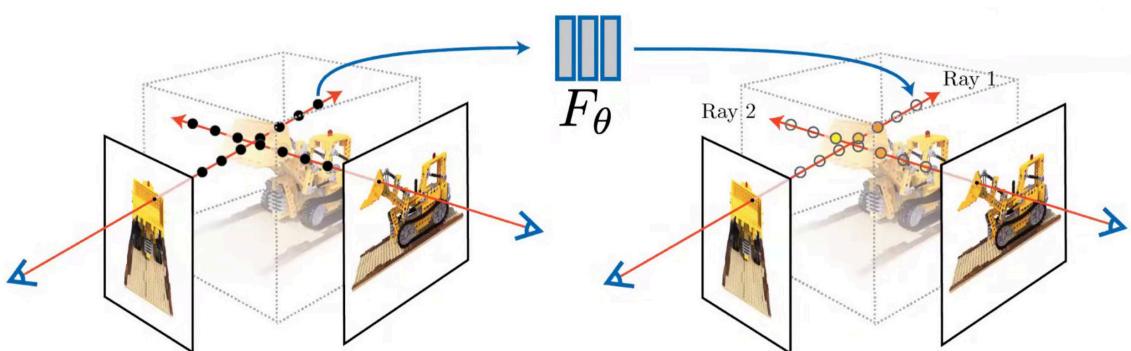
- 计算有多少光在射线上被阻挡

$$\underbrace{T_i}_{\text{weights}} = \prod \limits_{j=1}^{i-1} (1 - \alpha_j)$$

- 我们想象一下，在射入物体之前， $\alpha$  值都应该会很小，接近于0，那么在这

- 当第  $i$  段射线所在的位置物体密度很大时， $\alpha$  值就会很大，对应的光贡献值减小，与我们前面的结论一致

- NeRF 训练过程

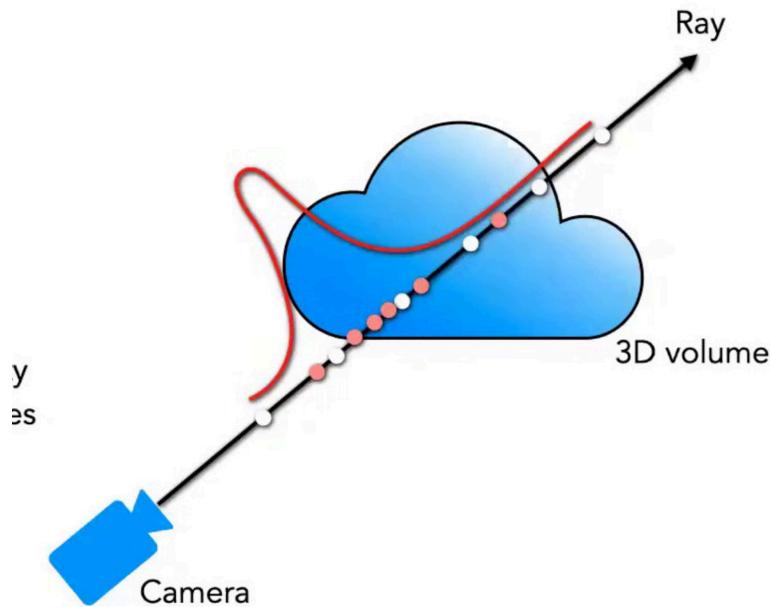


- NeRF 的训练，我们只需要最小化如下的损失函数 \$\$

$$\underbrace{\min}_{\theta} \sum \limits_i \text{render}_i(F(\theta) - \underbrace{I_i}_{\text{target}})^2$$

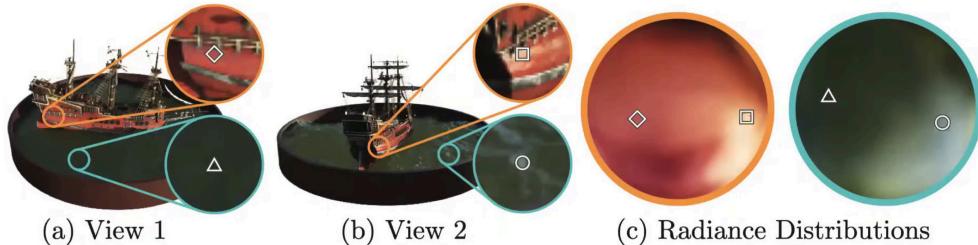
— 也即，最小化第  $i$  个像素的渲染结果和观察结果之间的 L2 损失值即可，

