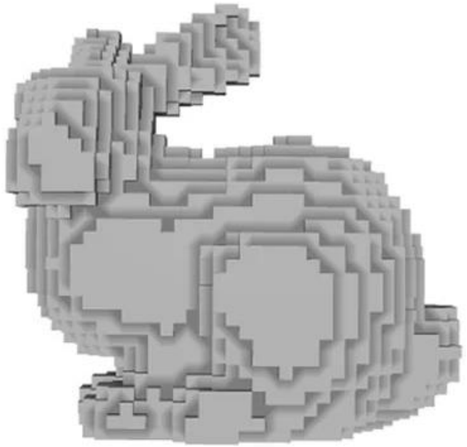
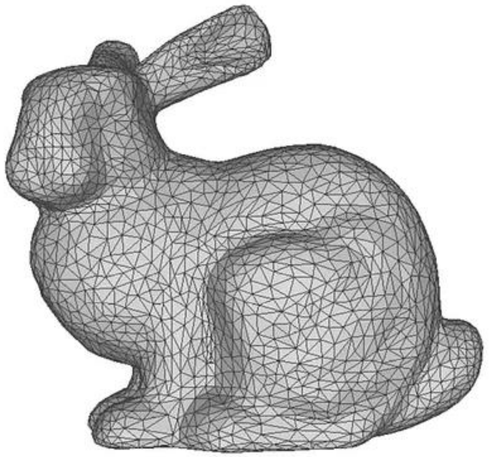
Neural Radiance Field

Previous Class: 3D Representations

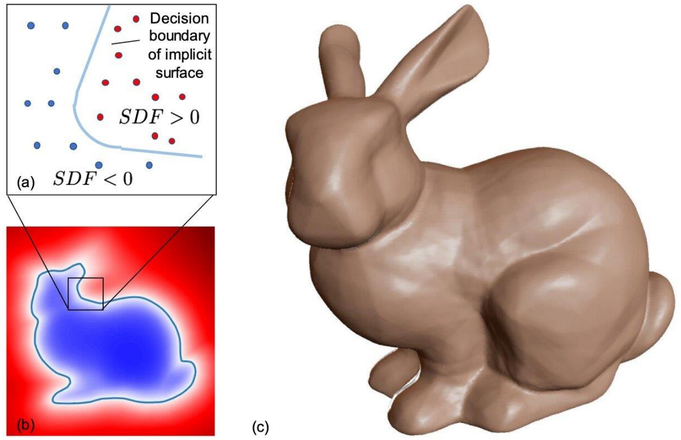
Point cloud



Volumetric



Mesh



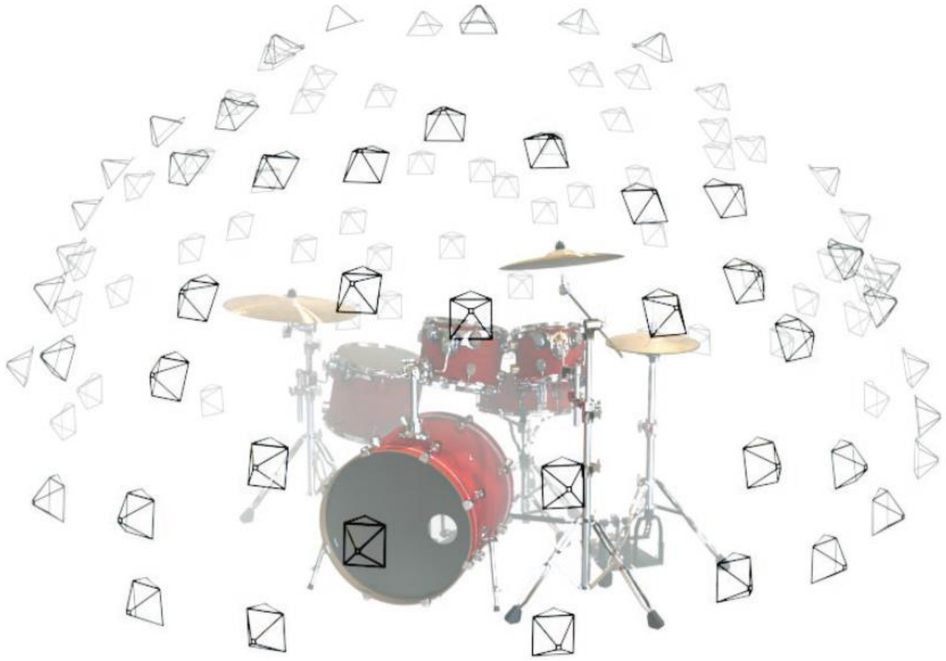
Implicit Function

Neural Radiance Field (NeRF)

Mildenhall*, Srinivasan*, Tancik*, Barron, Ramamoorthi, Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. ECCV 2020

Task: Novel View Synthesis

Input:
A set of calibrated images



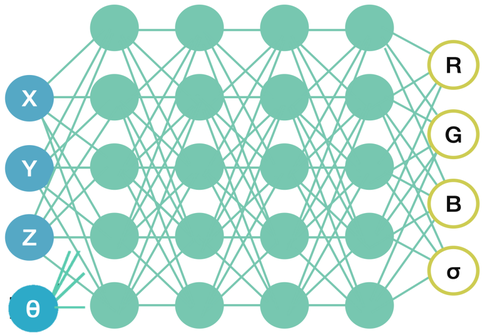
Output:
Novel views of the scene

Overview

Input:
A set of calibrated images

Optimize

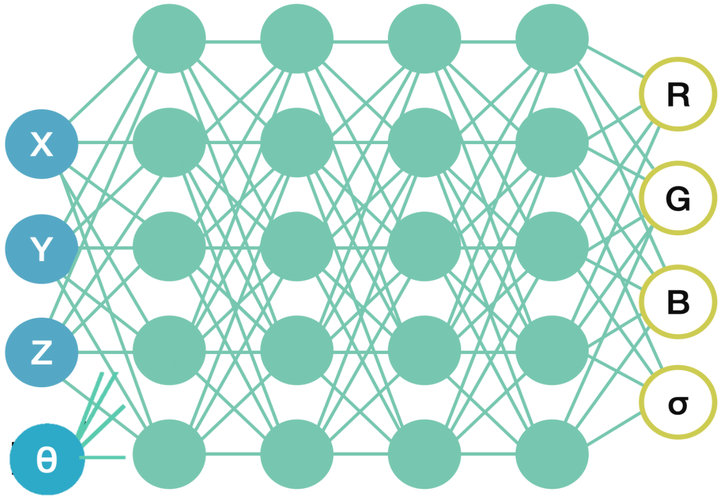
3D Representation:
Neural Radiance Field



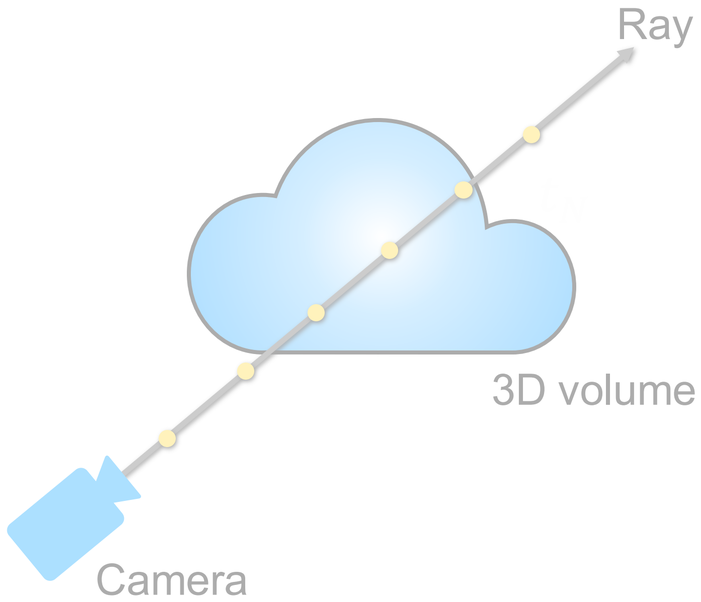
Render

Output:
Novel views of the scene

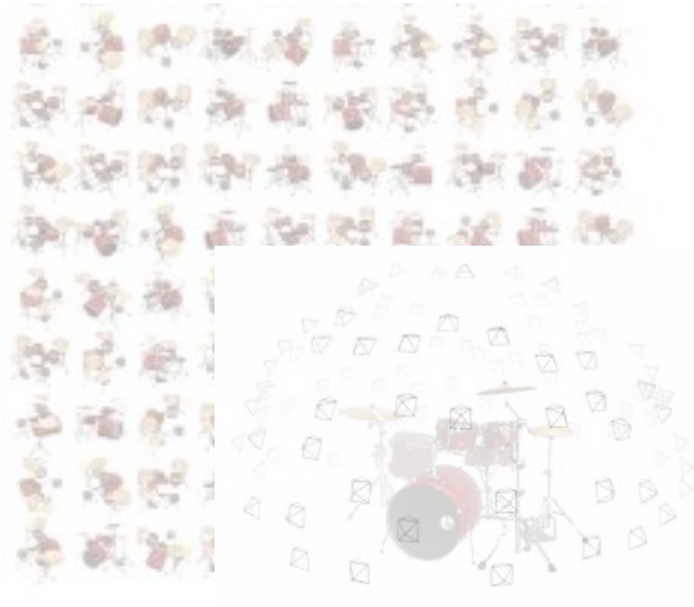
Three Components



Neural Volumetric
3D Scene Representation



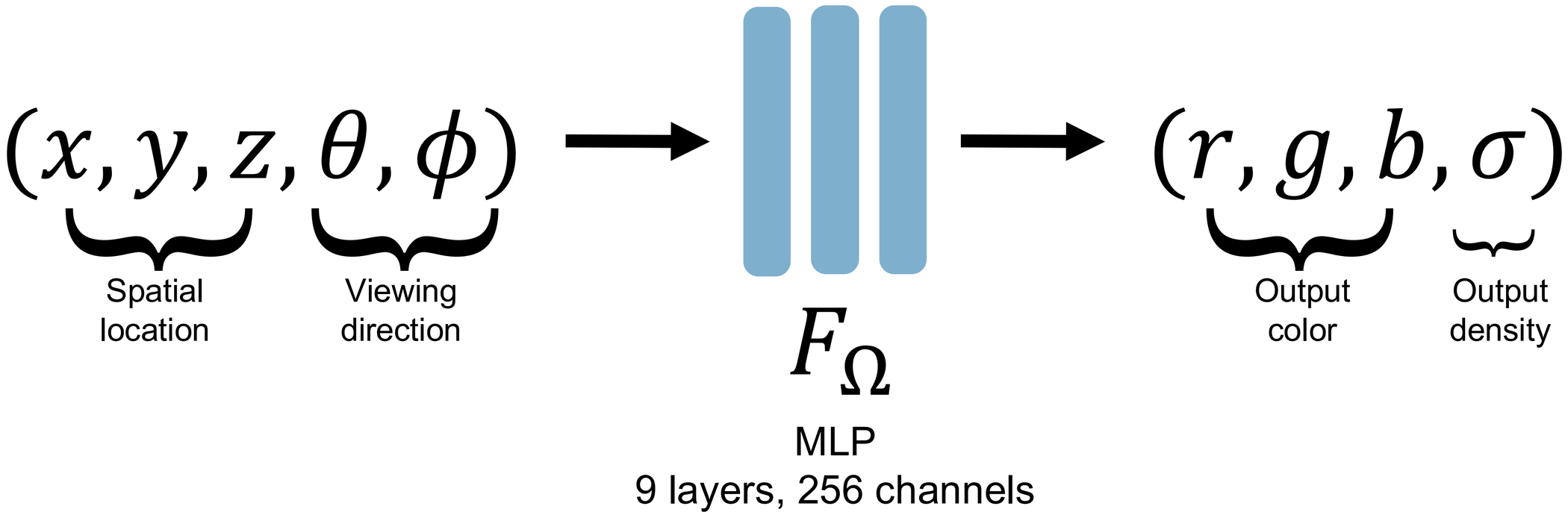
Differentiable
Volume Rendering



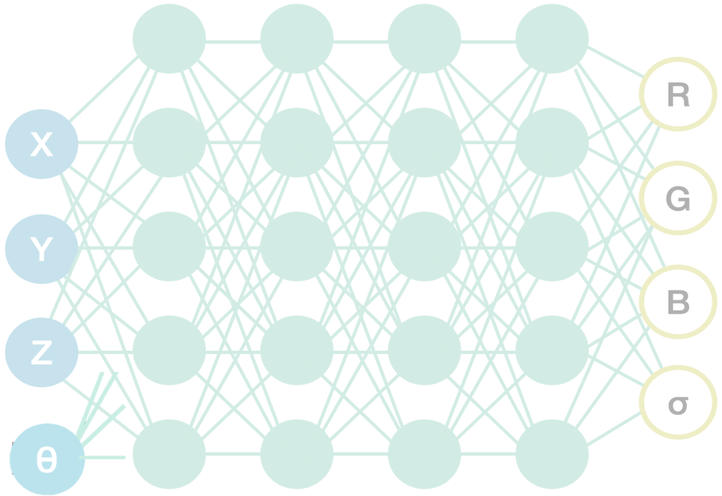
Optimization Techniques

3D Scene Representation (NeRF)

