ADVANCES IN COMPUTER VISION

MODULE 1: Geometry

RENDERING
Inverse Graphics & Differentiable Rendering



Prof. Vincent Sitzmann

Rendering (Graphics): Given 3D Scene + Camera parameters, yield images



3D Scene



Camera Poses

Render



Images

Inverse Graphics: Given Images, Infer Camera Poses & 3D Scene!



Images

Reconstruct



3D Scene



Camera Poses

How to get camera poses?



Images

Reconstruct

3D Scene



Camera Poses

Can assume we know the camera poses.



Images



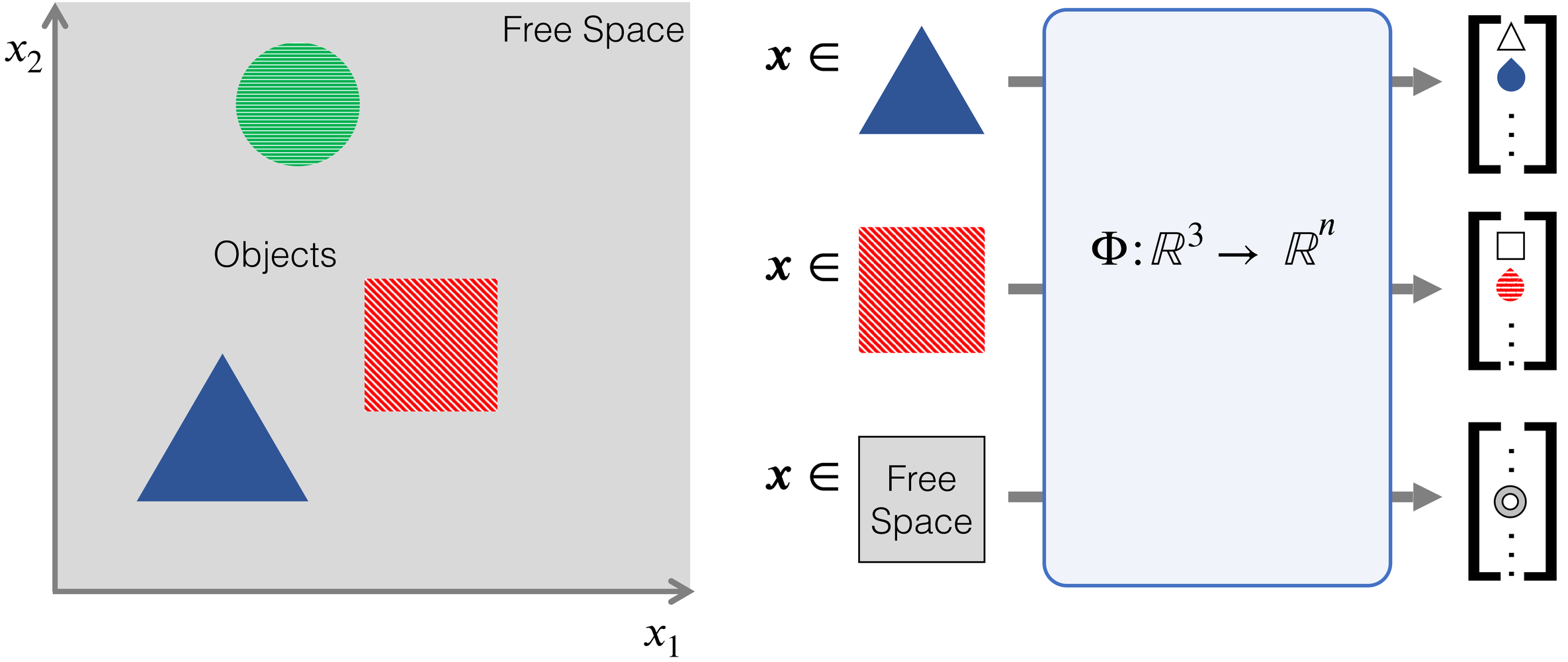
Camera Poses

Reconstruct

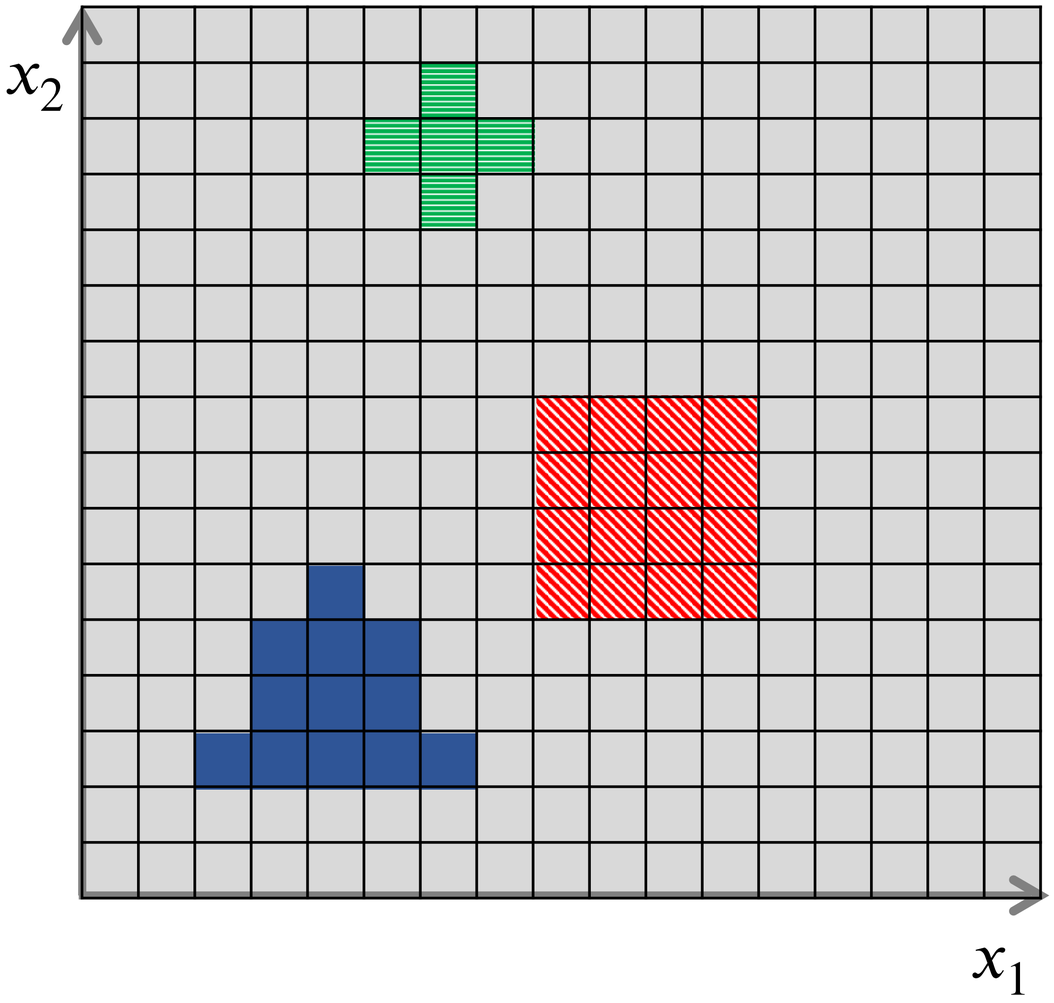


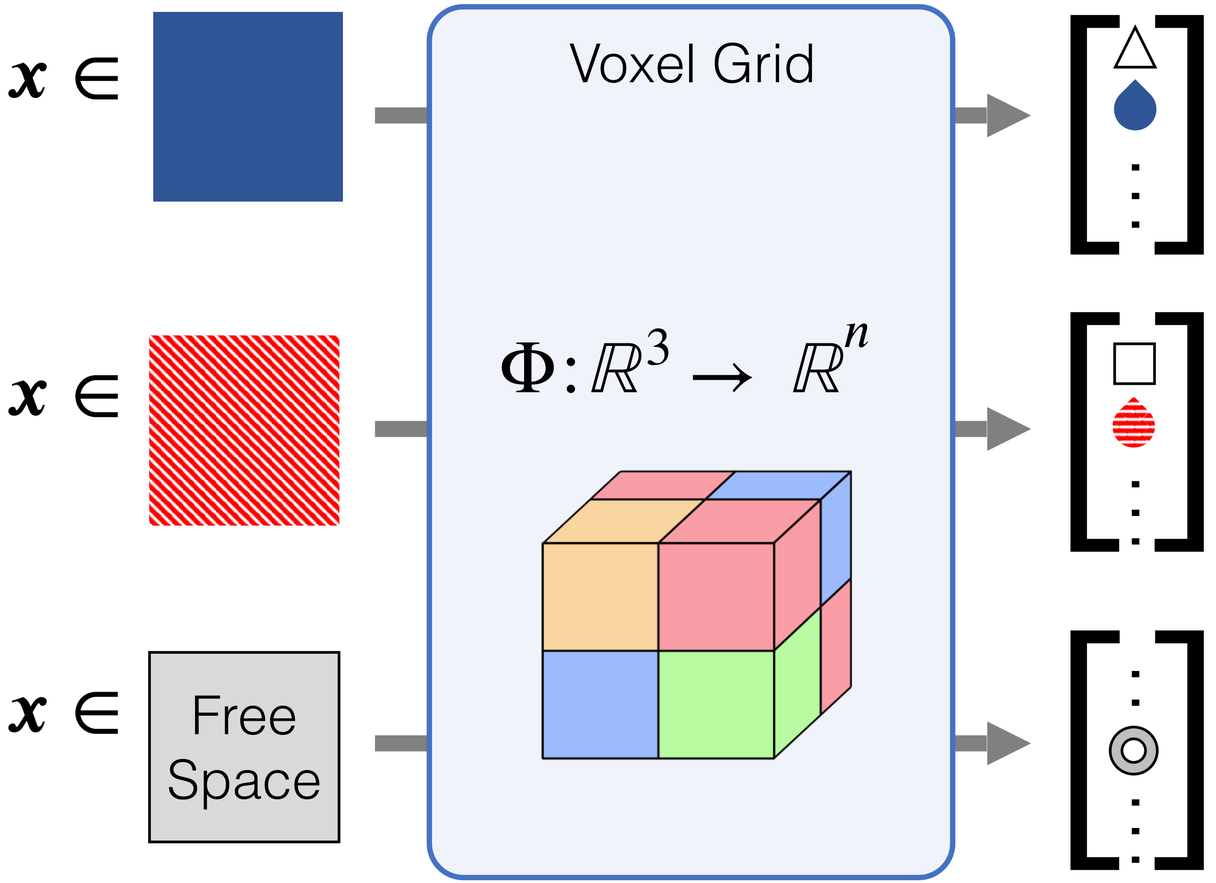
3D Scene

Starting Point: 3D Scene Representation

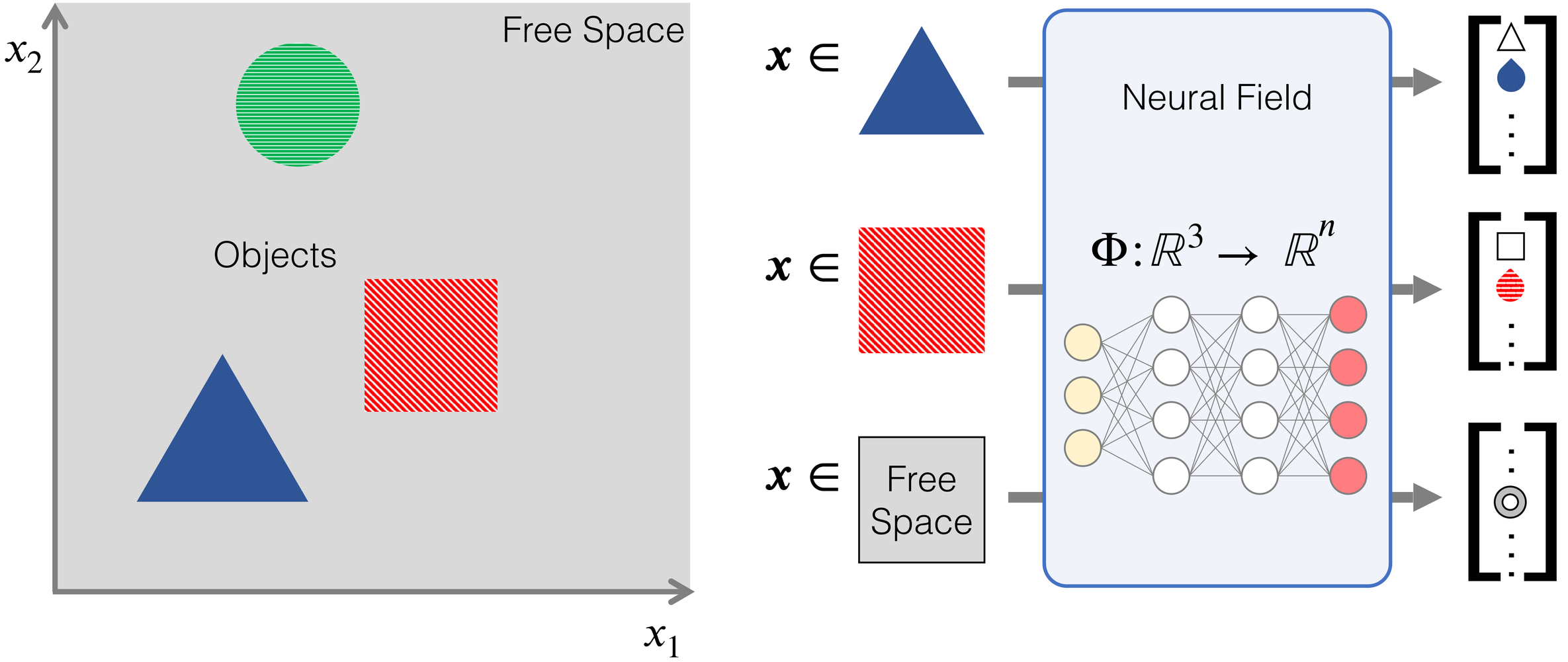


Starting Point: 3D Scene Representation





Starting Point: 3D Scene Representation



Differentiable Rendering