- Two Pass Rendering



- ■ 如果一条线上采样的点特别多，则模型整体的运行速度会非常慢，NeRF 采用了两步渲染的策略——也即，首先按照比较粗略的步长进行采样，找到接近于物体表面的区域，然后再用比较细致的步长进行采样

- 结合视角方向的依赖性



- ■ NeRF 模型支持依赖于视角的效果，例如高光或反射，这些效果通常不满足 Lambertian 假设；不同视角下，同一物体表面的颜色可能会变化，特别是对于光滑或反光材质
- 在 MLP 中，视角信息在后期才被引入，从而避免与几何结构信息发生纠缠
- 通过针对不同的视角方向查询颜色，可以生成不同的辐射分布，不同的颜色分布也体现了 NeRF 在建模非 Lambertian 材质时的能力

- Fourier 特征



NeRF (Naive)



NeRF (with positional encoding)

- ■ 直接输入神经网络的 NeRF 如左侧图所示，它难以捕捉一些高频的细节，例如边缘的清晰度和纹理
- NeRF 采用的另一个有趣的技巧是引入了 Positional Encoding (位置编码)，这是一种 Fourier 特征，对输入点的位置  $\mathbf{x}$  和方向  $\mathbf{d}$  进行编码，编码后输入的数据被扩展到高维空间，使得模型能够更好地表示高频细节
- 一篇后续的工作对这种现象专门做出了解释 [Tancik et al.: Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. NeurIPS, 2020.](#)

- 下面我们举一个具体的例子来说明位置编码的效果



- ○ 这应该是最简单的想法——利用一个简单的 MLP 将像素的位置坐标映射到对应的 RGB 颜色值

Ground truth image

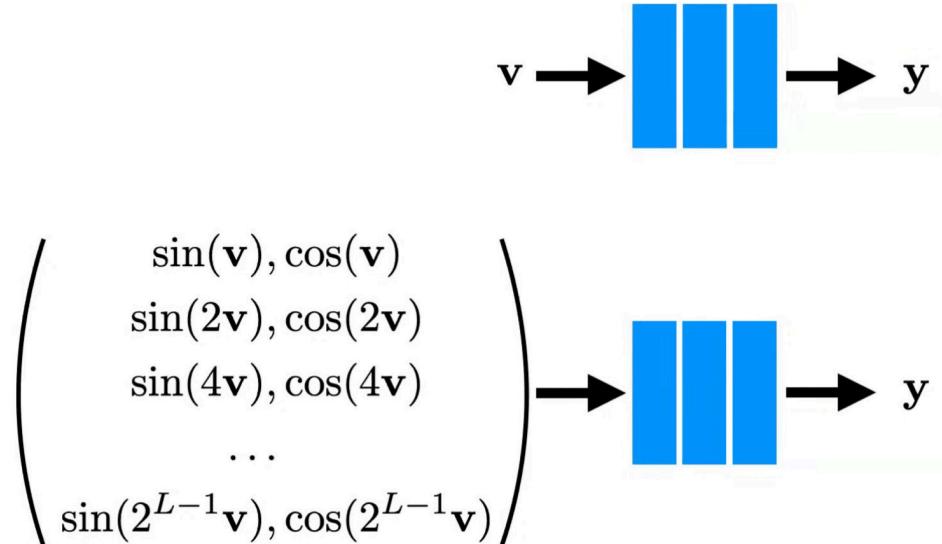


Standard fully-connected net



- ○ 而令人非常震惊的是，即使我们将模型的参数扩大 10 倍，这个模型的参数比我们想要记住的 RGB 图像的像素更多，甚至我们让它收敛几天时间，它也只能得到右侧的一种平滑的结果