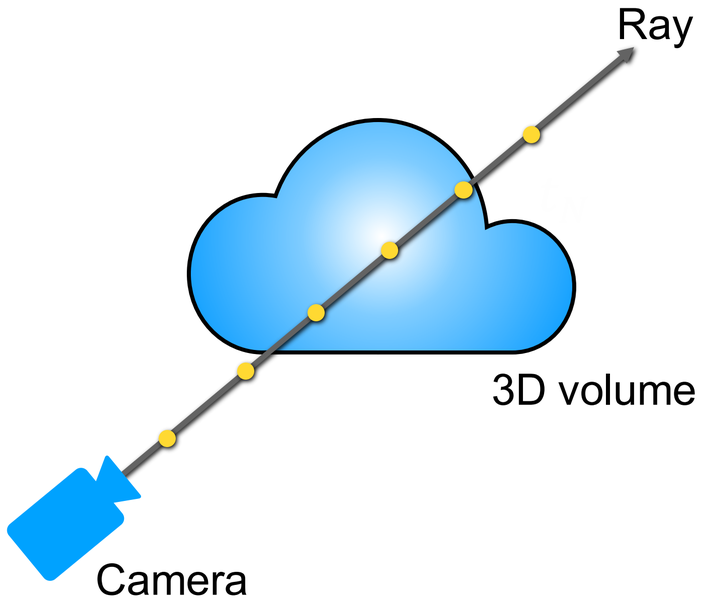
- Representing a 3D scene as a continuous 5D function



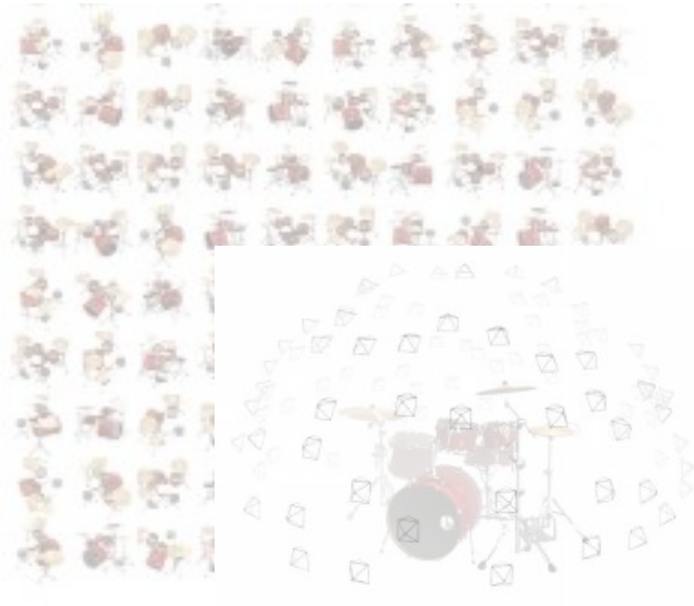
Three Components



Neural Volumetric
3D Scene Representation

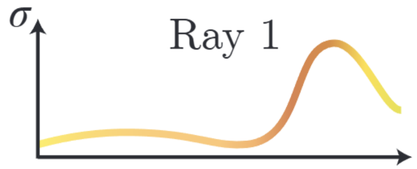


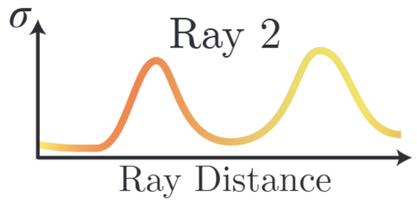
Differentiable
Volume Rendering



Optimization Techniques

Volume Rendering

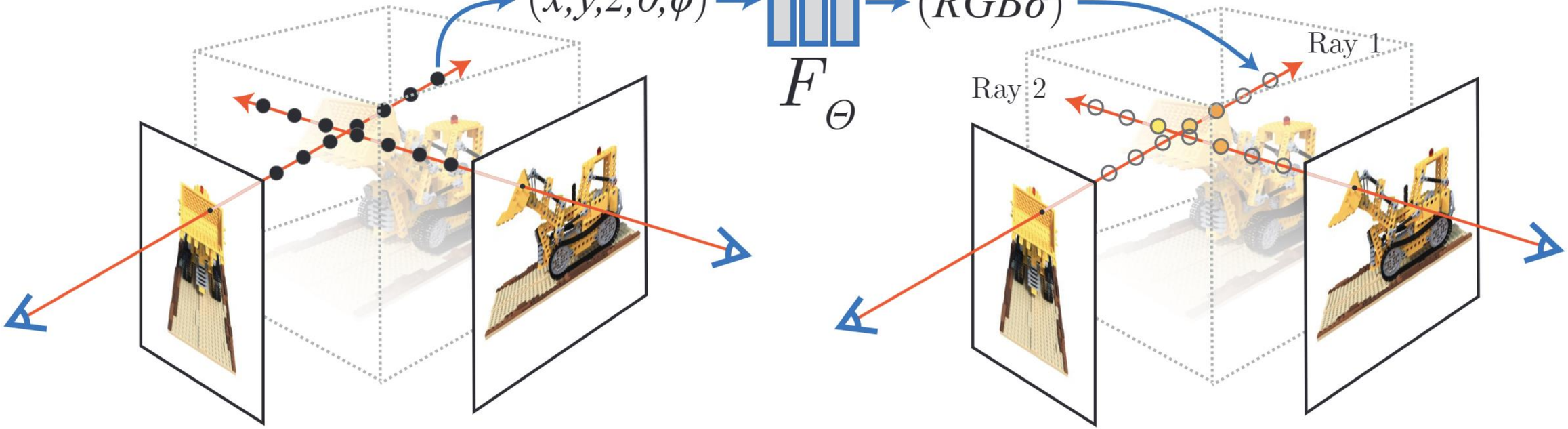




5D Input
Position + Direction

Output
Color + Density

(x,y,z,θ,ϕ) → F_Θ → $(RGB\sigma)$

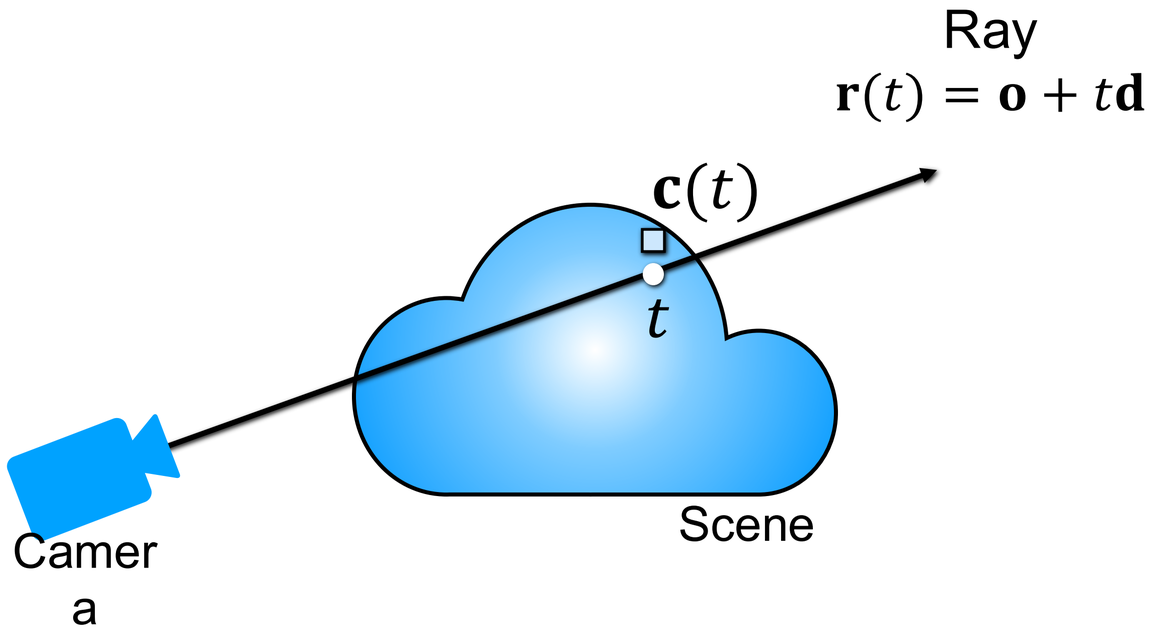


Volume Rendering

- Assume the scene is a cloud of tiny colored particles
- Shoot a ray from the camera view for a pixel using the equation:

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$$

  - $\mathbf{o}, \mathbf{d}$ are the origin and direction
  - t is the distance along the ray
- Each point at distance t has its color $\mathbf{c}(t)$



Volume Rendering

- The probability that the ray hits a particle in the small interval around t is:

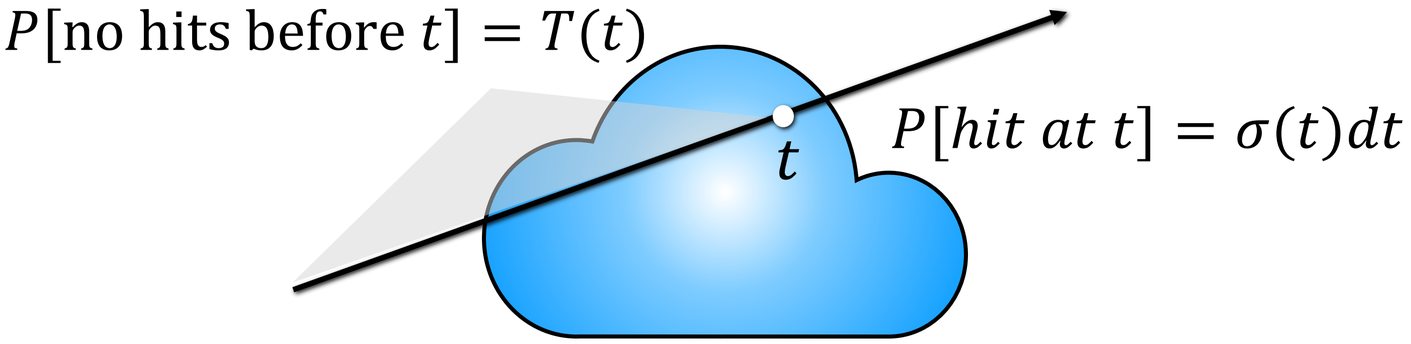
  $$P[hit\ at\ t] = \sigma(t)dt$$

  - σ is called the "**volume density**"

- The probability that the ray doesn't hit particles before t is:

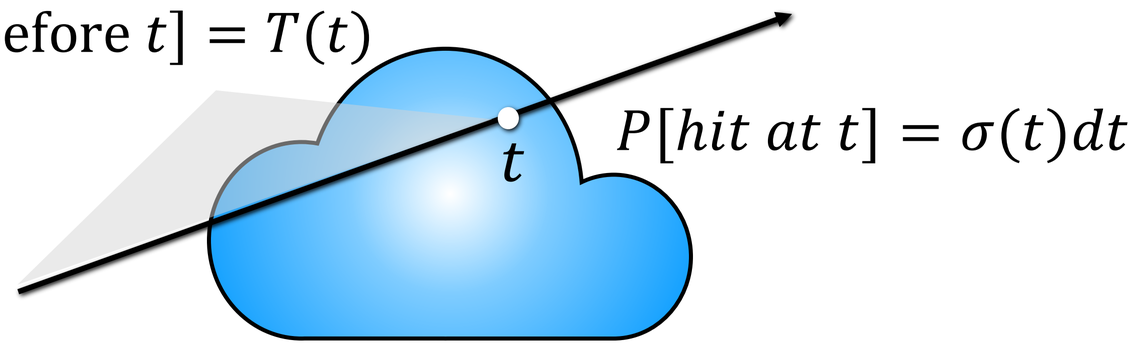
  $$P[\text{no hits before } t] = T(t)$$

  - $T(t)$ is called "**transmittance**"



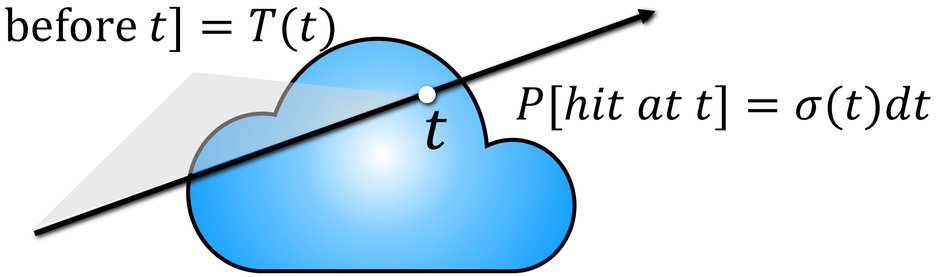
Volume Rendering

- The probability that we see the color of particles at t (the ray first hits at t) is:
  $P[\text{no hit before } t] \times P[hit\ at\ t]$



$$P[\text{first hit at } t] = P[\text{no hit before } t] \times P[hit\ at\ t] = T(t)\sigma(t)dt$$