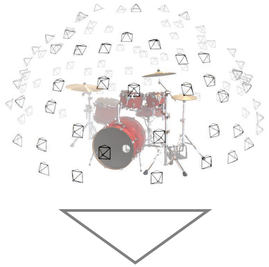
Scene Representation

Renderer



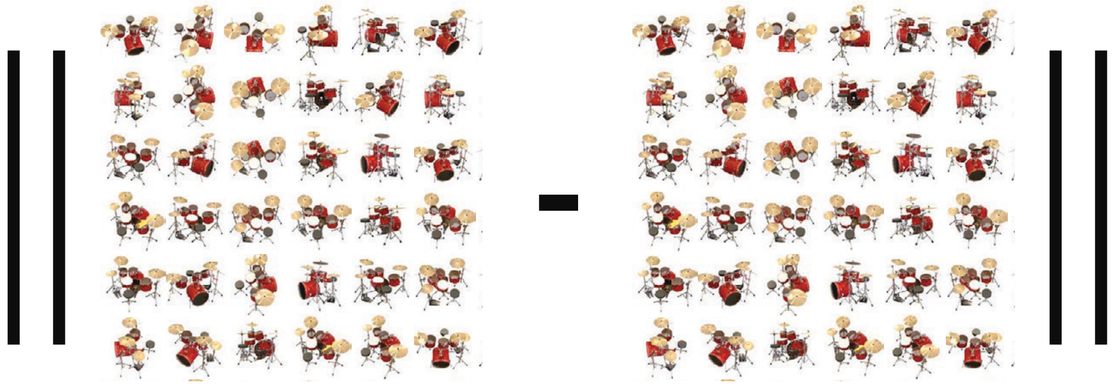
Rendered Images

Differentiable Rendering



Scene Representation

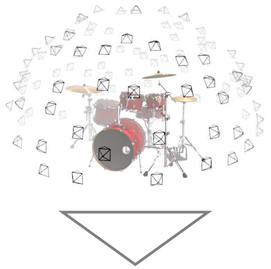
Renderer



Rendered Images

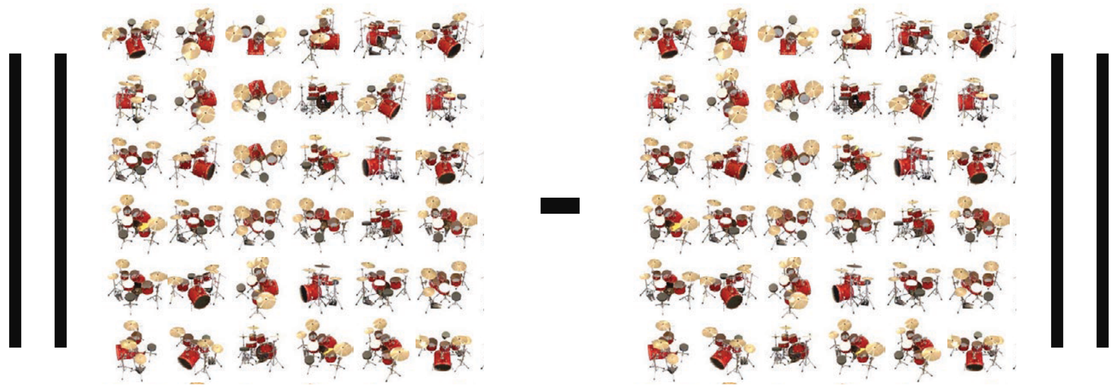
GT Images

Differentiable Rendering



Scene Representation

Differentiable Renderer

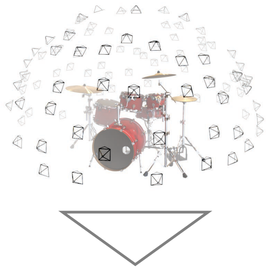


Rendered Images

GT Images

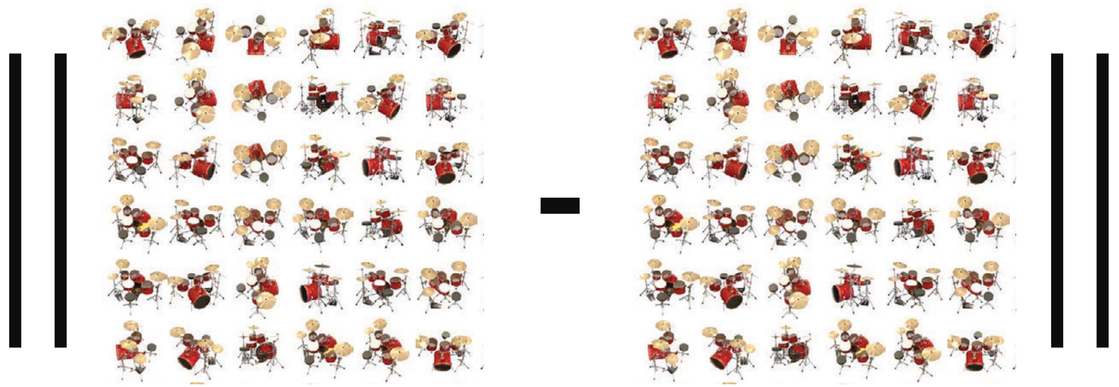
Optimization via SGD

Differentiable Rendering



Scene Representation

Differentiable Renderer

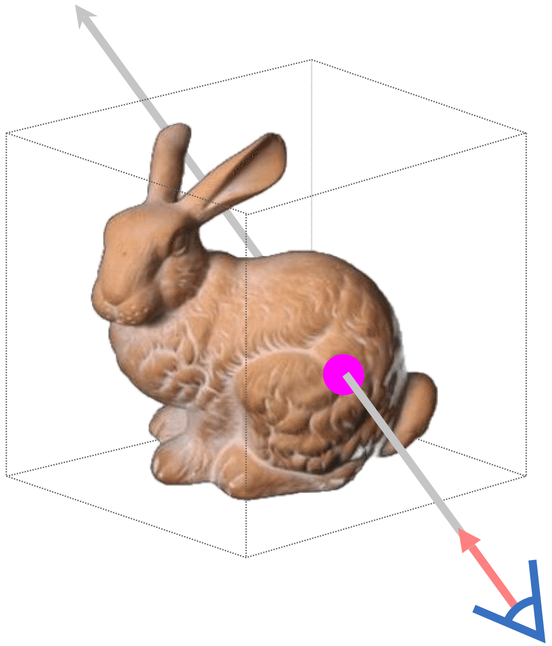


Rendered Images

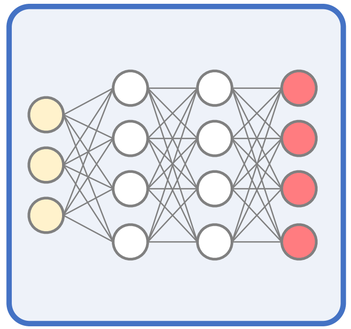
GT Images

Given an *observable* variable (pixel colors), we will build a differentiable forward model that we then use to estimate *unobserved (latent) variables* (geometry, appearance)!