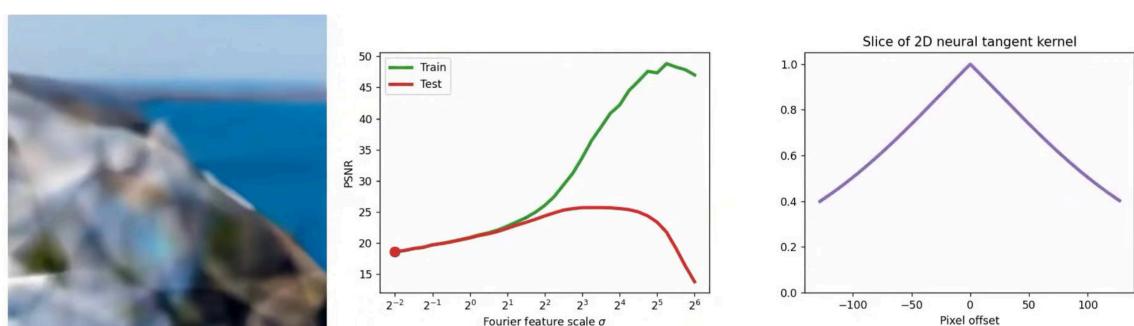
- 对于这种现象，NeRF 的作者想出了位置编码的解决方案，也即不直接将低维数据输入网络，而是先将其编码，映射到高维域中



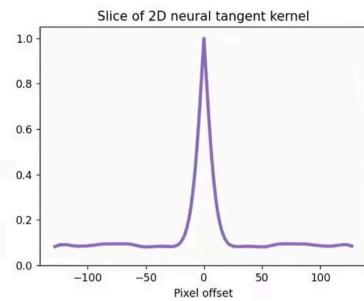
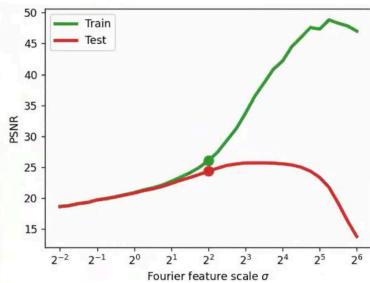
- 如图所示，我们将  $\mathbf{v}$  向量通过 Fourier 变换映射到高维空间，通过多种频率（由参数  $L$  控制）生成正弦和余弦分量，使得网络可以更高效地学习高频函数
- 利用这些 Fourier 特征，网络能够在低维空间中捕捉高频函数变化，从而提高对细节地建模能力



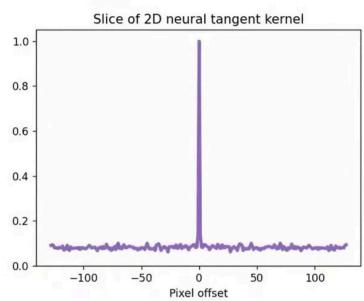
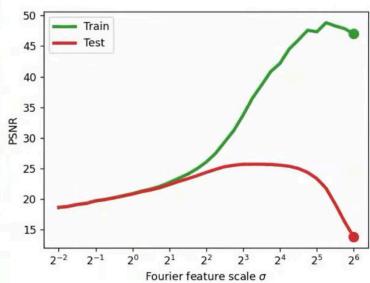
- 结果具有更高的锐度以及更丰富的细节，不再有 ReLU 的伪影，网络结构相同，且具有更快的收敛速度
- Fourier 带宽对欠拟合/过拟合的影响