Calculating T given σ



- σ and T are related by the probabilistic fact that

$$P[\underbrace{\text{no hit before } t + dt}_{T(t+dt)}] = \underbrace{P[\text{no hit before } t]}_{T(t)} \times \underbrace{P[\text{no hit at } t]}_{(1-\sigma(t)dt)}$$

$$T(t + dt) = T(t)(1 - \sigma(t)dt)$$

Taylor expansion for T $\Rightarrow$ $T(t) + T'(t)dt = T(t) - T(t)\sigma(t)dt$

Rearrange $\Rightarrow$ $\frac{T'(t)}{T(t)}dt = -\sigma(t)dt$

Integrate $\Rightarrow$ $\log T(t) = -\int_{t_0}^{t} \sigma(s)ds$

Exponentiate $\Rightarrow$ $T(t) = \exp\left(-\int_{t_0}^{t} \sigma(s)ds\right)$

Volume Rendering

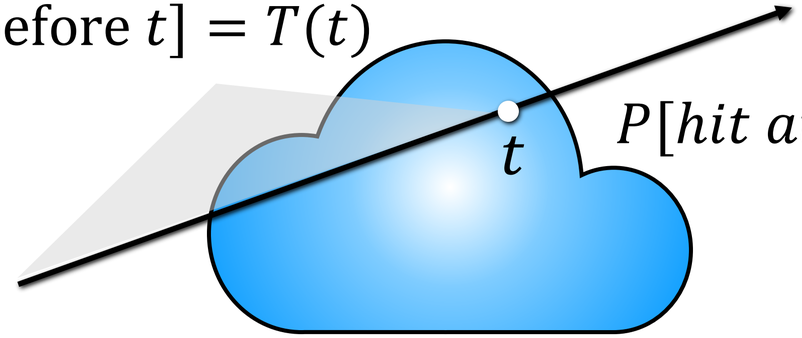
- With $T(t) = \exp\left(-\int_{t_0}^{t} \sigma(s)ds\right)$, we can rewrite the probability that the ray terminates at t as a function of only σ:

$$\exp\left(-\int_{t_0}^{t} \sigma(s)ds\right)\sigma(t)dt$$

- We can get the final color of the pixel/ray:

$$\mathbf{c} = \int T(t)\sigma(t)\mathbf{c}(t)dt$$

$P[\text{no hits before } t] = T(t)$



$P[hit\ at\ t] = \sigma(t)dt$

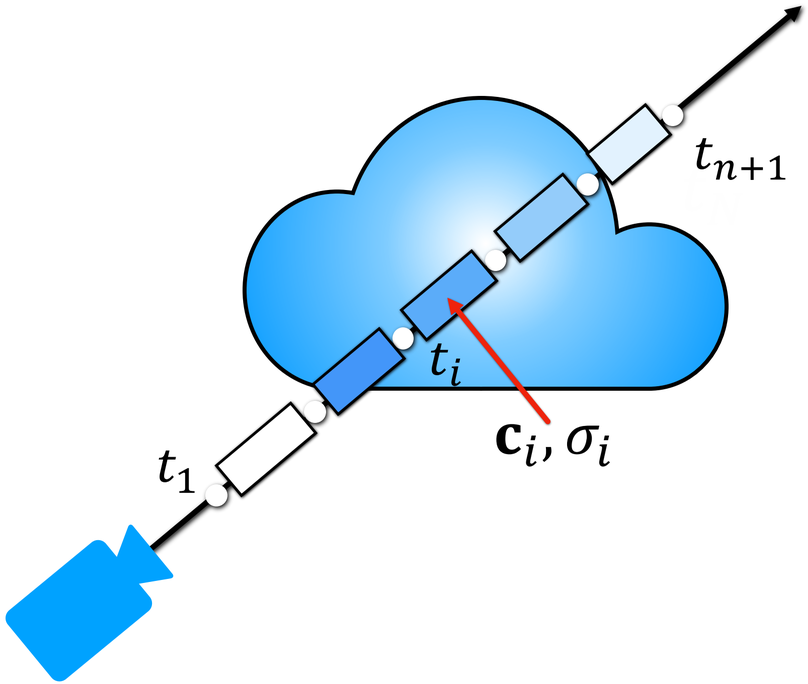
$$P[\text{first hit at } t] = P[\text{no hit before } t] \times P[hit\ at\ t]$$
$$= T(t)\sigma(t)dt$$
$$= \exp\left(-\int_{t_0}^{t} \sigma(s)ds\right)\sigma(t)dt$$

Volume Rendering

- In practice, we use quadrature to approximate the integral
  - Splitting the ray into n segments with endpoints $\{t_1, t_2, \ldots, t_{n+1}\}$, $\delta_i = t_{i+1} - t_i$ denotes segment length
  - We assume density σ and color $\mathbf{c}$ are **constant** within each interval
  - We can get:

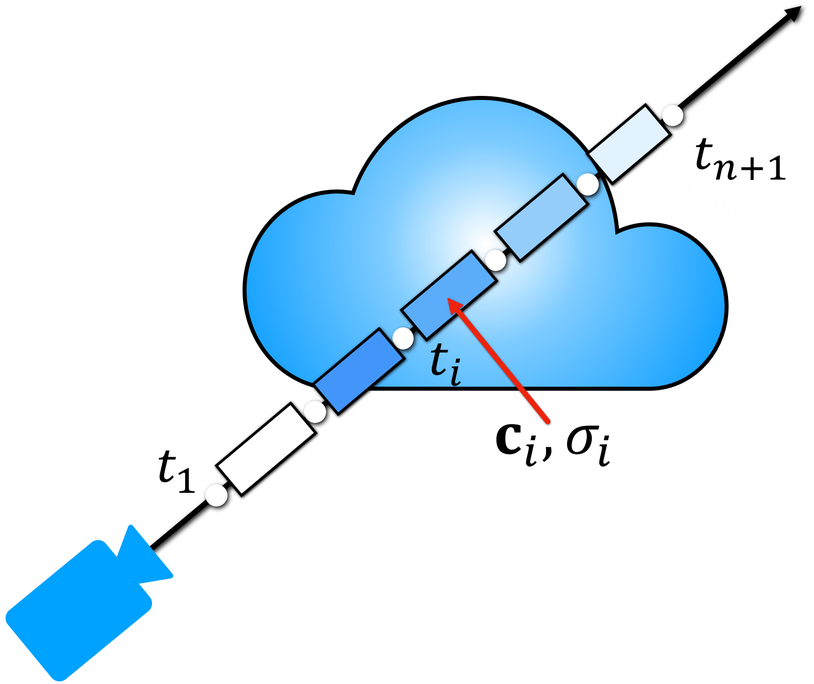
$$\mathbf{c} = \int T(t)\sigma(t)\mathbf{c}(t)dt$$

$$\approx \sum_{i=1}^{n} T_i \mathbf{c}_i (1 - \exp(-\sigma_i \delta_i))$$



Volume Rendering

- In practice, we use quadrature to approximate the integral
  - Splitting the ray into n segments with endpoints $\{t_1, t_2, \ldots, t_{n+1}\}$
  - We assume density σ and color $\mathbf{c}$ are **constant** within each interval
- This allows us to break the integral into a sum of more tractable integrals
  - We still need to handle the integral for $T(t)$



$$\int T(t)\sigma(t)\mathbf{c}(t)dt \approx \sum_{i=1}^{n} \int_{t_i}^{t_{i+1}} T(t)\sigma_i\mathbf{c}_i dt$$

Final pixel color

Volume Rendering

Final pixel color

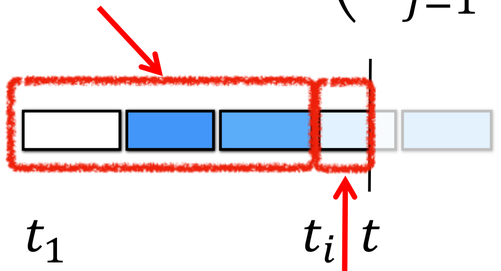
$$\int T(t)\sigma(t)\mathbf{c}(t)dt \approx \sum_{i=1}^{n} \int_{t_i}^{t_{i+1}} T(t)\sigma_i \mathbf{c}_i dt$$

- Split $T(t)$ into two parts:

$$T(t) = T_i \exp\left(-\int_{t_i}^{t} \sigma_i ds\right)$$

The probability that the ray doesn't hit particles before t_i

$$T_i = \exp\left(-\int_{t_1}^{t_i} \sigma_i ds\right) = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$$



$$\exp\left(-\int_{t_i}^{t} \sigma_i ds\right) = \exp(-\sigma_i(t - t_i))$$

The probability that the ray doesn't hit particles between t_i and t

Volume Rendering

- Split $T(t)$ into two parts:
  $$T(t) = T_i \exp\left(-\int_{t_i}^{t} \sigma_i ds\right)$$
- With $T(t) = T_i \exp(-\sigma_i(t - t_i))$, we can approximate the pixel color $\mathbf{c}$:
  $$\mathbf{c} \approx \sum_{i=1}^{n} T_i \mathbf{c}_i (1 - \exp(-\sigma_i \delta_i))$$

Final pixel color

$$\int T(t)\sigma(t)\mathbf{c}(t)dt \approx \sum_{i=1}^{n} \int_{t_i}^{t_{i+1}} T(t)\sigma_i \mathbf{c}_i dt$$

Substitute ⇒
$$= \sum_{i=1}^{n} T_i \sigma_i \mathbf{c}_i \int_{t_i}^{t_{i+1}} \exp(-\sigma_i(t - t_i))dt$$

Integrate ⇒
$$= \sum_{i=1}^{n} T_i \sigma_i \mathbf{c}_i \frac{\exp(-\sigma_i(t_{i+1} - t_i)) - 1}{-\sigma_i}$$

Cancel σ_i ⇒
$$= \sum_{i=1}^{n} T_i \mathbf{c}_i (1 - \exp(-\sigma_i \delta_i))$$

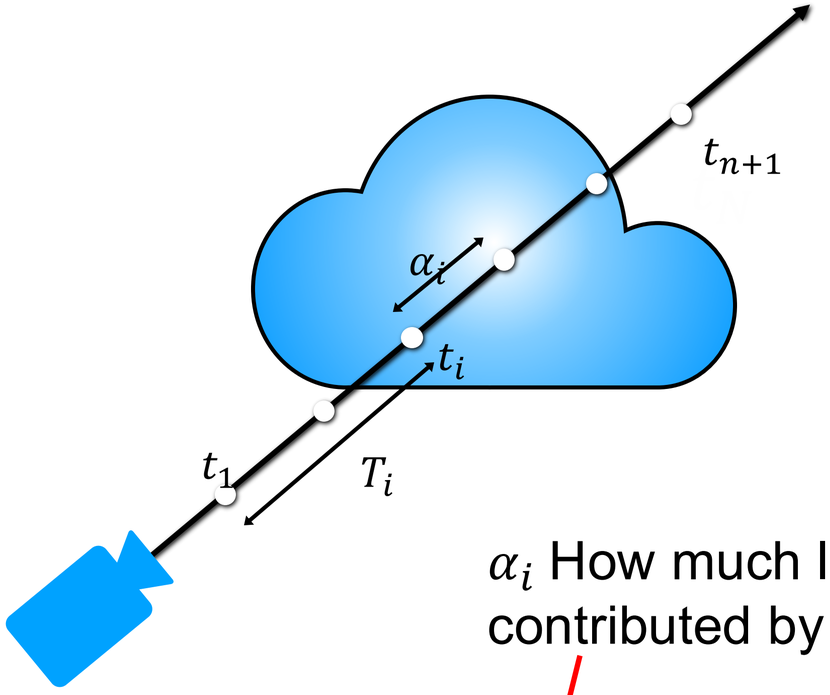
c

Volume Rendering

- The pixel color $\mathbf{c}$:

$$\mathbf{c} \approx \sum_{i=1}^{n} T_i \mathbf{c}_i (1 - \exp(-\sigma_i \delta_i))$$

- Introduce segment opacity α_i:
  - $\alpha_i = 1 - \exp(-\sigma_i \delta_i)$
  - $T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$