General structure of Neural Renderers for 3D Fields



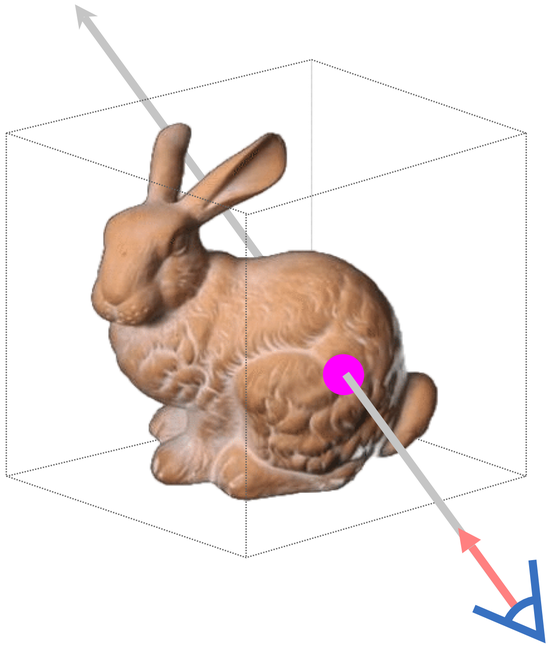


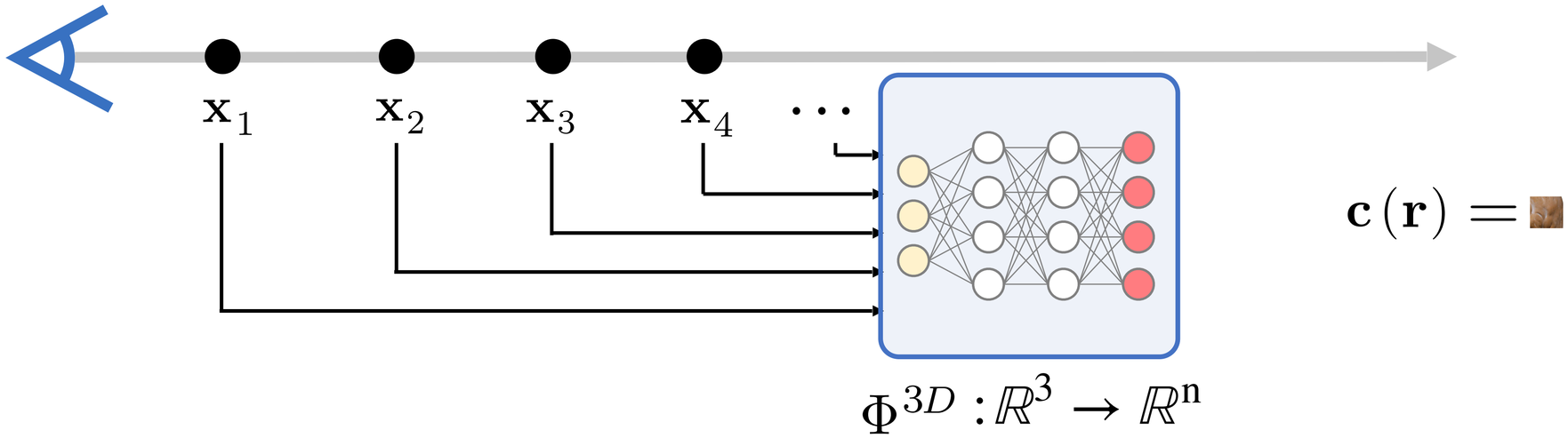


$$\Phi^{3D} : \mathbb{R}^3 \rightarrow \mathbb{R}^n$$

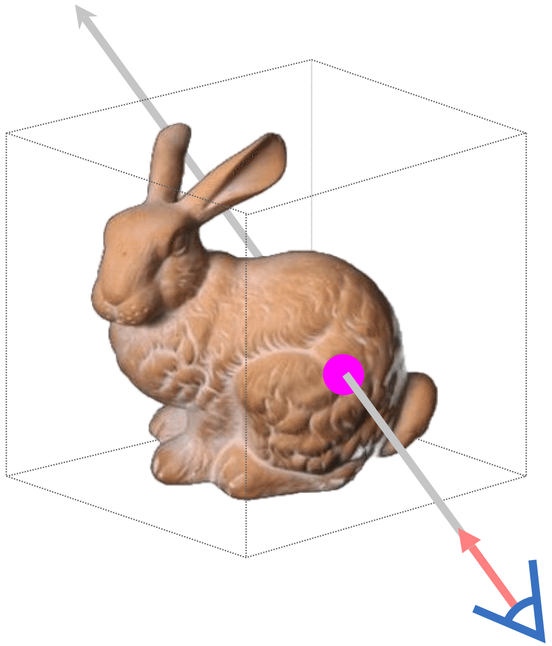
$$\mathbf{c}(\mathbf{r}) =$$ 

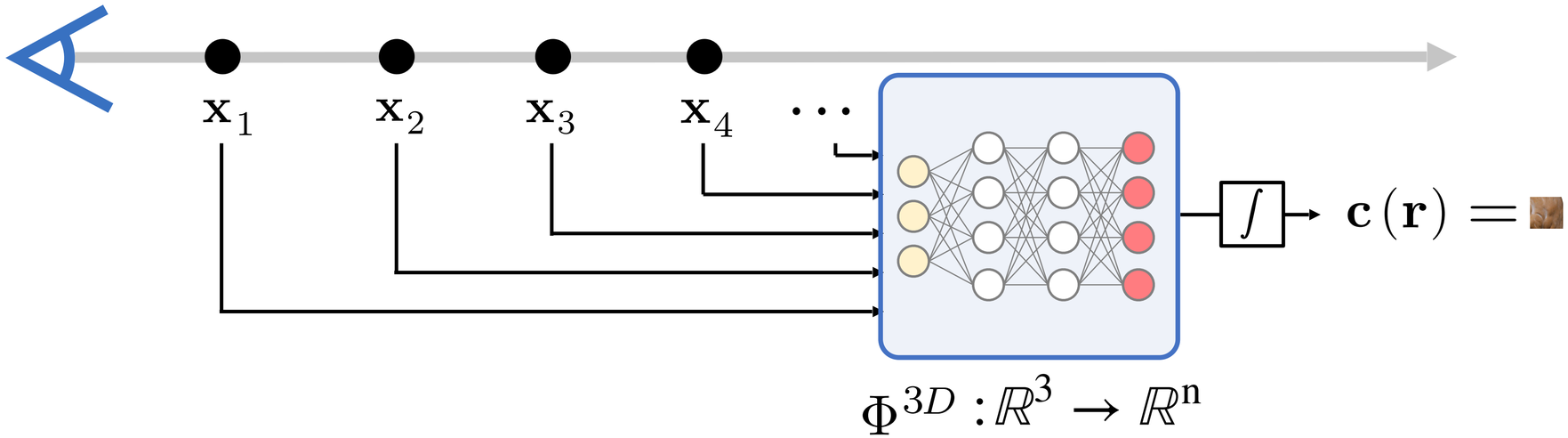
General structure of Neural Renderers for 3D Fields



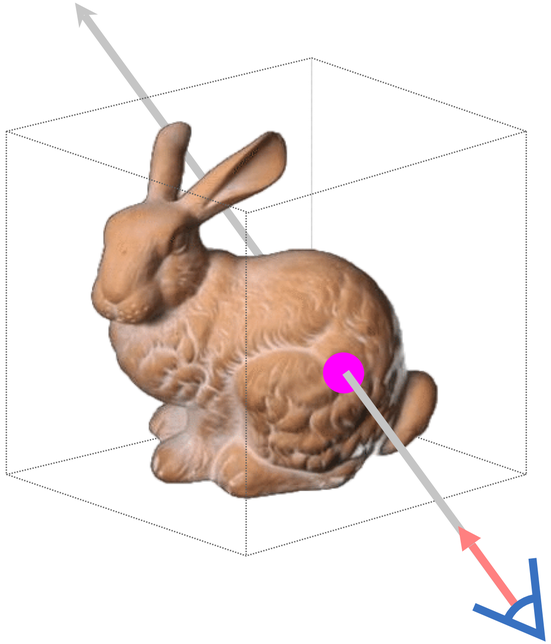


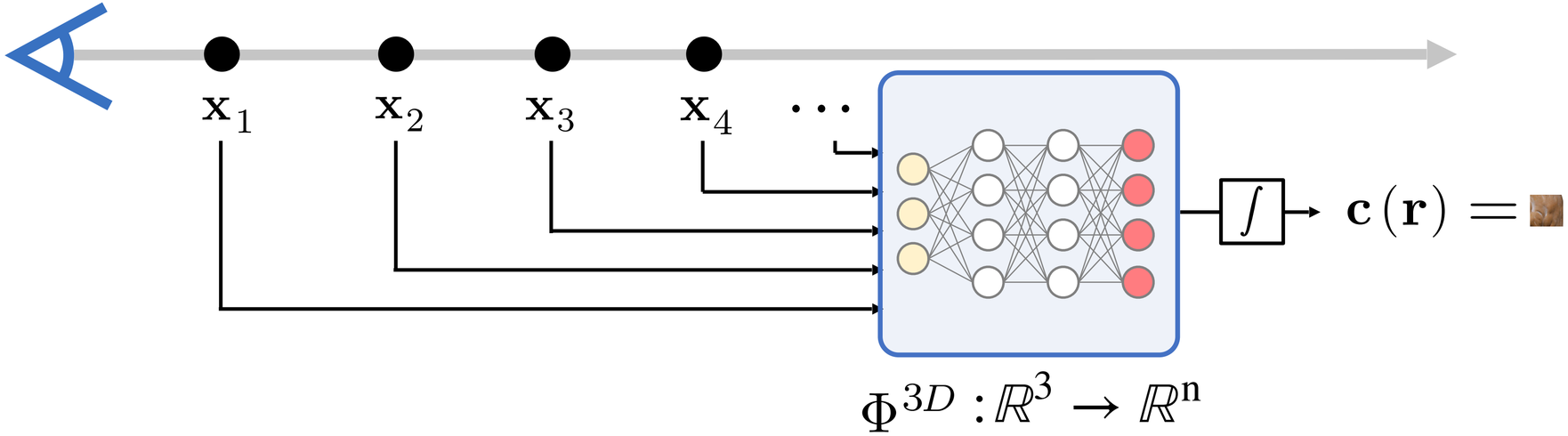
General structure of Neural Renderers for 3D Fields





General structure of Neural Renderers for 3D Fields





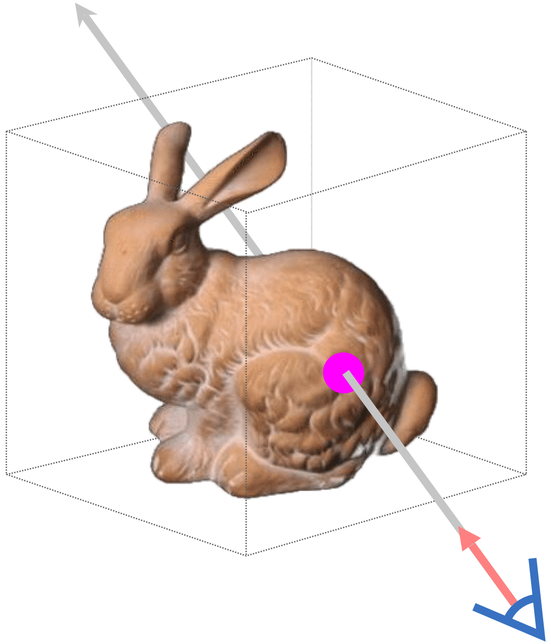
Sphere-Tracing [JC Hart, 1996]

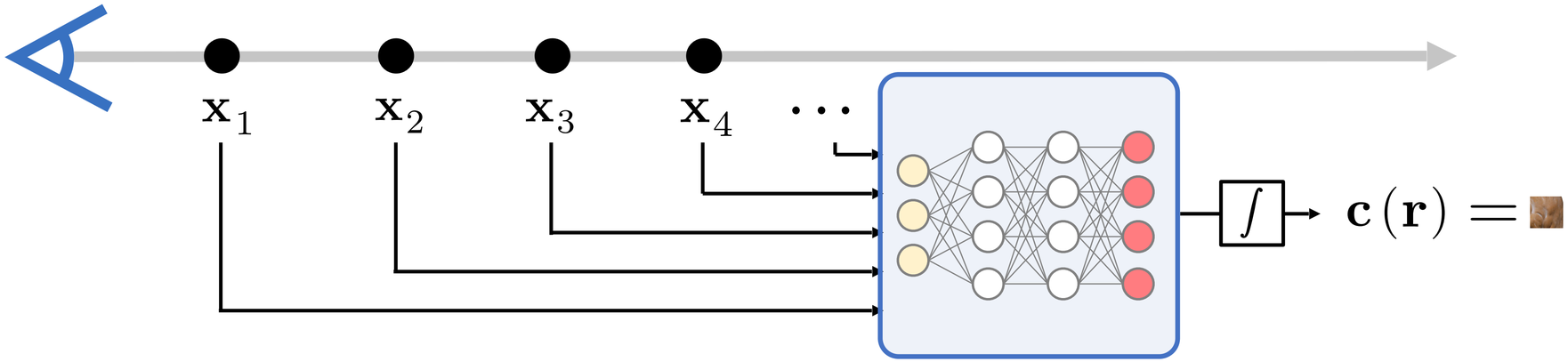
Volumetric

Hybrid implicit-volumetric

Learned aggregation

General structure of Neural Renderers for 3D Fields





Sphere-Tracing [JC Hart, 1996]