- 在左侧的极端情况下，结果欠拟合，效果非常模糊



- 在中间的位置时，效果是最清晰的，这也是我们希望寻找的位置



- 在右侧极端时，结果过拟合，出现很多噪点，失去平滑性

- 结果展示



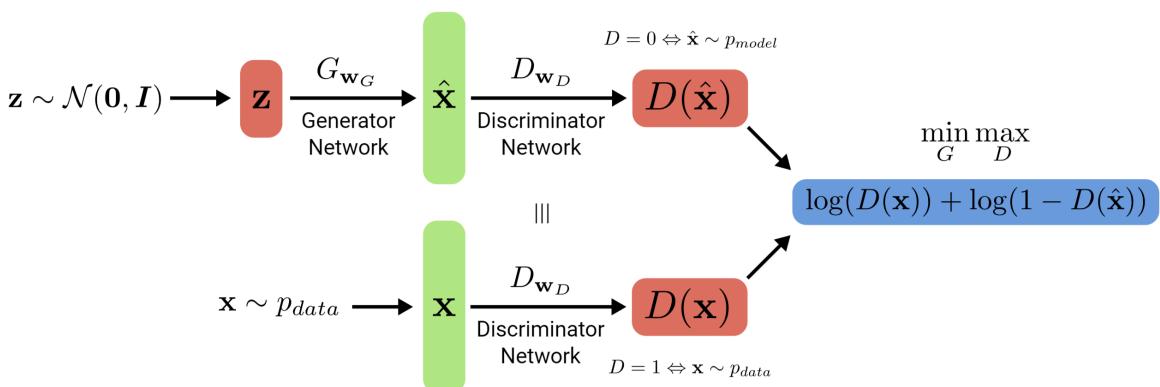
- 由于 Markdown 中不能插入视频，具体的效果可以参考原文献等资料

## 9.4 Generative Radiance Fields

现在我们采用 9.3 介绍的 NeRF 模型结构，尝试构建一个生成式的模型；它不针对特定场景，而是针对一些物品来渲染对应的图像，甚至不需要输入相机姿态

- 首先我们回顾一下 Generative Adversarial Networks (GAN) 的内容
  - 生成式模型，指的是从数据集中学习数据分布得到的分布  $p_{model}$ ，从中生成  $p_{data}$  的模型（也即可以生成与输入数据分布类似的数据的模型）
    - 一些模型显式地评估  $p_{model}$ ，因此它们允许估计样本  $x$  的似然  $p_{model}(x)$
    - 另一些模型，它们只能从  $p_{model}$  生成样本，这一类模型为隐式模型
    - GANs 则是这一类隐式模型的一个突出的例子，它们提供了一个不需要显式似然来训练模型的框架

- Generative Adversarial Networks



- 我们首先从一个噪声分布  $\mathcal{N}(0, I)$  中生成一个向量  $\mathbf{z}$ ，然后将其输入一个生成器网络 Generator，输出一个生成的样本（规格与我们希望生成的样本一致） $\hat{\mathbf{x}} = G(z)$ ；然后输入一个判别器网络 Discriminator，判断输入是否来自真实数据分布，并输出置信度  $D(\hat{\mathbf{x}})$

- 同时，我们将一个真实数据的样本也输入另一个判别器网络（两个判别器网络相同），然后得到一个置信度  $D(\mathbf{x})$

- 然后我们计算这两个置信度之间的差异

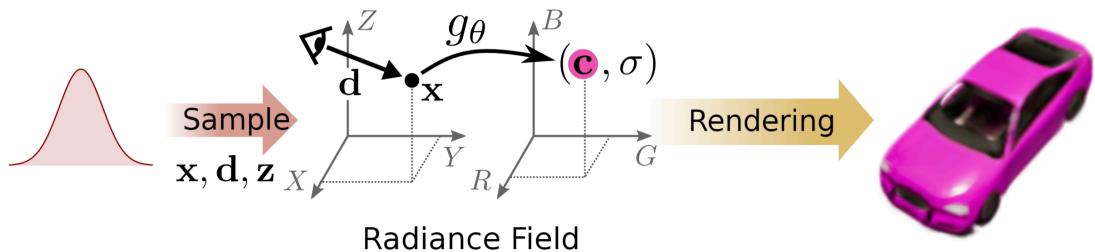
$$\begin{aligned} & \overline{\min_G \quad \max_D} \\ & \{\log(D(\mathbf{x})) + \log(1 - D(\hat{\mathbf{x}}))\} \end{aligned}$$

- 在这里，网络  $D$  和网络  $G$  通过  $\min \max$  两人零和博弈 (*two-player minimax game*)