α_i How much light is contributed by ray segment i

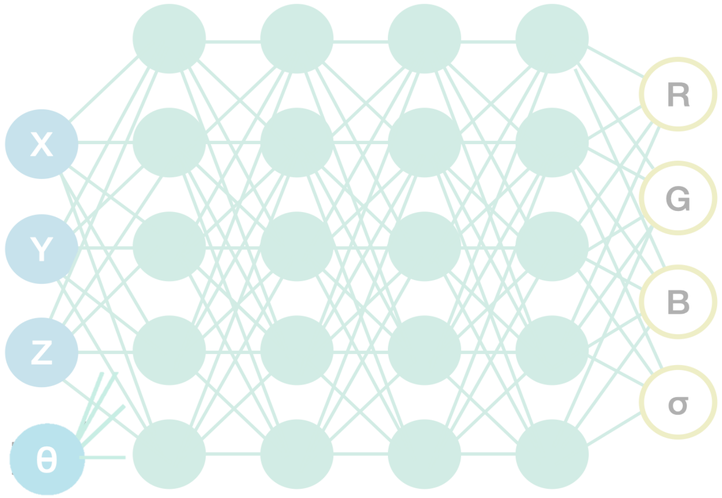
$$\mathbf{c} \approx \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i$$

$\mathbf{c}_i$ Color

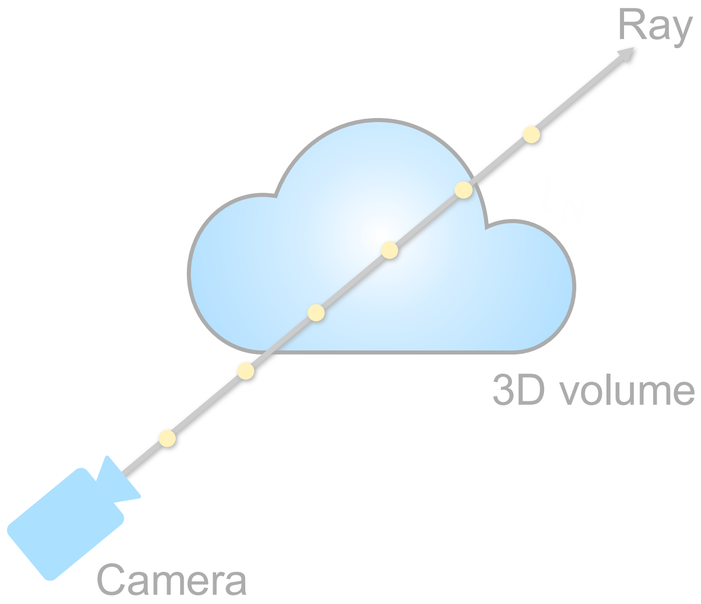
T_i How much light can go through the space before t_i

$T_i \alpha_i$ is “rendering weights”

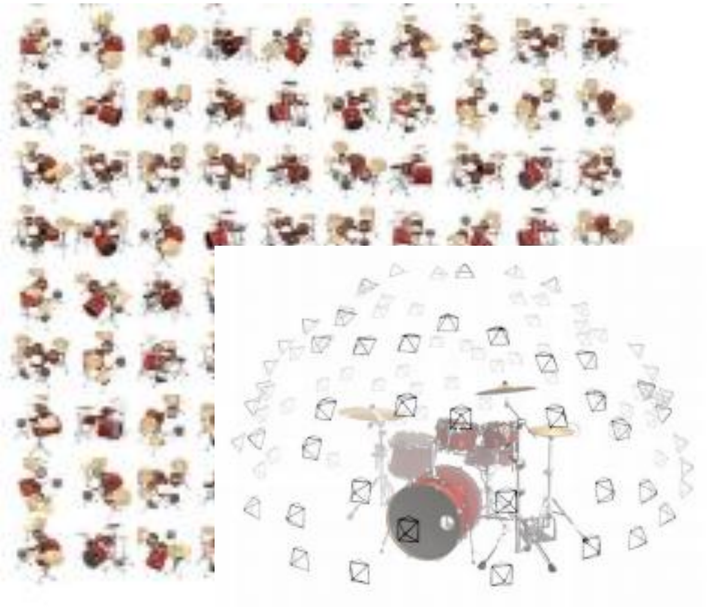
Three Components



Neural Volumetric
3D Scene Representation

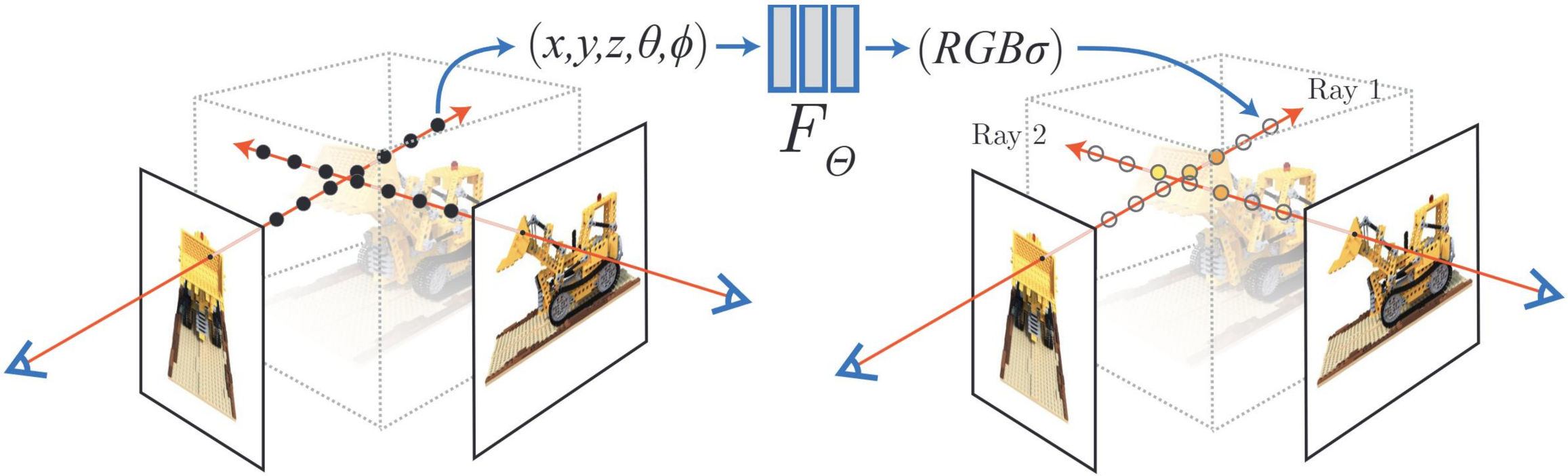


Differentiable
Volume Rendering



Optimization Techniques

Optimize with Rendering Loss



$$\min_{\theta} \sum_{i} \| \text{render}_i(F_\theta) - I_i \|^2$$

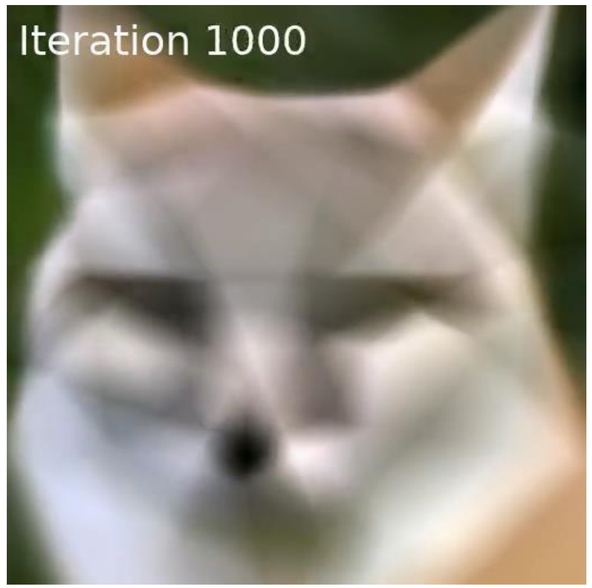
Position Encoding

- $xyz\theta\phi$ input coordinates perform poorly on high-frequency signals

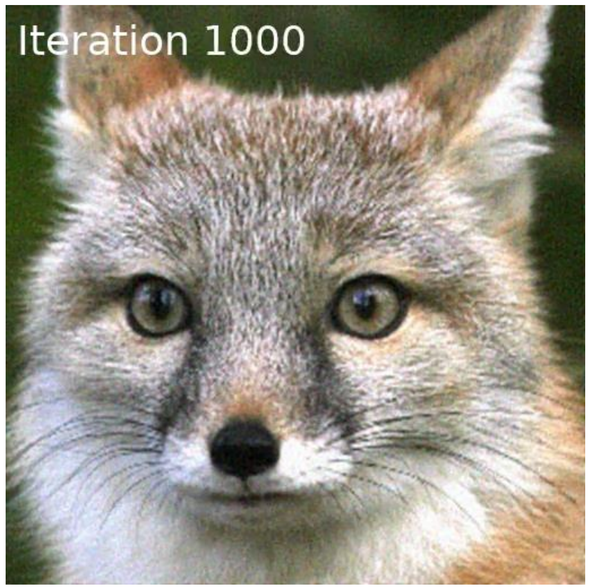
$$\boldsymbol{x} \rightarrow$$ 

- Position encoding helps

$$\begin{pmatrix} \sin(\boldsymbol{x}), \cos(\boldsymbol{x}) \\ \sin(2\boldsymbol{x}), \cos(2\boldsymbol{x}) \\ \sin(4\boldsymbol{x}), \cos(4\boldsymbol{x}) \\ \dots \\ \sin(2^{L-1}\boldsymbol{x}), \cos(2^{L-1}\boldsymbol{x}) \end{pmatrix} \rightarrow$$ 



Standard MLPS



with Position Encoding

Position Encoding

NeRF (Naive)

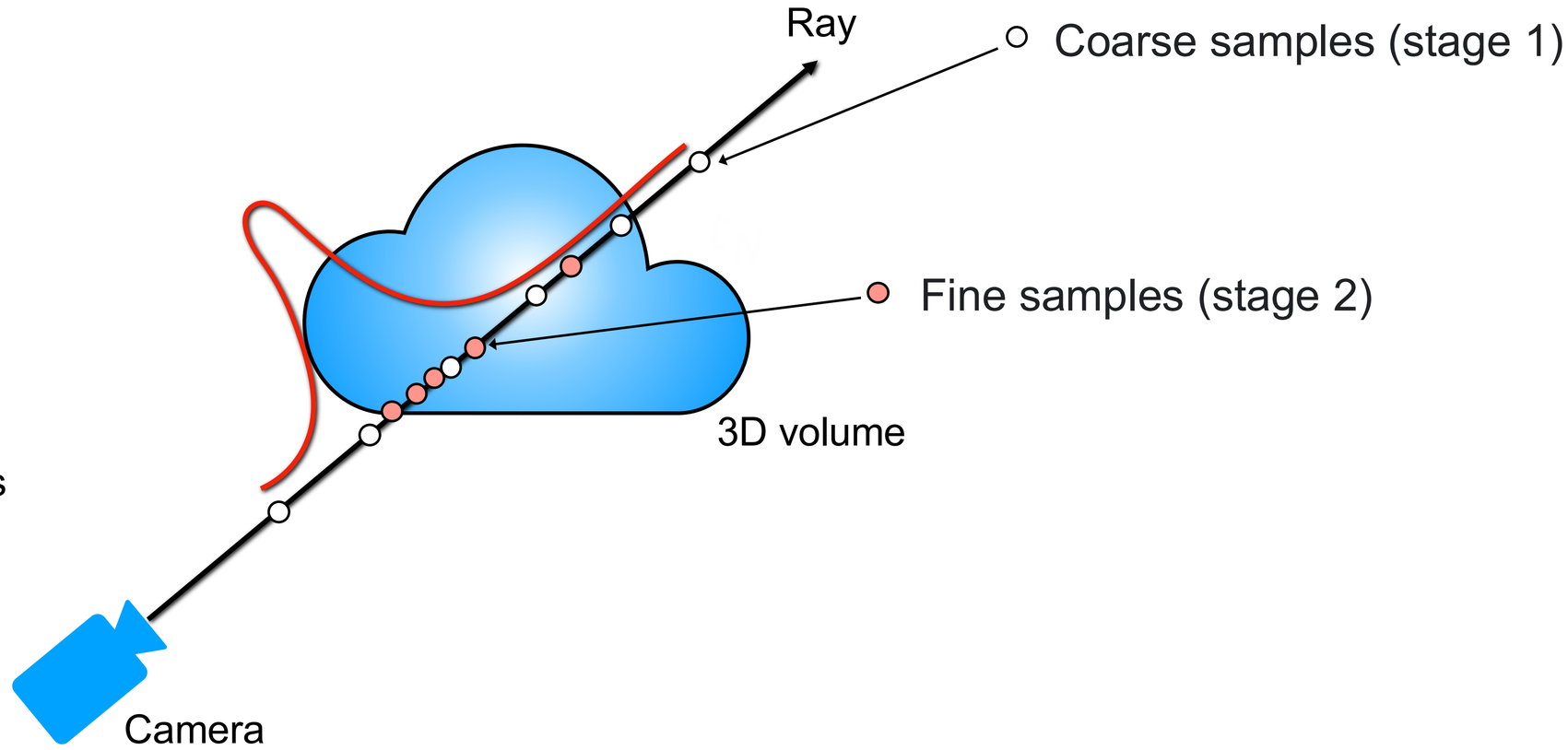
NeRF (with Position Encoding)