Volumetric

Hybrid implicit-volumetric

Learned aggregation

Summary: volume rendering integral estimate

Rendering model for ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

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

colors

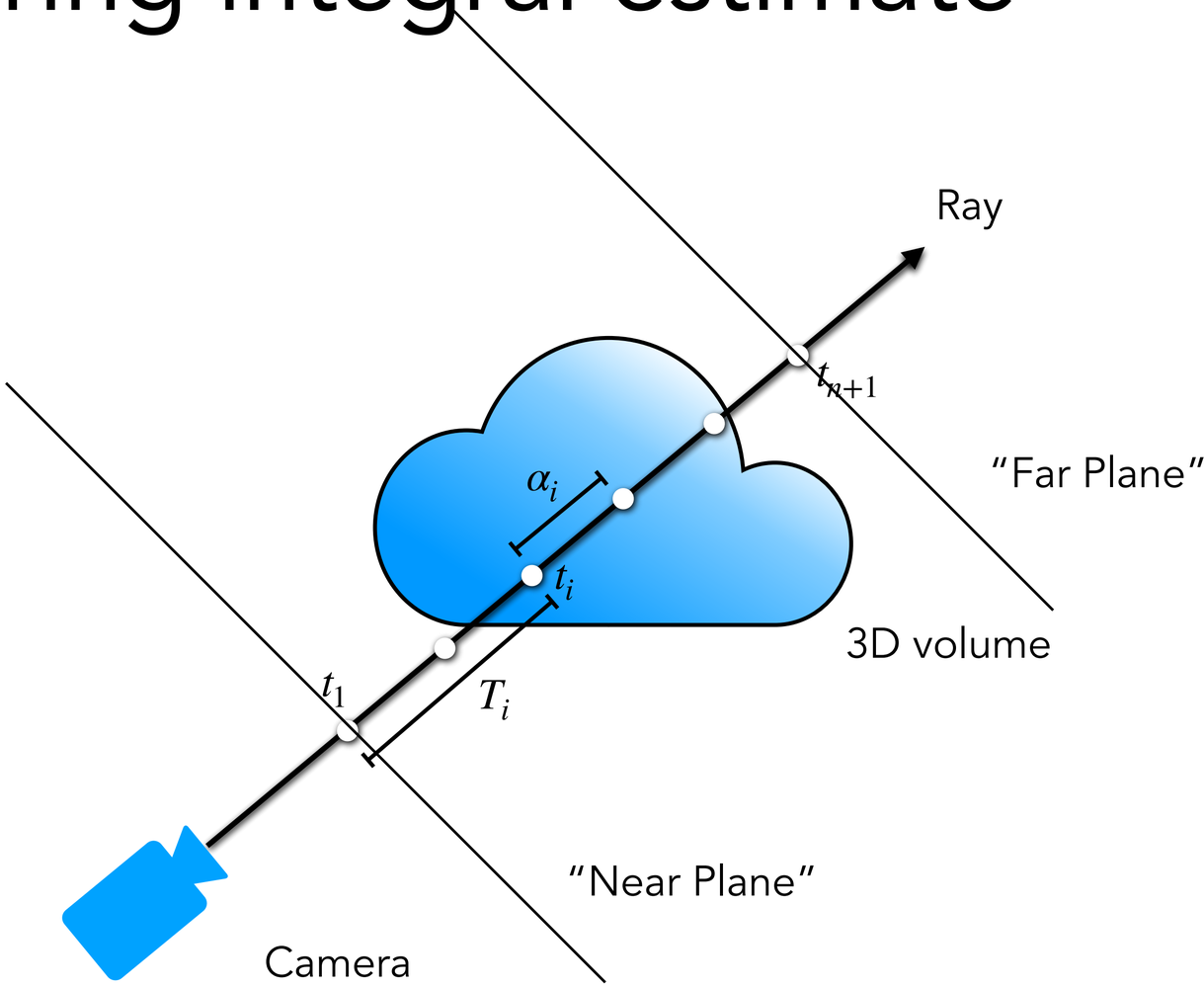
weights

How much light is blocked earlier along ray:

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$

How much light is contributed by ray segment *i*:

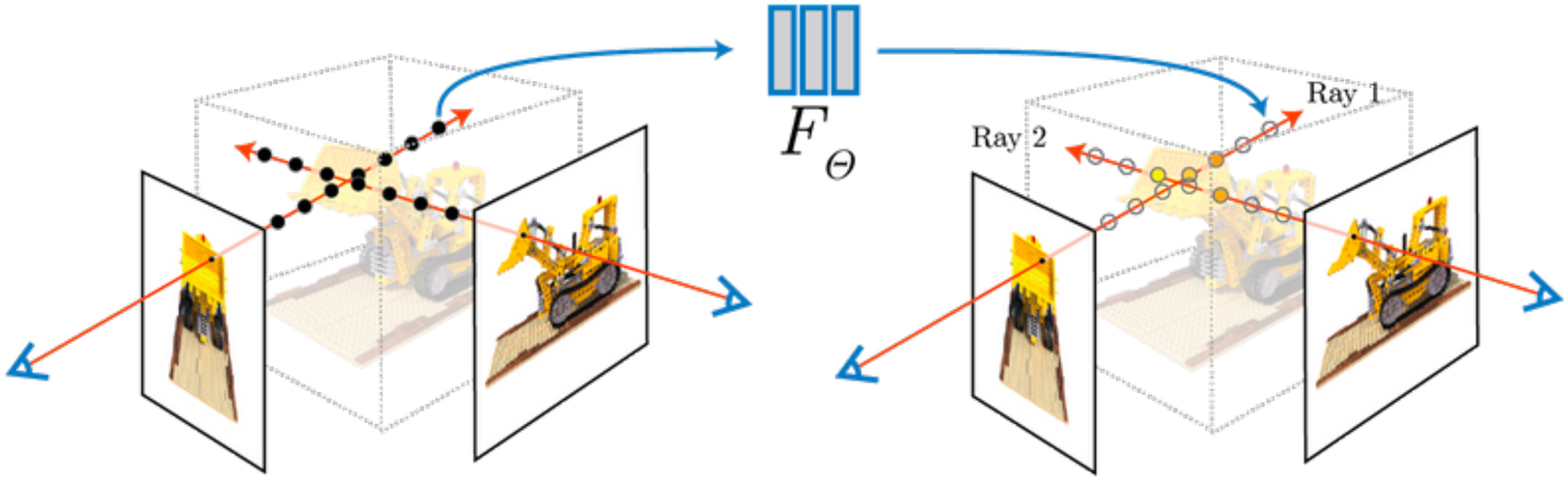
$$\alpha_i = 1 - \exp(-\sigma_i \delta_i)$$

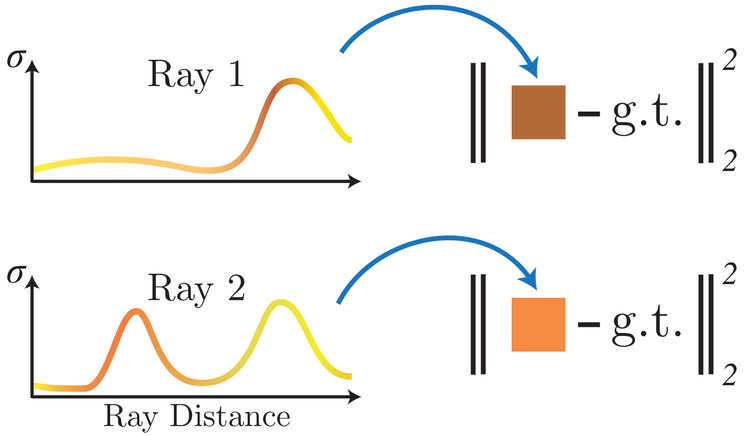


Neural Radiance Field: Parameterize Radiance Field via neural network

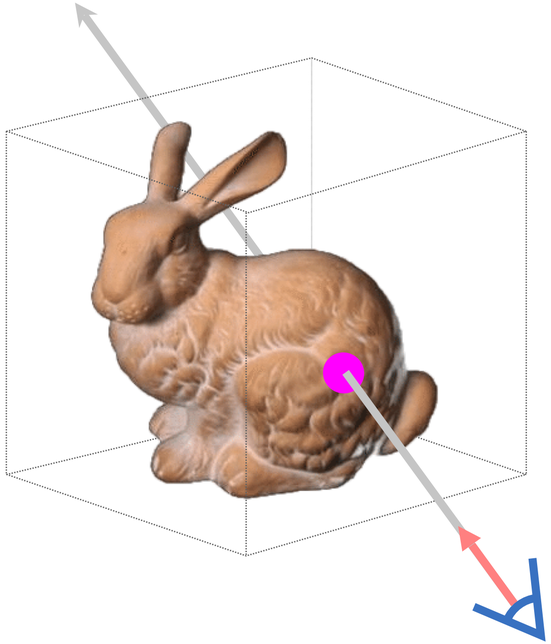
$$(x,y,z,\theta,\phi) \rightarrow \boxed{\quad} \rightarrow (RGB\sigma)$$

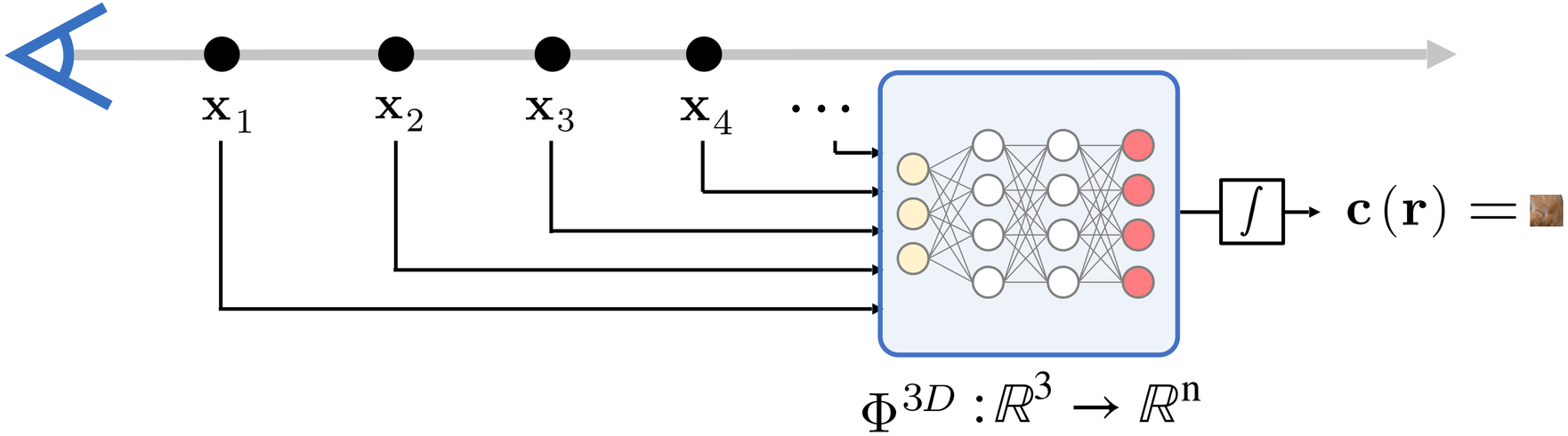
$$F_\Theta$$





General structure of Neural Renderers for 3D Fields



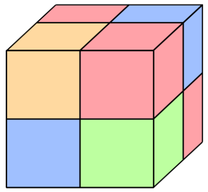


Sphere-Tracing [JC Hart, 1996]

Volumetric

Hybrid implicit-volumetric

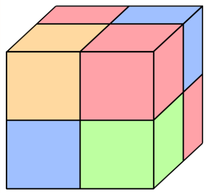
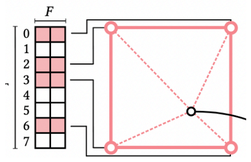
Learned aggregation

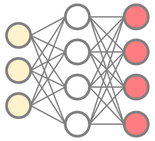


Voxel Grid

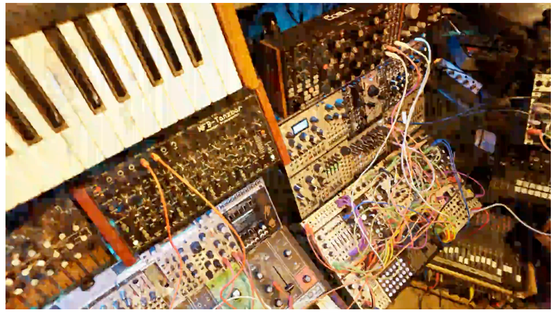


Plenoxels: Radiance Fields …
[Yu et al. 2022]
Direct Voxel Grid Optimization
[Sun et al. 2021]