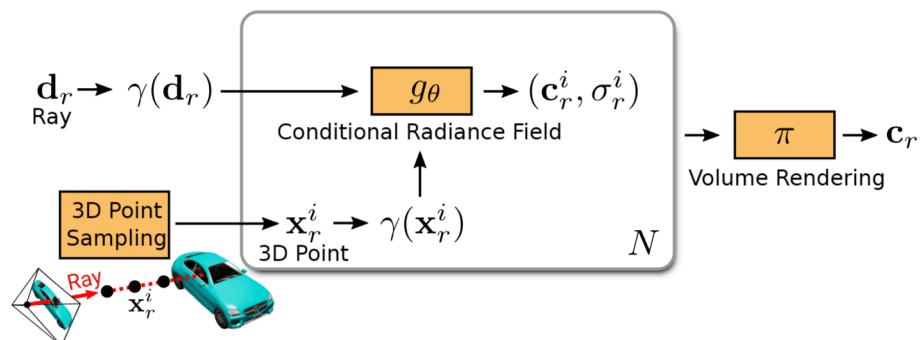
$$\begin{aligned} & \begin{aligned} G^*, D^* &= \arg \min_G \arg \max_D V(D, G) \\ & V(D, G) = \\ & \mathbb{E}_{\mathbf{x} \sim p_{data}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(z)}[\log(1 - D(G(\mathbf{z})))] \end{aligned} \end{aligned}$$

- \*\*理论上来说\*\*，如果  $G$  和  $D$  的容量足够，并且假设优化能够顺利达到极

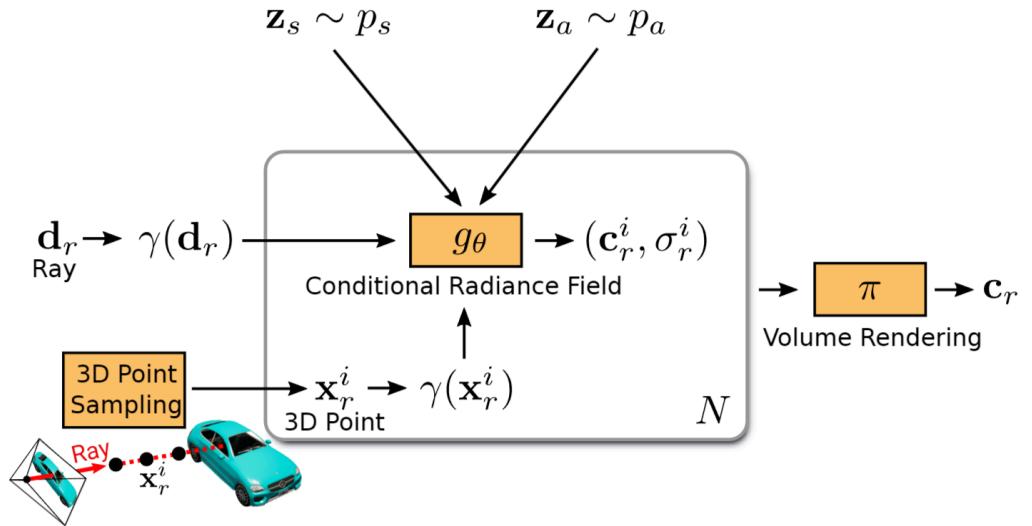
- 现在，让我们将这个想法应用于辐射场中
- GRAF: Generative Radiance Fields [Schwarz, Liao, Niemeyer, Geiger: GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis. NeurIPS, 2020](#)



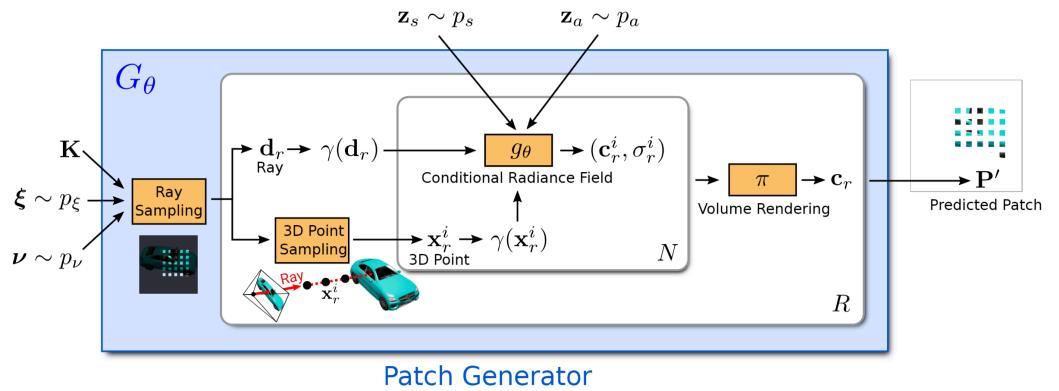
- 我们希望对一个随机的样本进行编码，这个样本应该包含 3D 点位置、观察视角，以及一些外观属性信息（颜色、纹理），还有相机姿态（没有在这里展示出来）
  - 通过辐射场，将这些输入信息映射为颜色和密度信息  $(c, \sigma)$ ，并通过体积渲染得到最后的可视化图像
  - 我们可以在没有 3D 数据标注的情况下，从无结构、未标注的 2D 图像集合中训练
  - 但和 GAN 不同的是，我们没有办法直接构建一个可以区分 3D 结构的判别器，因为训练数据是 2D 图像，而不是 3D 数据——如果我们每一次调用判别器  $D$ ，都要使用一次体积渲染，那么整体的效率会很低
- 因此，这篇文献提出了一个新的方法来解决这一问题