Hierarchical Ray Sampling

- Sampling more points in the high-density area

$$C \approx \sum_{i=1}^{N} T_i \alpha_i c_i$$

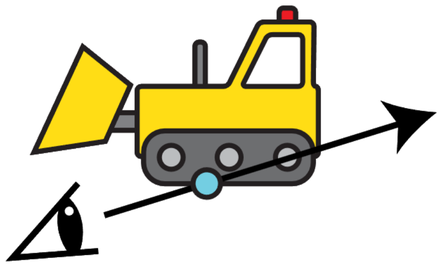
treat weights as probability distribution for new samples



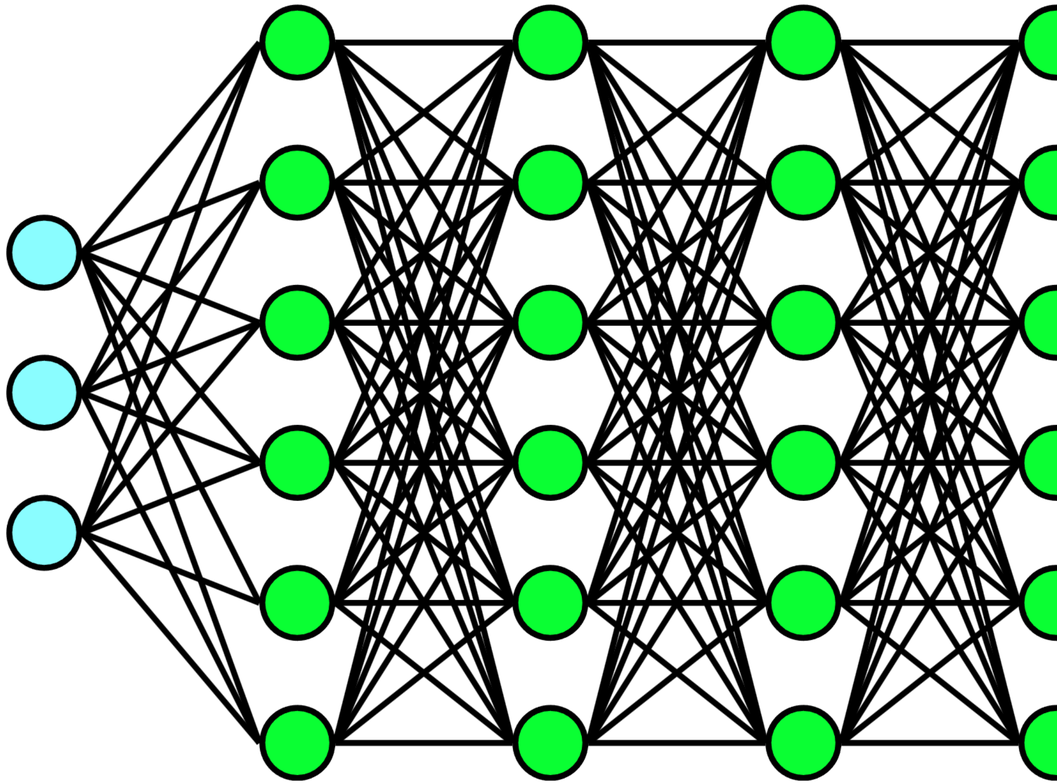
NeRF Follow-ups

Acceleration Techniques

- MLP is slow to optimize



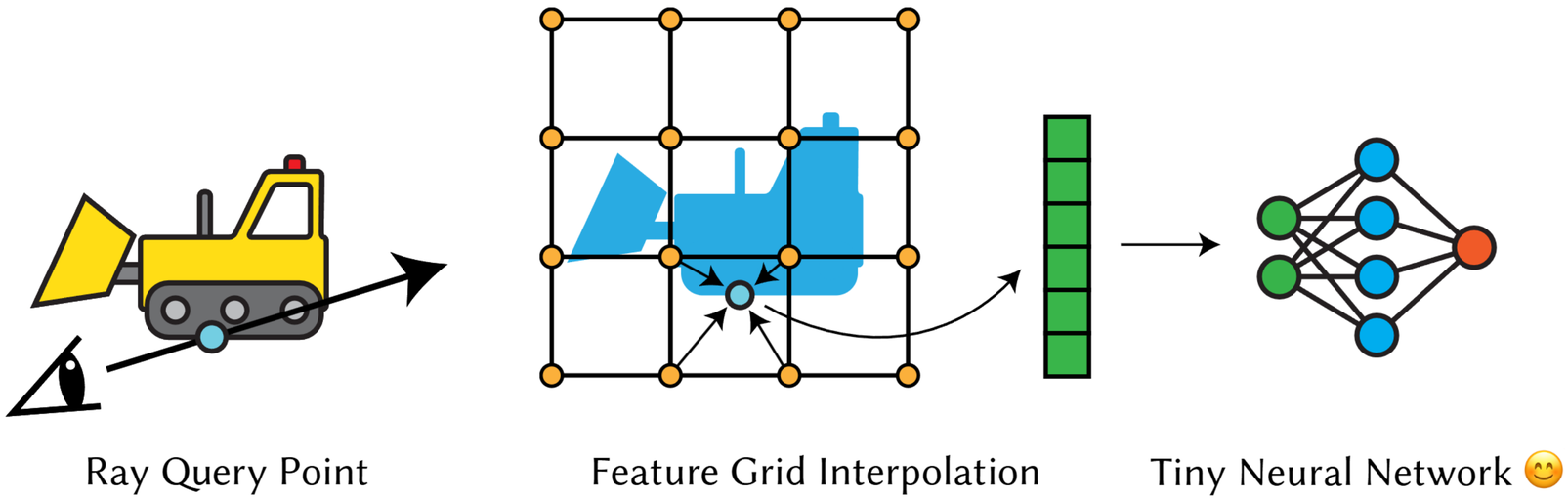
Ray Query Point



NeRF Follow-ups

Acceleration Techniques

- MLP is slow to optimize
- Hybrid representations for the acceleration:
  - Feature grid

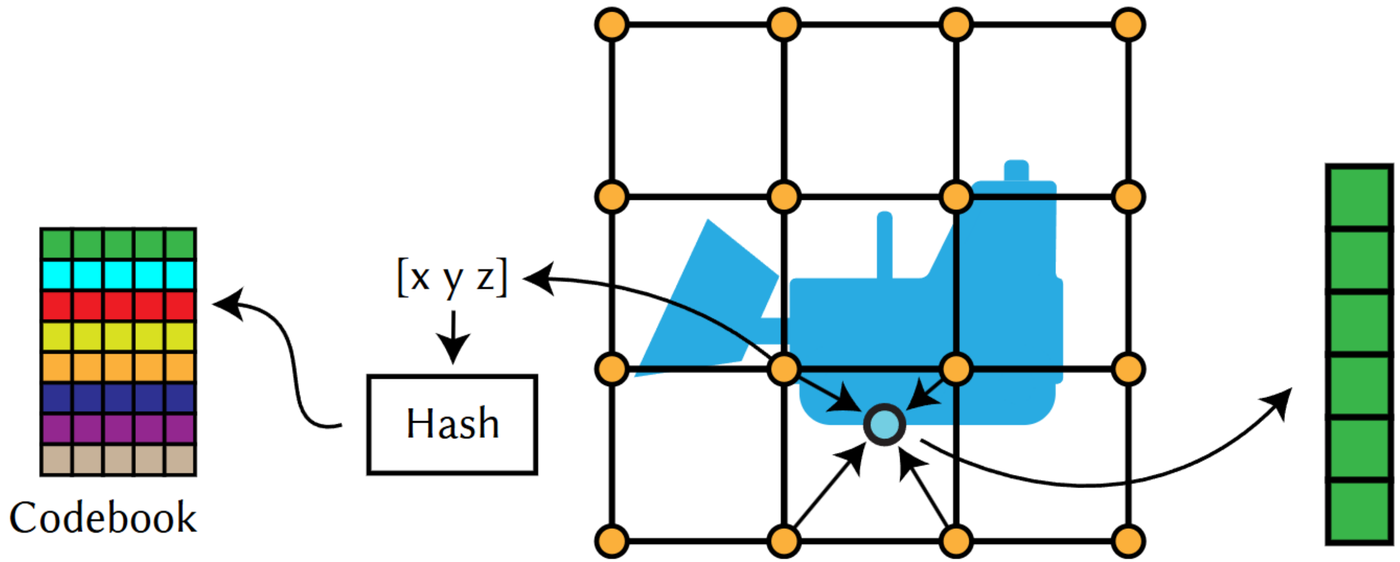


Fridovich-Keil, Yu, Tancik, Chen, Recht, & Kanazawa. Plenoxels: Radiance fields without neural networks. CVPR 2022.
Tancik, Casser, Yan, Pradhan, Mildenhall, Srinivasan, ... & Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. CVPR 2022.
Zhang, Bi, Sunkavalli, Su, & Xu. Nerfusion: Fusing radiance fields for large-scale scene reconstruction. CVPR 2022.

NeRF Follow-ups

Acceleration Techniques

- MLP is slow to optimize
- Hybrid representations for the acceleration:
  - Feature grid
  - Codebook



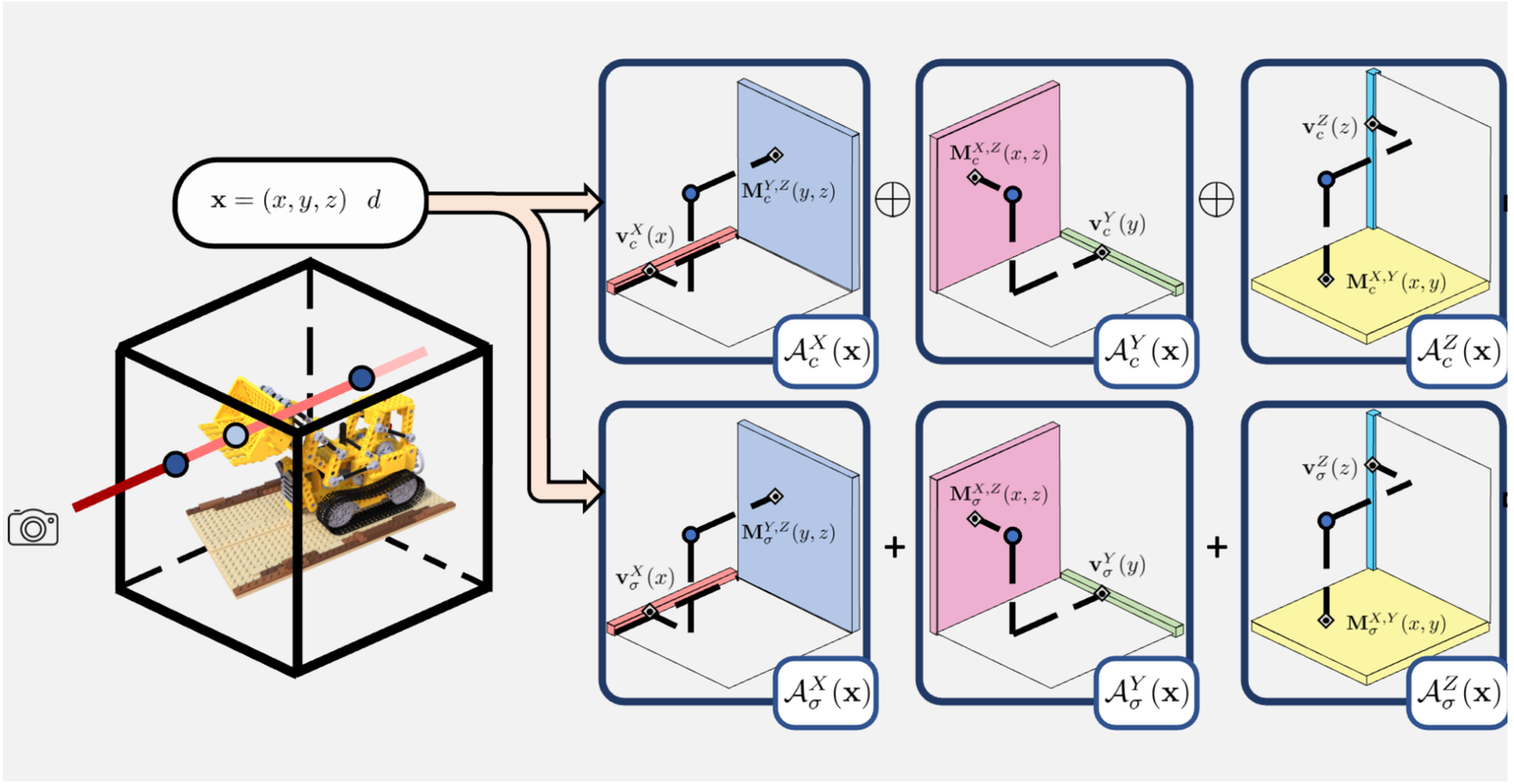
[Instant-NGP (Muller et al.)]