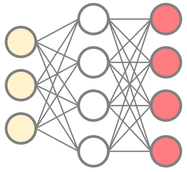





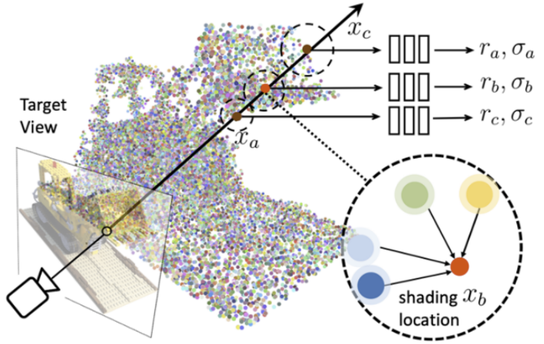
Voxel Grid + Hashmap + MLP



InstantNGP: Instant Neural …
[Müller et al. 2022]

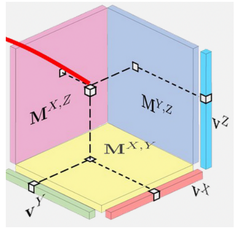



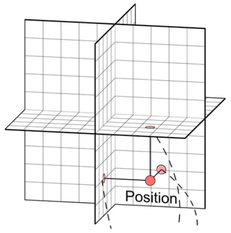
Point Cloud + MLP

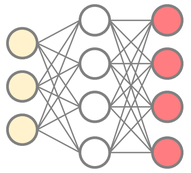


(b) Point-NeRF Representation with Volum

PointNeRF: Point-based Neural …
[Xu et al. 2022]



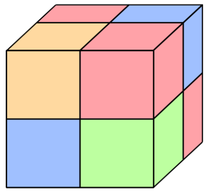




Tensor Factorization / Triplane + MLP



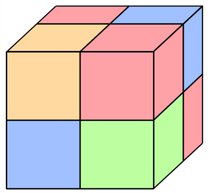
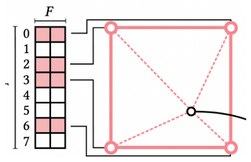
Efficient Geometry-aware 3D…
[Chan et al. 2022]
TensorRF: Tensor Radiance Fields
[Chen & Xu et al. 2022]

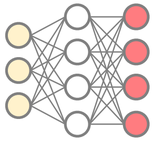


Voxel Grid

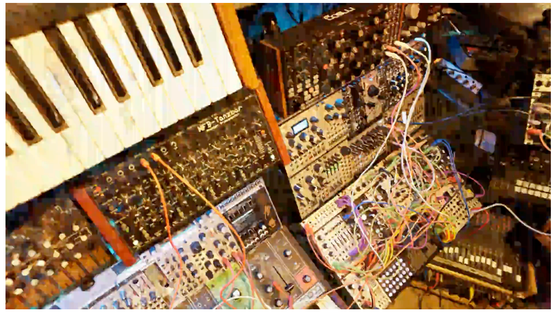


Plenoxels: Radiance Fields …
[Yu et al. 2022]
Direct Voxel Grid Optimization
[Sun et al. 2021]

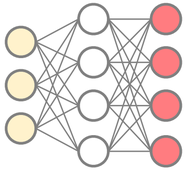





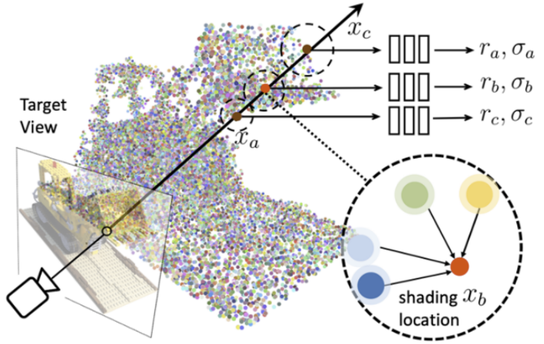
Voxel Grid + Hashmap + MLP



InstantNGP: Instant Neural …
[Müller et al. 2022]

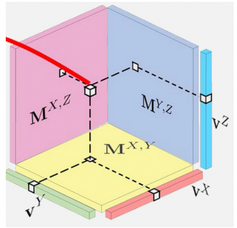



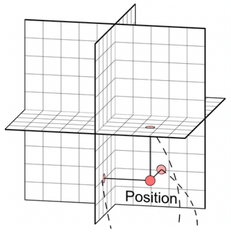
Point Cloud + MLP

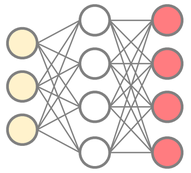


(b) Point-NeRF Representation with Volum

PointNeRF: Point-based Neural …
[Xu et al. 2022]





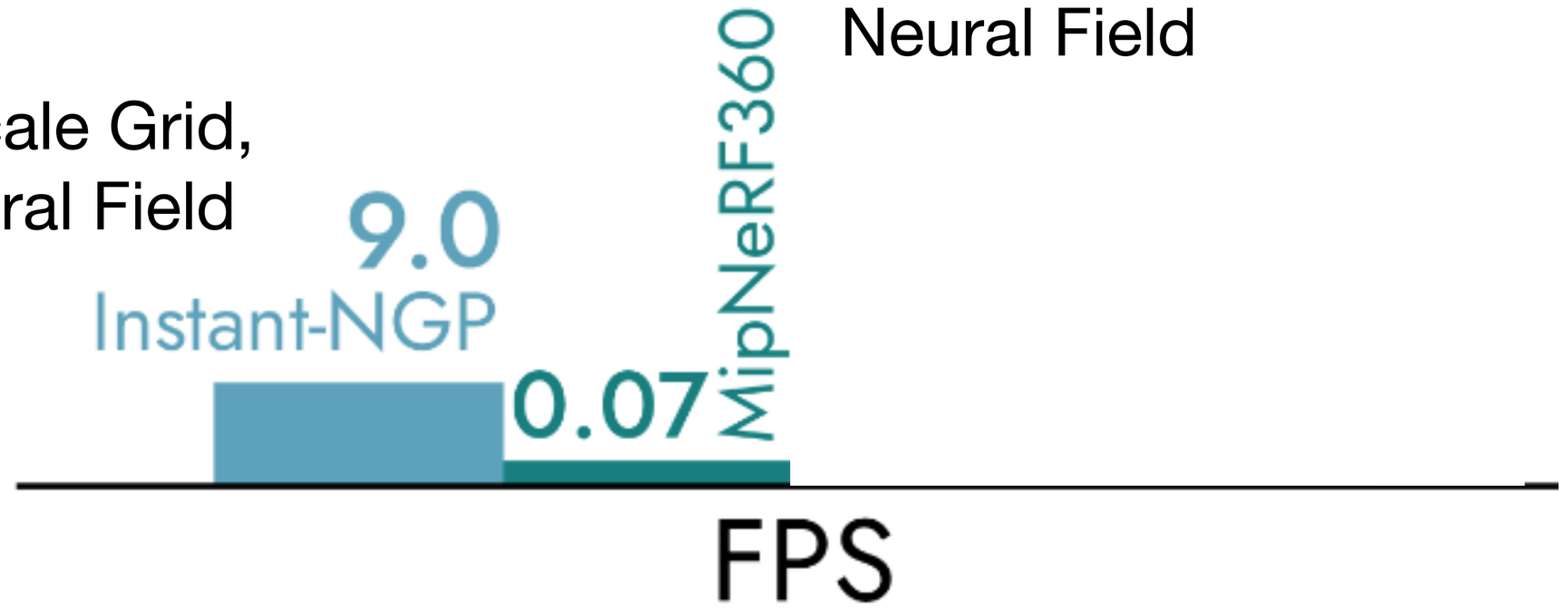


Tensor Factorization / Triplane + MLP

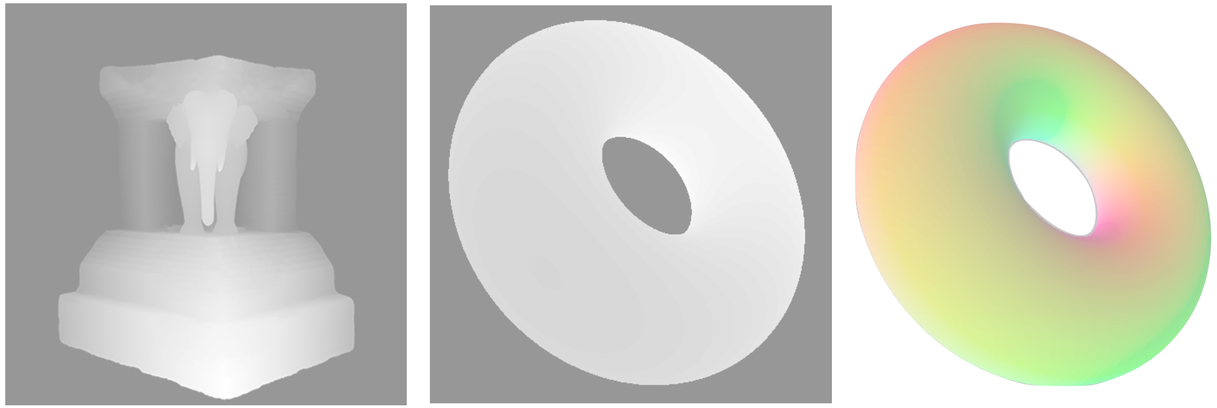


Efficient Geometry-aware 3D…
[Chan et al. 2022]
TensorRF: Tensor Radiance Fields
[Chen & Xu et al. 2022]

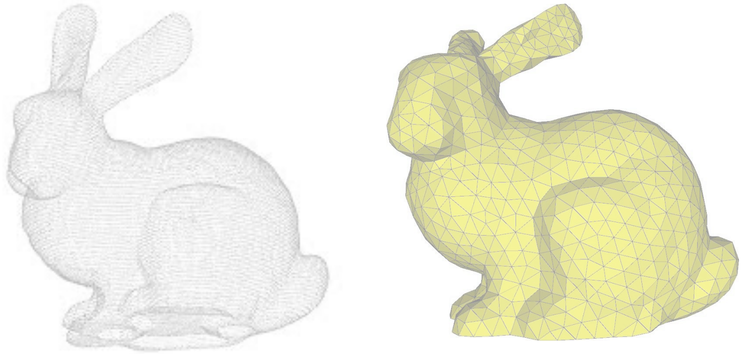
Hybrid Multi-Scale Grid,
HashMap, Neural Field