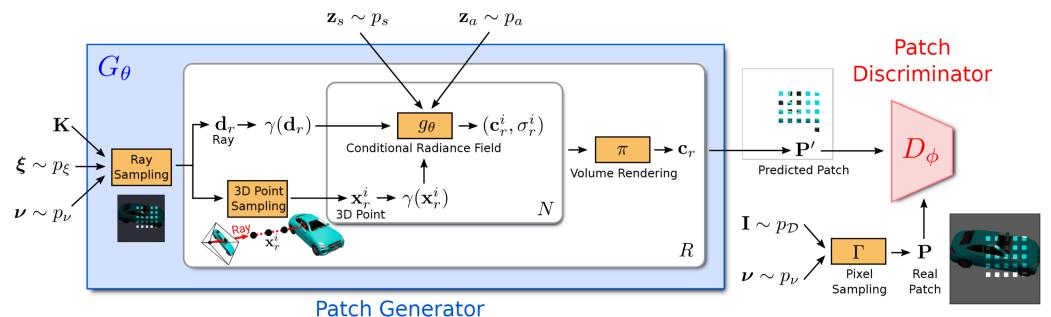
- 这是一个原始的辐射场结构，我们有一个辐射场  $g_\theta$  将 3D 点  $x_r^i$  和观察视角方向  $d_r$  映射为颜色和密度信息  $(c_r^i, \sigma_r^i)$ ，其中  $\gamma$  表示位置编码，正如在 NeRF 中提到的一样
  - 我们将输出的结果推入体积渲染  $\pi$  中，通过沿着每条射线的  $N$  个点的采样，我们可以渲染出像素的颜色  $c_r$



- GRAF 将  $g_\theta$  扩展为一个条件模型，增加了一个形状条件和外观条件
- 这里  $z_s$  是形状信息的潜在编码， $z_a$  是外观属性的潜在编码



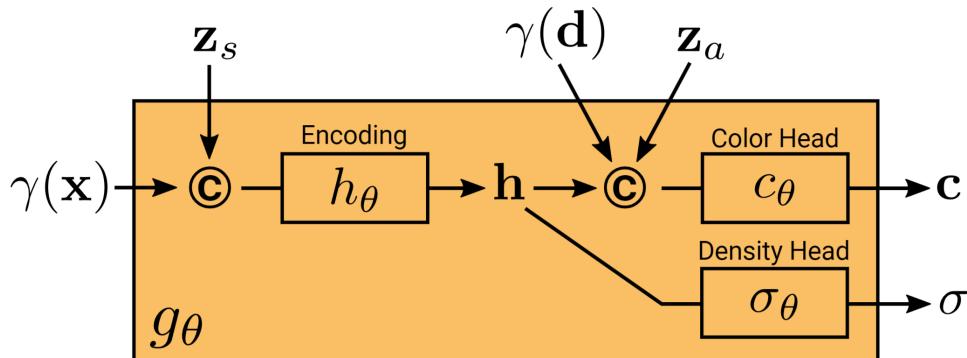
- 我们对相机内参  $\mathbf{K}$ 、相机外参  $\xi$ 、2d 网格  $\nu$ ，随机取值，并生成光线；在光线上均匀采样  $N$  个 3D 点，将采样点和光线方向输入条件辐射场  $g_\theta$ ，输出对应的颜色和密度，然后进行体积渲染
- 在这里，我们取一个 patch，例如  $32 \times 32$ ，然后仅对这个 patch（并非全部像素）出发的射线进行体积渲染，作为  $G$  的输出