Müller, Evans, Schied, & Keller. Instant neural graphics primitives with a multiresolution hash encoding. TOG 2022.

NeRF Follow-ups

Acceleration Techniques

- MLP is slow to optimize
- Hybrid representations for the acceleration:
  - Feature grid
  - Codebook
  - Vector-matrix factorization
  - ……



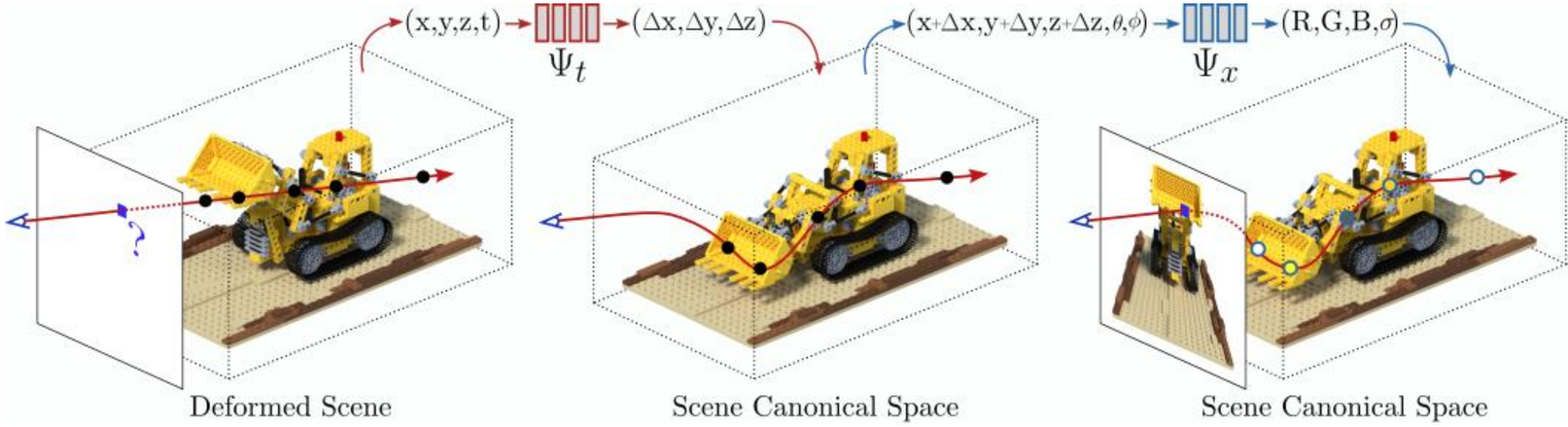
TensoRF

Chen, Xu, Geiger, Yu, & Su. TensoRF: Tensorial Radiance Fields. ECCV 2022.

NeRF Follow-ups

Dynamic Scene

- Vanilla NeRF is designed for the static scene
- D-NeRF utilizes a deformation network to align different frames



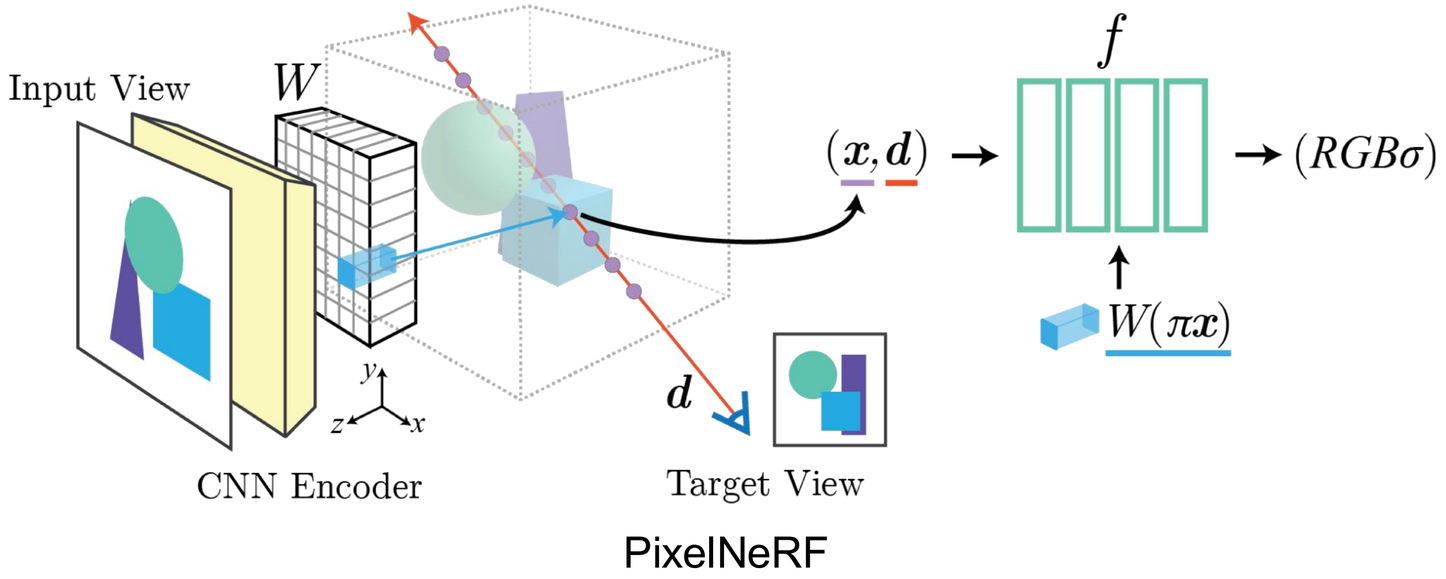
D-NeRF

Pumarola, Corona, Pons-Moll, & Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. CVPR 2021.

NeRF Follow-ups

Generalization

- Vanilla NeRF is optimized for a single scene with dense views (~100)
- Generalizable NeRFs can predict a NeRF volume with single/few-shot views
  - Pretraining on large-scale datasets



PixelNeRF

Yu, Ye., Tancik, & Kanazawa. pixelnerf: Neural radiance fields from one or few images. CVPR 2021.

NeRF Follow-ups

Generation

- Combine NeRF and 2D generative models

DreamFusion

Poole, Jain, Barron, & Mildenhall. (2022). Dreamfusion: Text-to-3d using 2d diffusion. *ICLR 2023.*

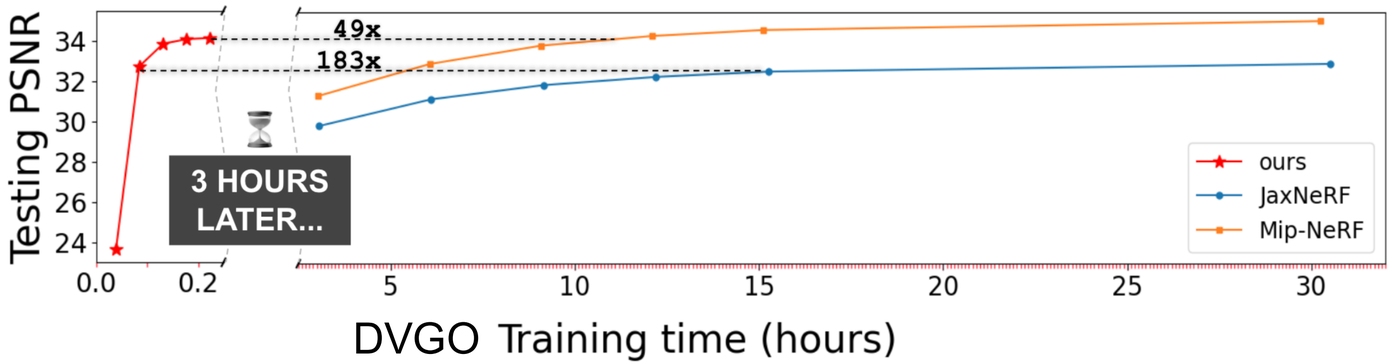
3D Gaussian Splatting

for Real-Time Radiance Field Rendering

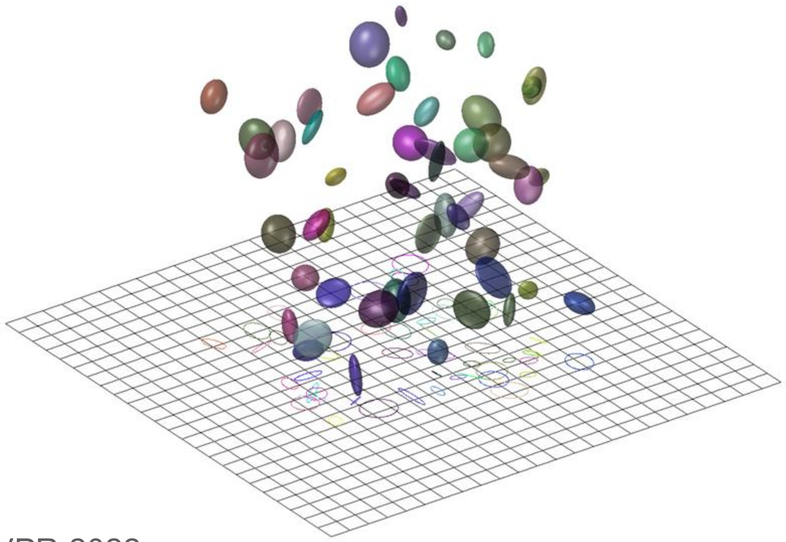
Kerbl, Kopanas, Leimkühler, & Drettakis. 3d gaussian splatting for real-time radiance field rendering. TOG 2023.

3D Gaussian Splatting

- NeRFs suffer from slow training and rendering



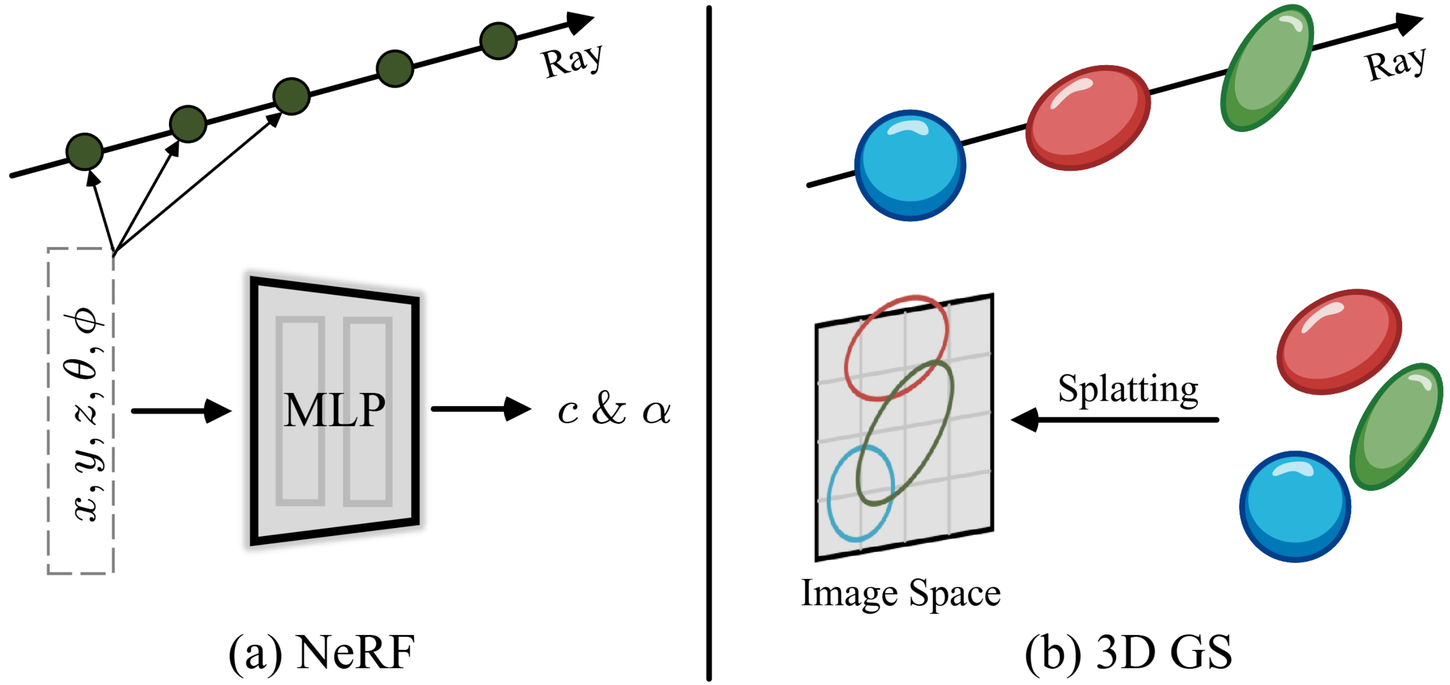
- Gaussian Splatting employs **Gaussian splats** as primitives to **efficiently** represent and render 3D scenes
  - Rendering with efficiency
  - Rendering with high-quality
  - Gaussian splats as primitives



Sun, Sun, & Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. CVPR 2022.