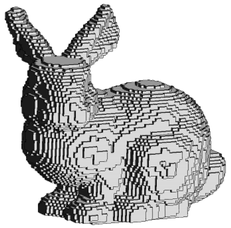
2.5D Representations

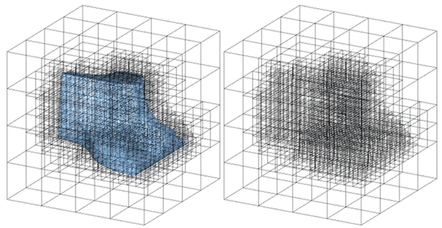


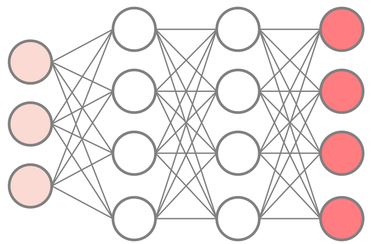
Surface Representations

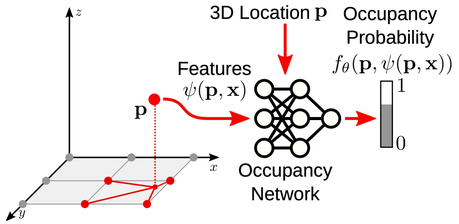


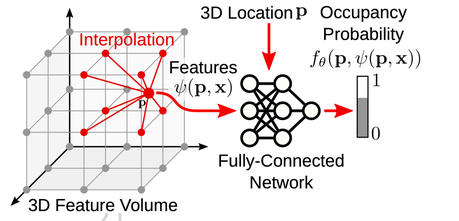
Field Parameterizations



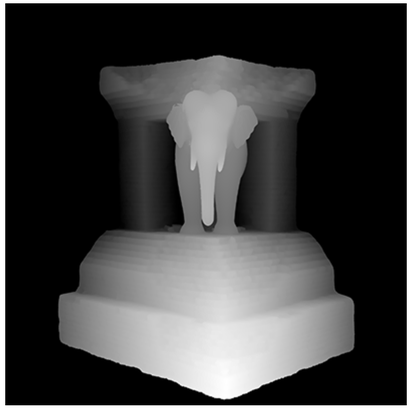


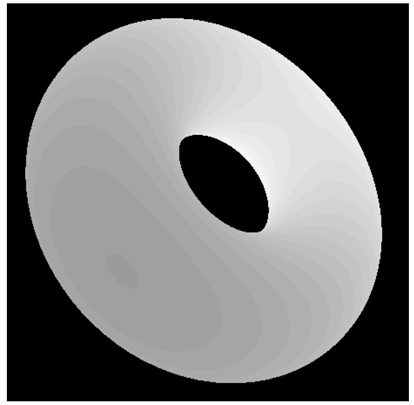


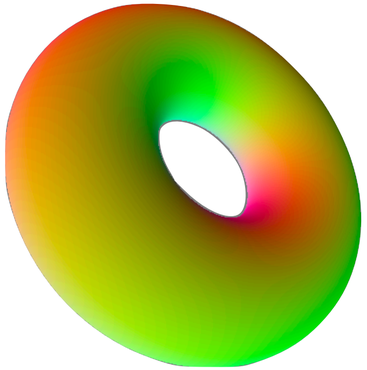




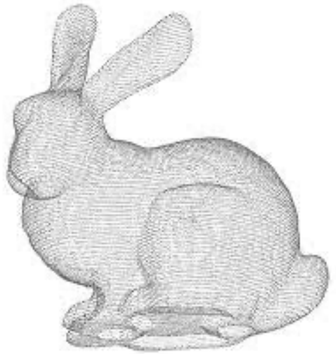
2.5D Representations

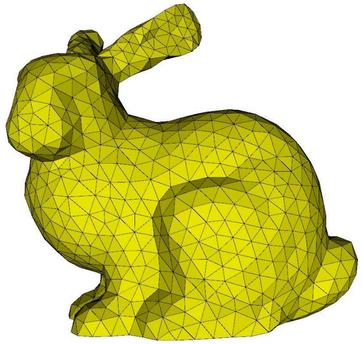




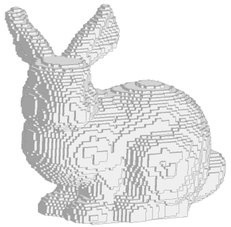


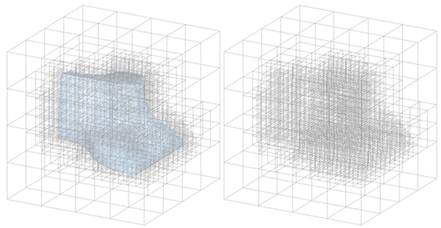
Surface Representations

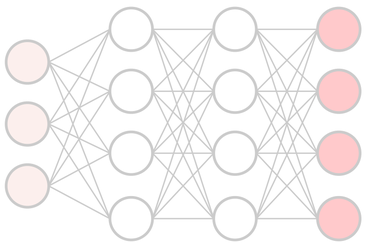


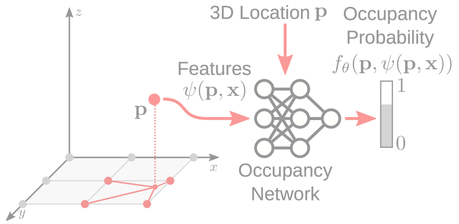


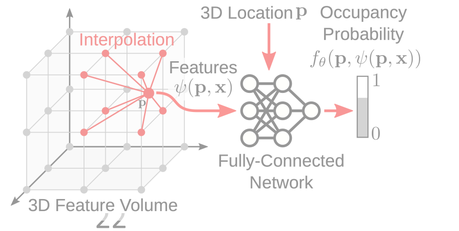
Field Parameterizations











3D Gaussian Splatting for Real-Time Radiance Field Rendering

BERNHARD KERBL*, Inria, Université Côte d'Azur, France
GEORGIOS KOPANAS*, Inria, Université Côte d'Azur, France
THOMAS LEIMKÜHLER, Max-Planck-Institut für Informatik, Germany
GEORGE DRETTAKIS, Inria, Université Côte d'Azur, France



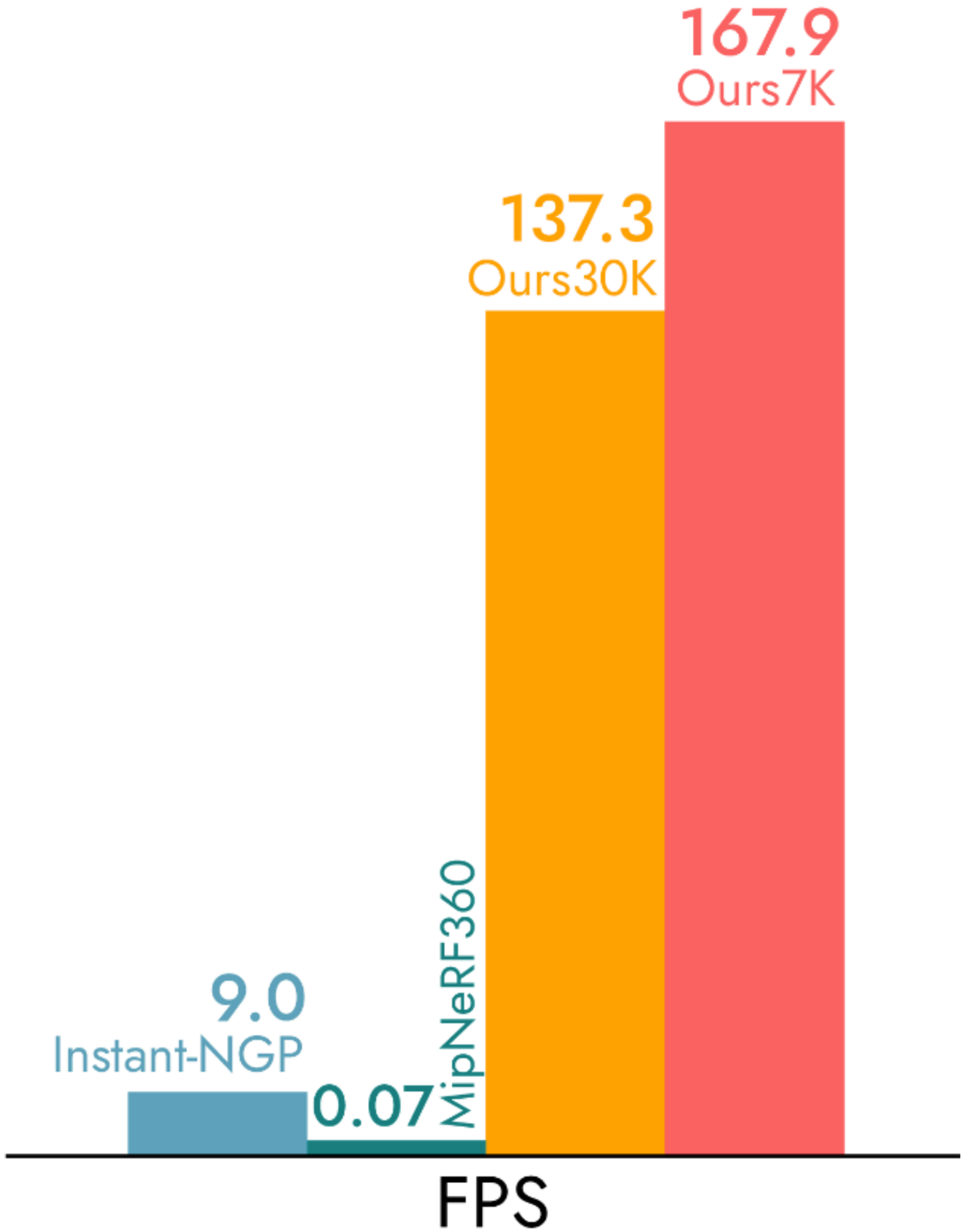
Fig. 1. Our method achieves real-time rendering of radiance fields while only requiring optimization times competitive with the faste performance is a novel 3D Gaussian scene representation coupled optimization and novel view synthesis. Note that for comparable trai this is the maximum quality they reach, by training for 51min we acl

EWA Splatting

Matthias Zwicker, Hanspeter Pfister, *Member*, *IEEE*,
Jeroen van Baar, and Markus Gross, *Member*, *IEEE*

Abstract—In this paper, we present a framework for high quality splatting based on elliptical Gaussian kernels. To avoid aliasing artifacts, we introduce the concept of a resampling filter, combining a reconstruction kernel with a low-pass filter. Because of the similarity to Heckbert's EWA (elliptical weighted average) filter for texture mapping, we call our technique EWA splatting. Our framework allows us to derive EWA splat primitives for volume data and for point-sampled surface data. It provides high image quality without aliasing artifacts or excessive blurring for volume data and, additionally, features anisotropic texture filtering for point-sampled surfaces. It also handles nonspherical volume kernels efficiently; hence, it is suitable for regular, rectilinear, and irregular volume datasets. Moreover, our framework introduces a novel approach to compute the footprint function, facilitating efficient perspective projection of arbitrary elliptical kernels at very little additional cost. Finally, we show that EWA volume reconstruction kernels can be reduced to surface reconstruction kernels. This makes our splat primitive universal in rendering surface and volume data.

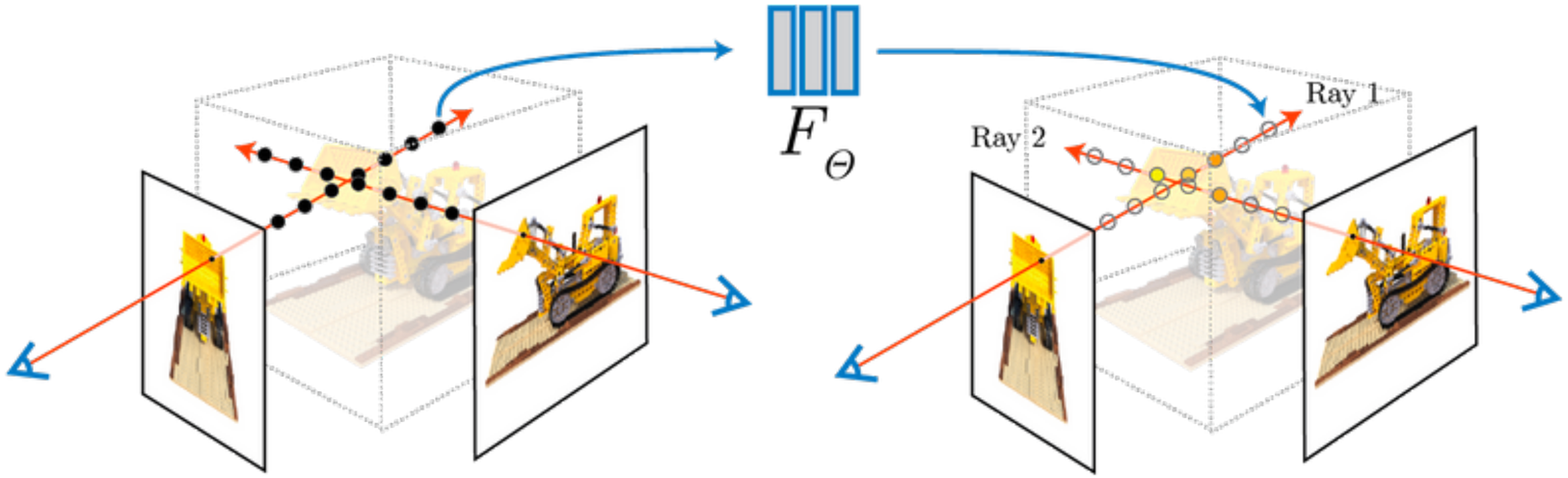
Index Terms—Rendering systems, volume rendering, texture mapping, splatting, antialiasing.