- 类似于生成器  $G$ ，我们也可以从真实数据的图像中，取一个同样规格的 patch（这里  $\mathbf{I}$  表示从数据分布  $p_D$  中采样的

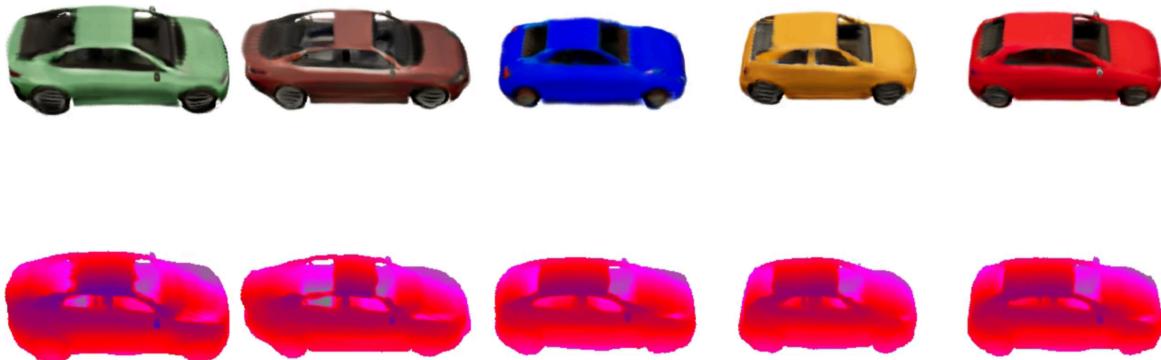
真实图像），然后和 G 输出的 patch 一起输入判别器 D 中——在这里，我们进行判别的只是 2D 的图像，因此我们只需要简单地用一个 4 层的卷积网络即可

- 这里细致讲解一下条件辐射场  $g_\theta$  的结构



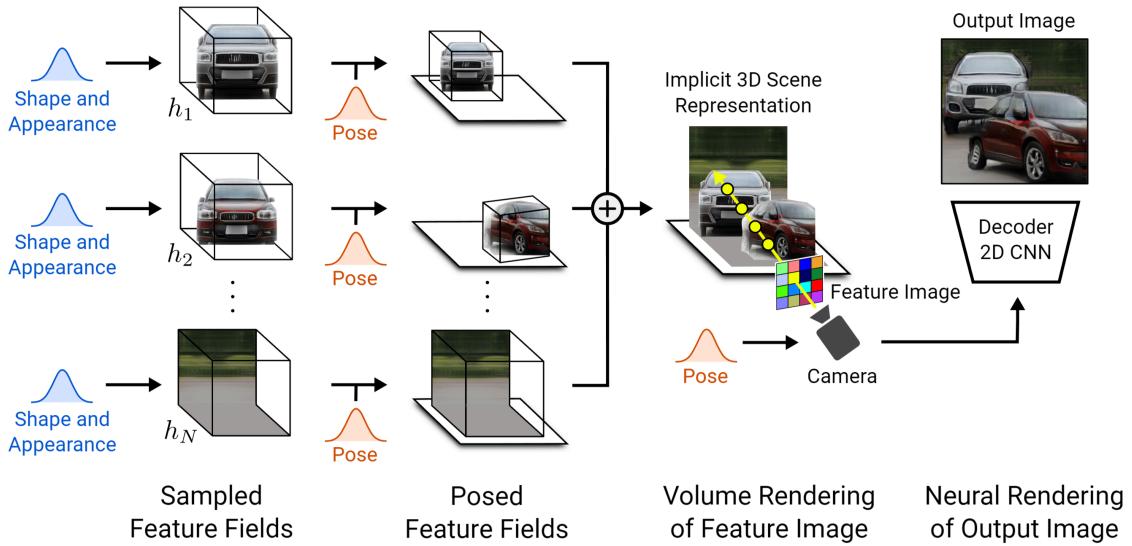
- 记得我们之前说，GRAF 在原始  $g_\theta$  的基础上增加了 2 个条件  $z_s$  和  $z_a$ ，那么为什么可以直接添加呢？它们之间不会有纠缠吗？
  - 体积密度  $\sigma$  仅取决于 3D 点  $\mathbf{x}$  和形状编码  $z_s$
  - 预测颜色  $\mathbf{c}$  仅取决于观察视角  $\mathbf{d}$  和外观属性  $z_a$ 
    - 因此，它们之间不会有什么纠缠的问题
  - 通过引入视角方向  $\mathbf{d}$  以及外观潜在编码  $z_a$ ，模型可以生成符合不同观察角度的图像