3D Gaussian Splatting

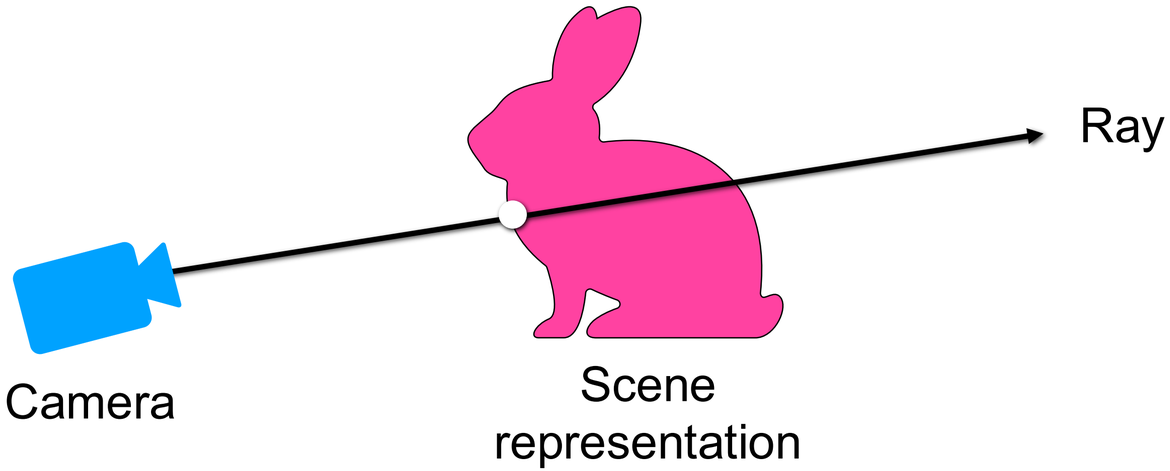
- Positions - μ
- Covariance - Σ
- Color - c
- Opacity - α



Sun, Sun, & Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. CVPR 2022.

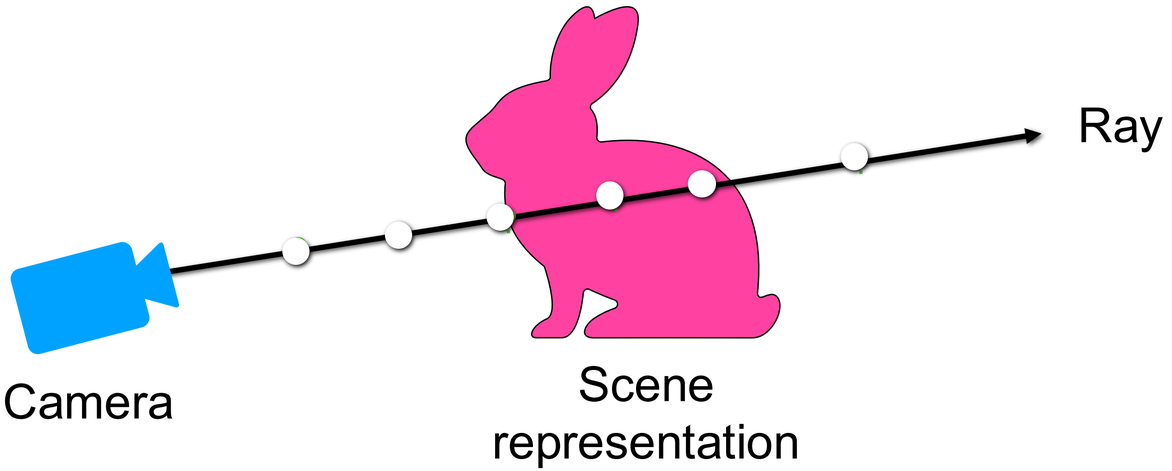
3D Gaussian Splatting

Surface Rendering VS Volume Rendering



Surface Rendering

We know where is the surface



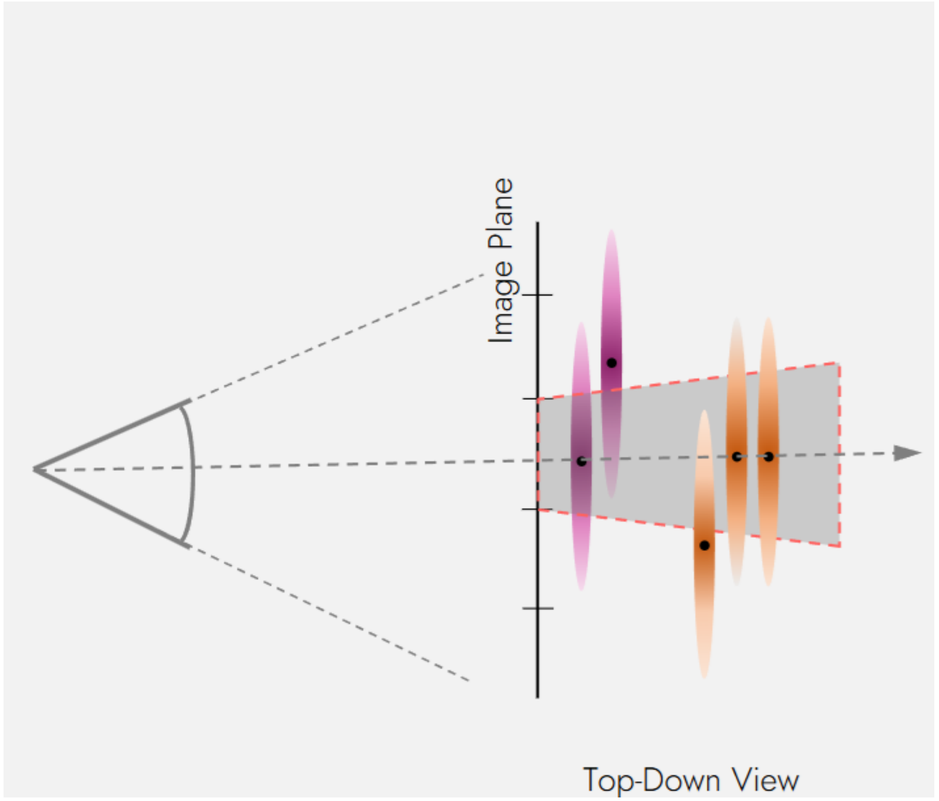
Volume Rendering

We don’t know where is the surface

3D Gaussian Splatting

Rendering

1. **Sort**: globally based on depth
2. **Splat**: compute the shape of the Gaussian after projection
3. **Blend**: alpha composite



3D Gaussian Splatting

Rendering

- **Blending**: μ_i - position, Σ_i - covariance, c_i - color, o_i - opacity

$$\alpha_i(p) = sigm(o_i)\exp(-\frac{1}{2}(p-\mu_i)^T\Sigma_i^{-1}(p-\mu_i))$$

- **Splatting**: project 3D gaussians into gaussians, K,W – camera intrinsic/extrinsic, J_i- Jacobian matrix of projection matrix ($\partial\mu_i^{2D}/\partial\mu_i$)

$$\mu_i^{2D} = \mathrm{K}(\mathrm{W}\mu_i/(\mathrm{W}\mu_i)_z) \qquad \Sigma_i^{2D} = J_i\mathrm{W}\Sigma\mathrm{W}^T J_i^T$$

- **Rendering**:

$$c_i^{2D} = \sum_{i\in N} c_i\alpha_i^{2D}\prod_{j=1}^{i-1}(1-\alpha_i^{2D})$$

Volumetric Rendering in NeRFs

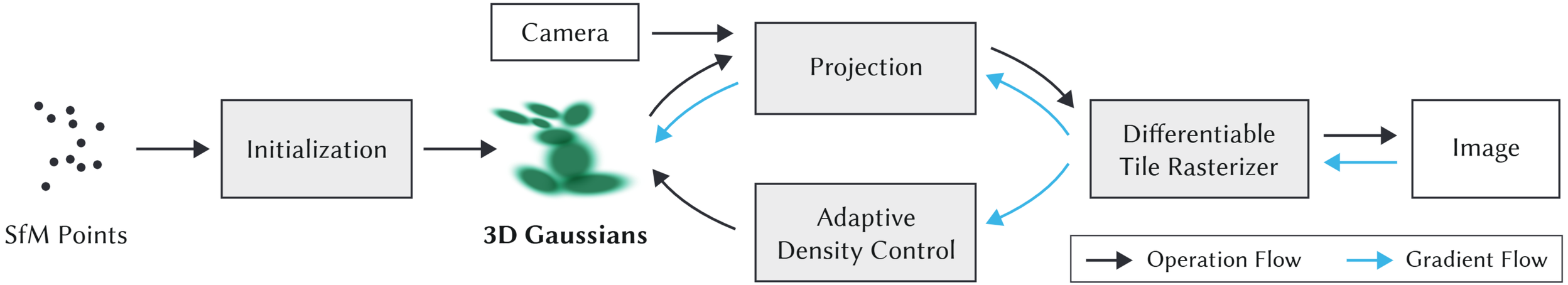
- With $T(t) = T_i\exp(-\sigma_i(t-t_i))$, we can approximate the pixel color $\mathbf{c}$:

$$\mathbf{c} \approx \sum_{i=1}^{n} T_i\mathbf{c}_i(1-\exp(-\sigma_i\delta_i))$$

- Introduce segment opacity α_i:
  - $\alpha_i = 1-\exp(-\sigma_i\delta_i)$
  - $T_i = \prod_{j=1}^{i-1}(1-\alpha_j)$