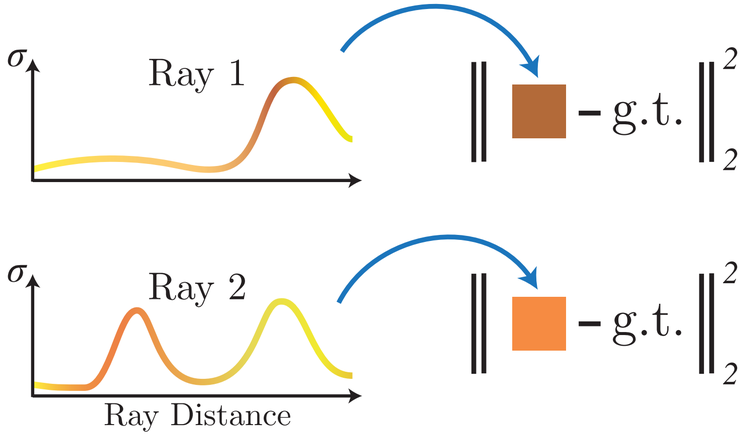

167.9
Ours7K
137.3
Ours30K
9.0
Instant-NGP
0.07
MipNeRF360
FPS

Neural Radiance Field: Parameterize Radiance Field densely, at *every* point in space

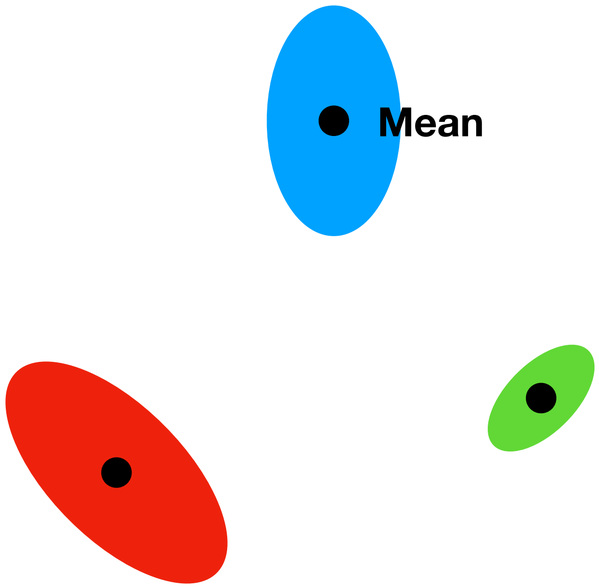
$$(x,y,z,\theta,\phi) \rightarrow \boxed{\quad} \rightarrow (RGB\sigma)$$