- 结果展示



这里，教授展示了他的一位博士生做的视频演示，非常令人印象深刻，可以去原视频看 9.4 节大概 13:42 秒的位置

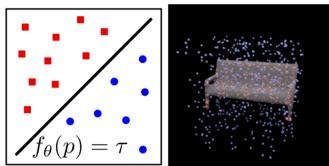
- 最后，来介绍一下一项新的工作，获得了 CVPR2021 的最佳论文奖
- GIRAFFE: Compositional Generative Neural Feature Fields [Niemeyer and Geiger: GIRAFFE: Representing Scenes as Compositional Generative Neural Feature Fields. CVPR, 2021](#)



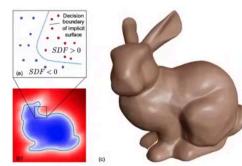
- ■ 对于每个场景对象单独建模（例如车辆和背景），分别生成特征场，描述其形状和外观属性，每个对象的特征表示  $h_1, \dots, h_N$  被分别编码
- 每个对象根据特定的姿态 (Pose) 进行位置化，使其在场景中定位，姿态描述了每个对象在 3D 场景中的位置、方向和大小
- 所有位置化的特征场被组合，形成一个完整的隐式 3D 场景表示，这种表示支持 GIRAFFE 动态建模多对象的场景
- 然后对隐式 3D 场景进行体积渲染，生成结果图像

#### 9.4.1 Summary

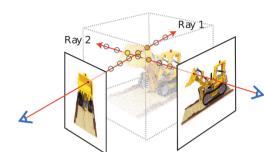
**Occupancy Networks**  
[Mescheder et al. 2019]  
 $(x, y, z) \rightarrow$  occupancy



**DeepSDF**  
[Park et al. 2019]  
 $(x, y, z) \rightarrow$  distance



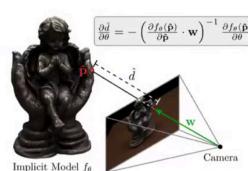
**Neural Radiance Fields**  
[Mildenhall et al. 2020]  
 $(x, y, z, \theta, \phi) \rightarrow$  color, density



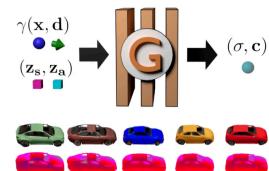
**Scene Representation Networks**  
[Sitzmann et al. 2019]  
 $(x, y, z) \rightarrow$  latent vec. (color, dist)



**Differentiable Volumetric Rendering**  
[Niemeyer et al. 2020]  
 $(x, y, z) \rightarrow$  color, occ.



**Generative Radiance Fields**  
[Schwarz et al. 2020]  
 $(x, y, z, \theta, \phi, \mathbf{z}) \rightarrow$  color, density



- 至此，我们已经了解过了许多的神经网络

- Coordinate-Based 神经网络
  - 它们可以有效输出形状、外观属性、材质、运动的表示
  - 没有离散化，因此可以任意改变模型拓扑结构
  - 可以通过可微分渲染从图像中进行学习
  - 已经有了许多领域的应用，包括重建、合成可视化、机器人等
- 然而
  - 几何特征必须从后处理步骤中提取（例如使用 ONet，至少需要花费 1s 的时间）
  - 仍然不能直接拓展到 4D 空间
  - 全连接的模型结构会导致一些局部细节过度平滑
  - 比较有前景的方向：局部特征 (ConvONet)，更好的输入编码 (NeRF)