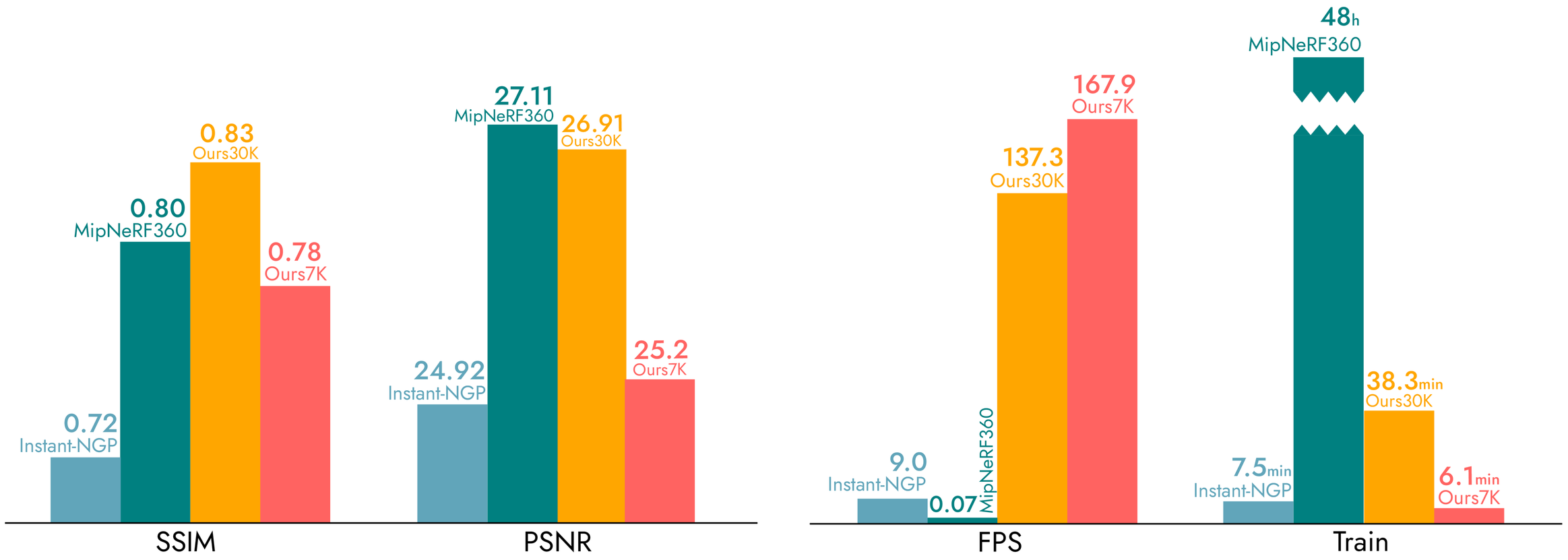
3D Gaussian Splatting

Pipeline



3D Gaussian Splatting

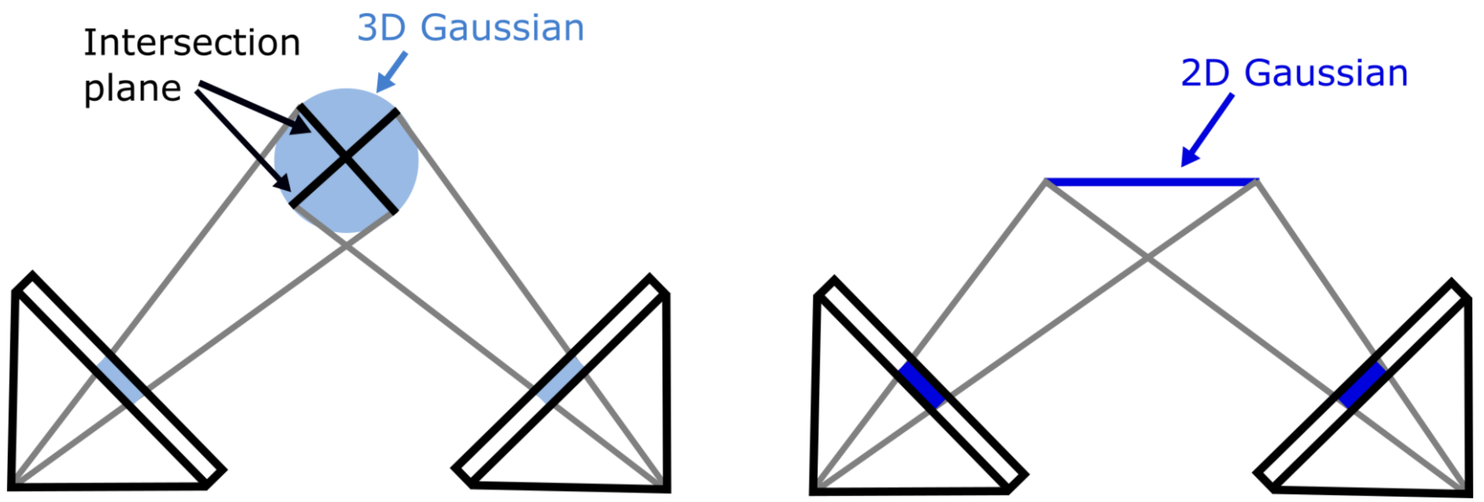
Evaluation



Gaussian Splatting Follow-ups

2D Gaussian Splatting

- Collapse the 3D volume into a set of 2D oriented planar Gaussian ***disks***

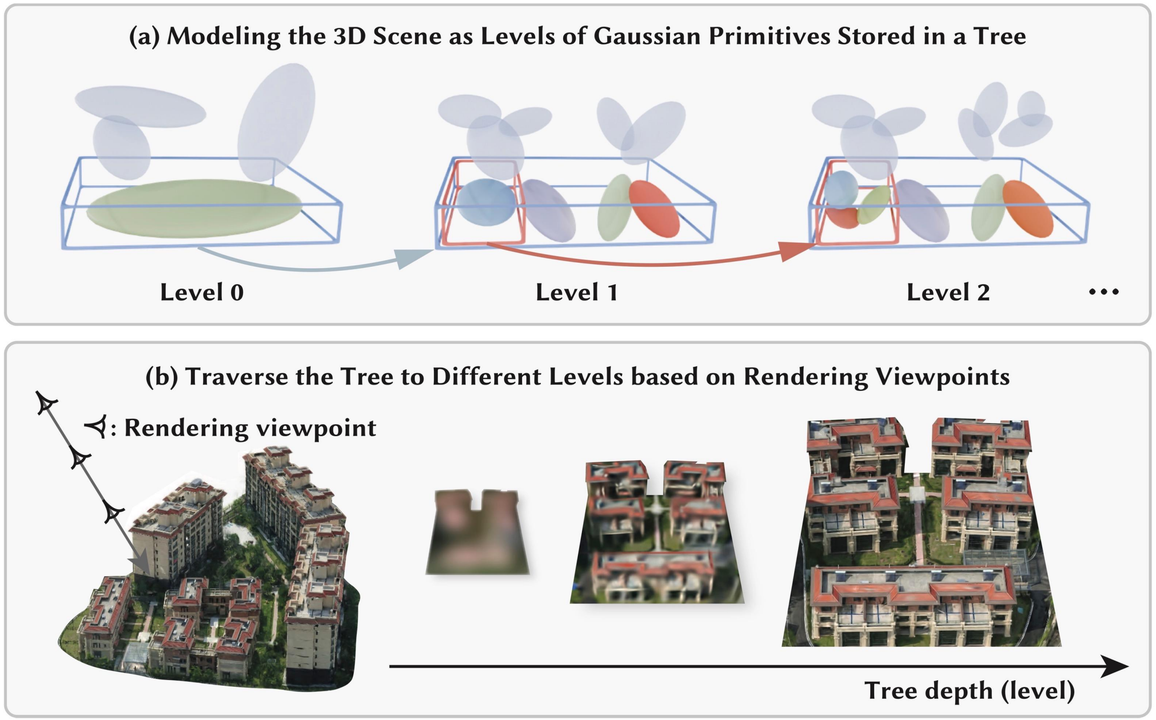


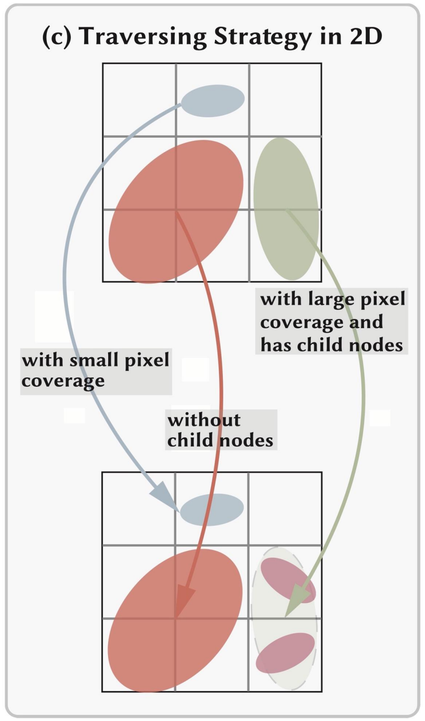
Huang et al., 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. SIGGRAPH 2024.

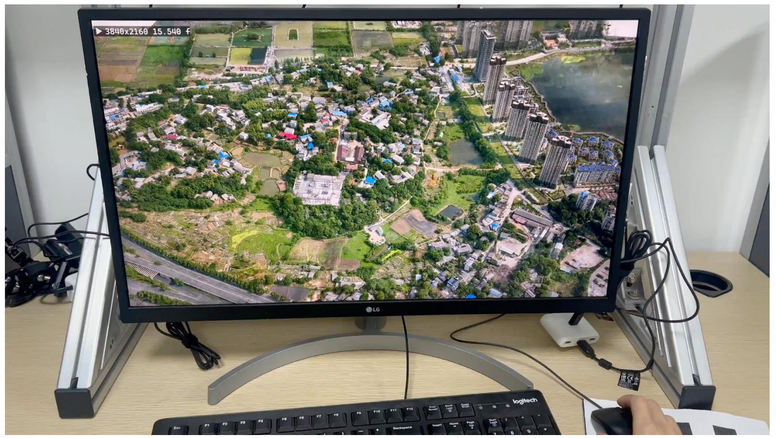
Gaussian Splatting Follow-ups

Large-scale

- Level of Gaussians



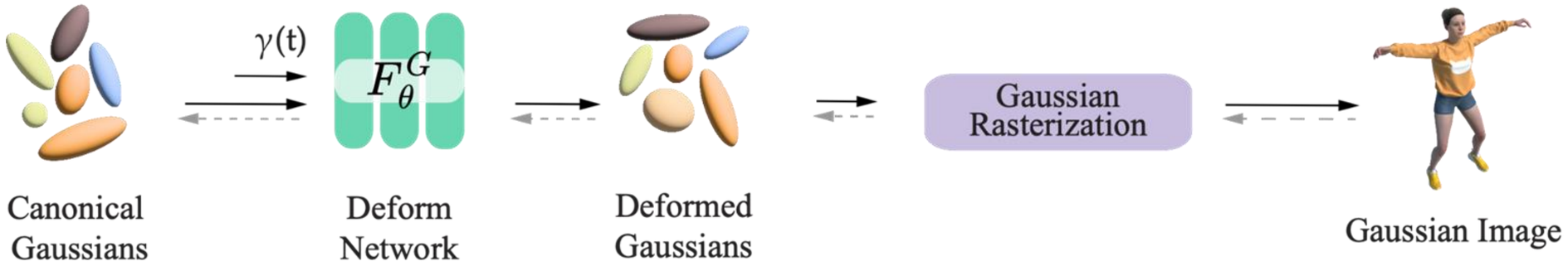




Gaussian Splatting Follow-ups

Dynamic Scene

- DG-mesh deform gaussians to align different frames → consistent mesh

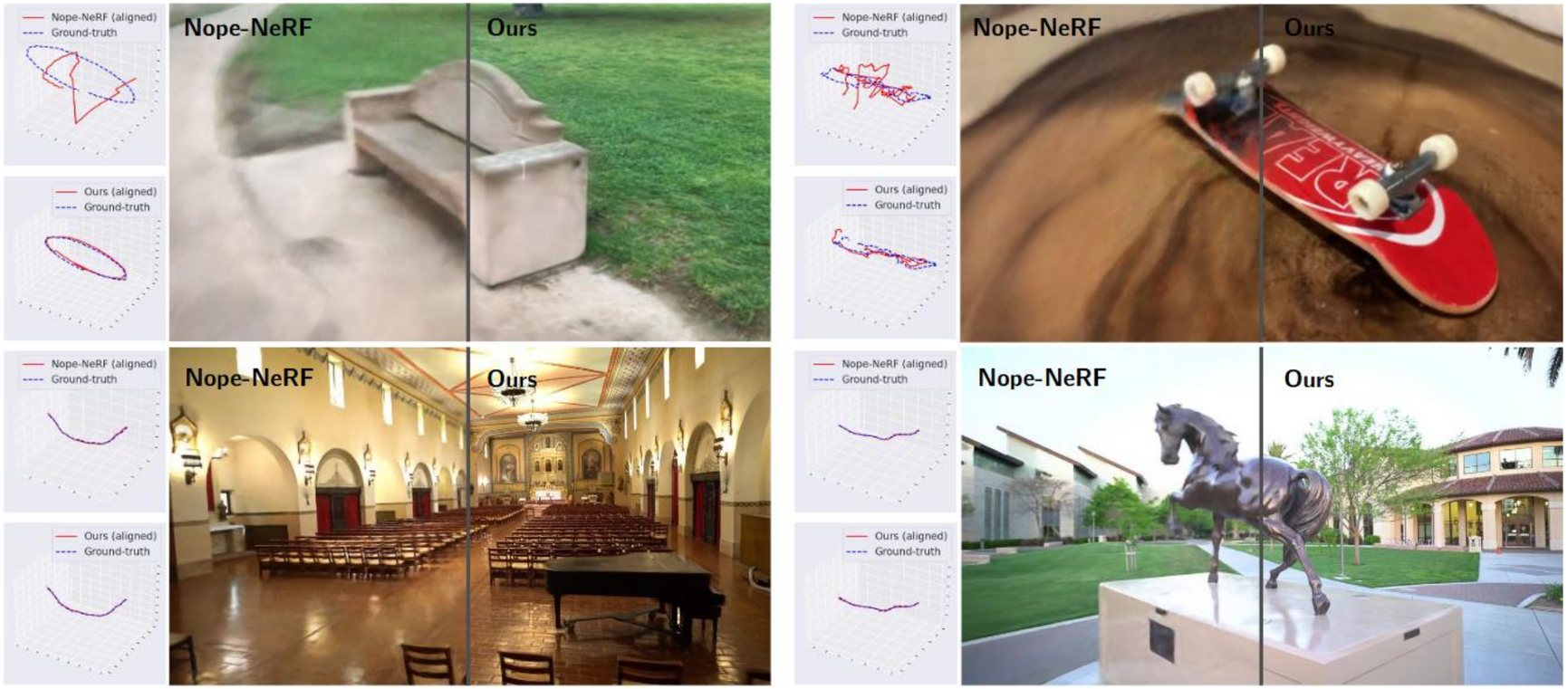


Liu, Su, Wang. Dynamic Gaussian Mesh: Consistent Mesh Reconstruction from Monocular Videos. ICLR 2025.

Gaussian Splatting Follow-ups

Pose-Free

- Colmap-free GS



Fu, Liu, Kulkarni, Kautz, Efros, & Wang. Colmap-free 3d gaussian splatting. CVPR 2024.

Gaussian Splatting Follow-ups

Feature Fields

- Optimize for features from large-scale 2D vision models

Zou, Song, Qiu, Peng, Ye, Liu & Wang. M3: 3D-Spatial MultiModel Memory. ICLR 2025.
Ji, Qiu, Zou & Wang. GraspSplats: Efficient Manipulation with 3D Feature Splatting. CoRL 2024.