$$F_\Theta$$



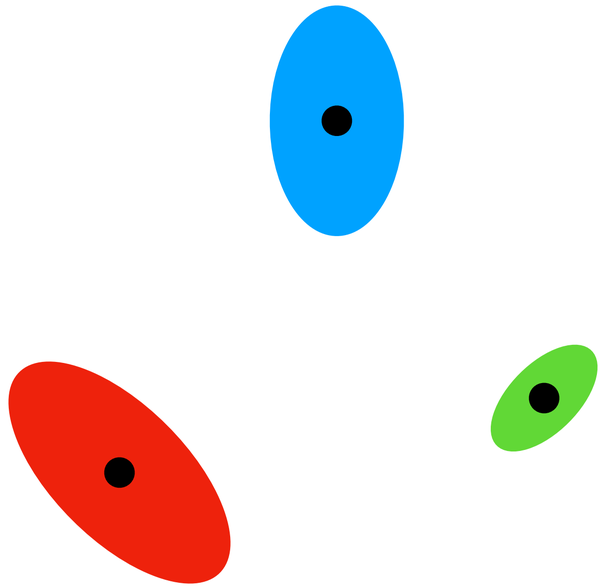


Key Idea: Parameterize Radiance Field *sparsely, only where density is nonzero*



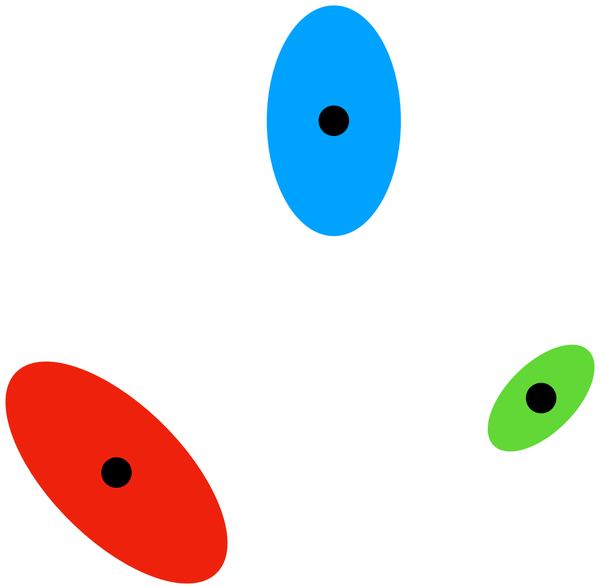
3D Gaussian Blobs floating in Space

Key Idea: Parameterize Radiance Field *sparsely, only where density is nonzero*

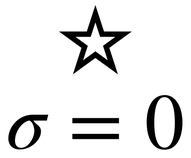


3D Gaussian Blobs floating in Space

Key Idea: Parameterize Radiance Field *sparsely, only where density is nonzero*

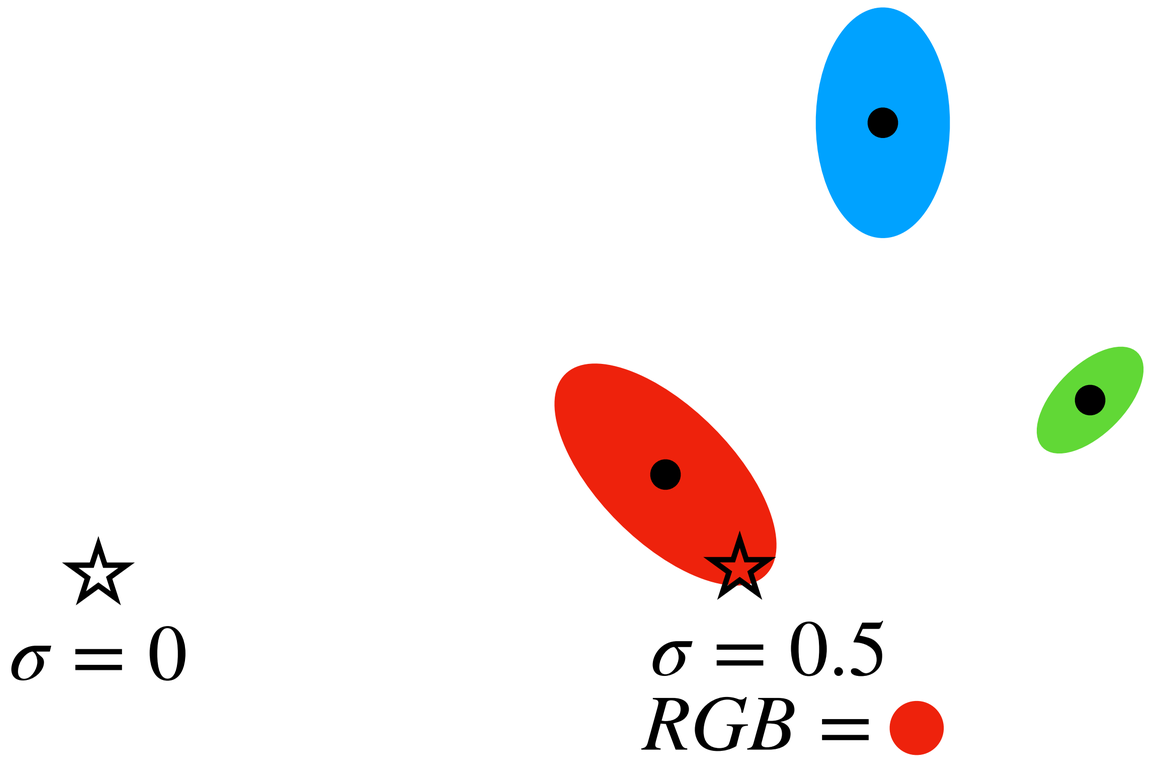


3D Gaussian Blobs floating in Space



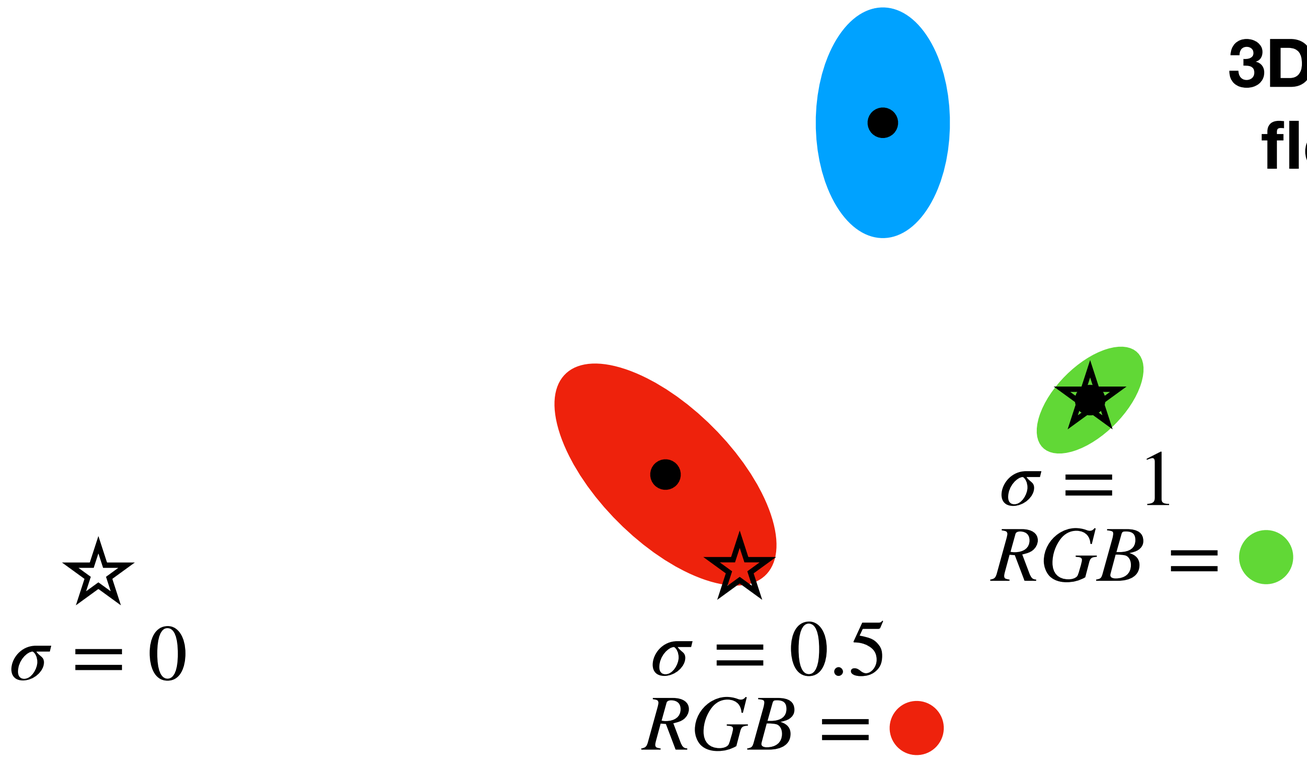
$\sigma = 0$

Key Idea: Parameterize Radiance Field *sparsely, only where density is nonzero*



3D Gaussian Blobs floating in Space

Key Idea: Parameterize Radiance Field *sparsely, only where density is nonzero*



Anisotropic Volumetric 3D Gaussians



Final Rendering

3D Gaussian Visualization

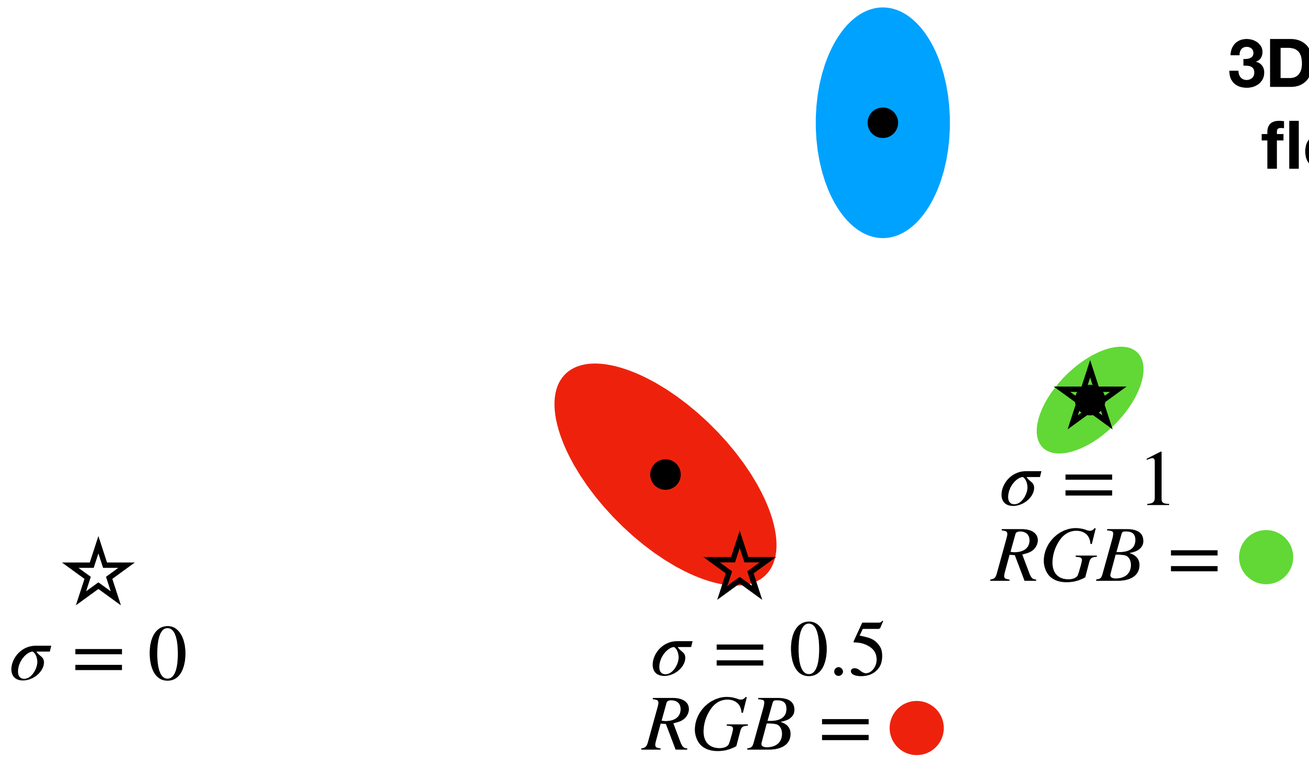
Anisotropic Volumetric 3D Gaussians



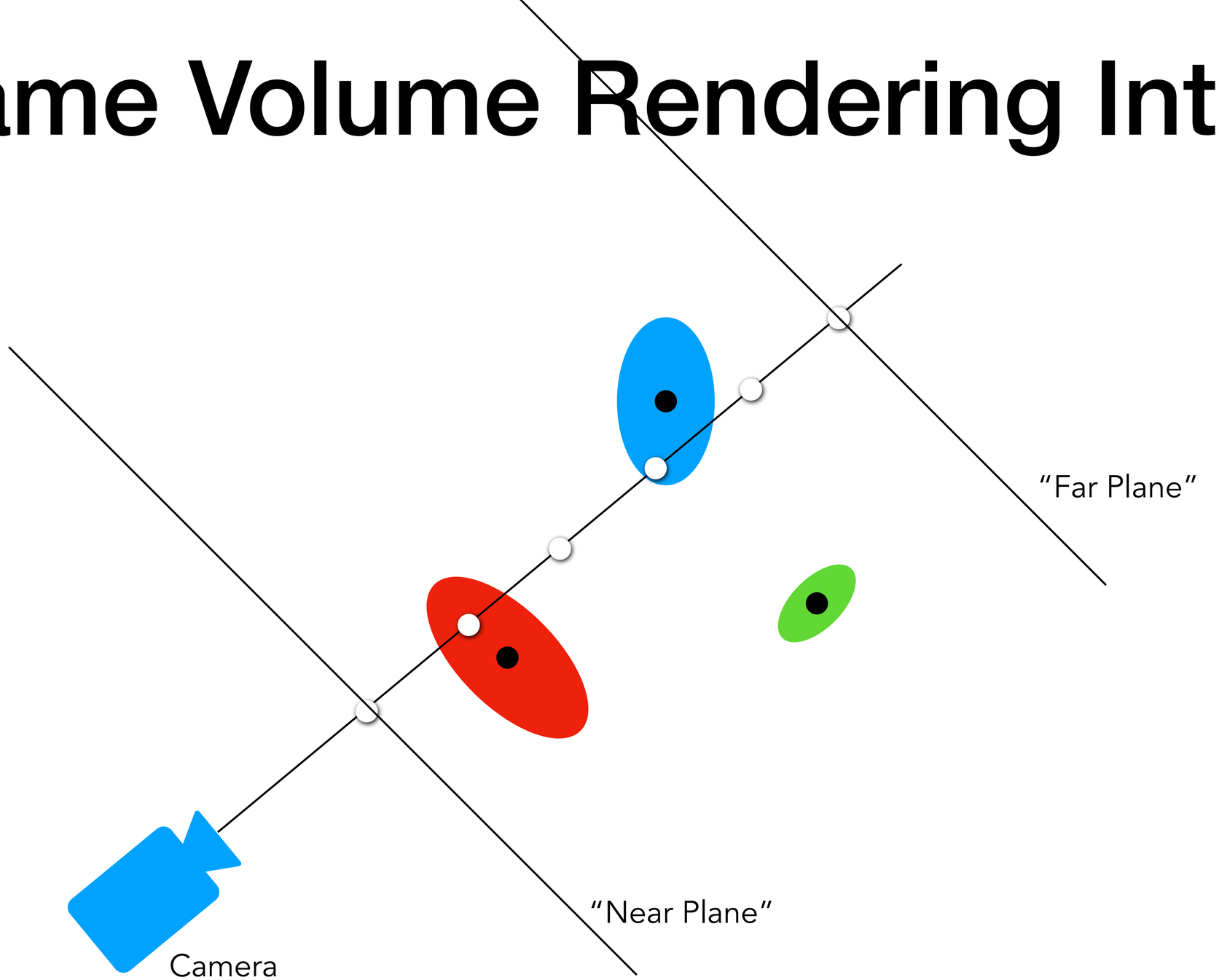
Final Rendering

3D Gaussian Visualization

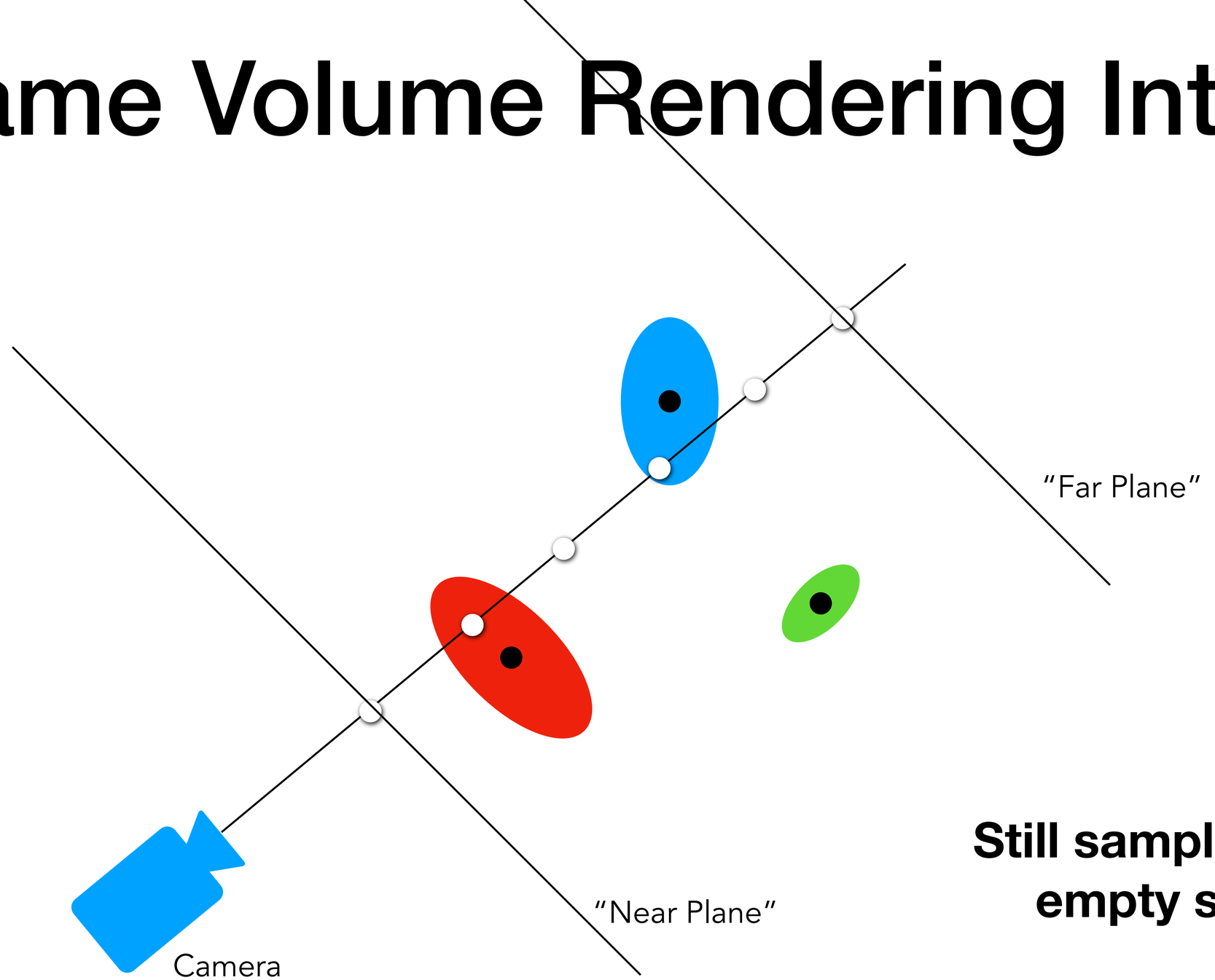
How to Render?



Same Volume Rendering Integral!

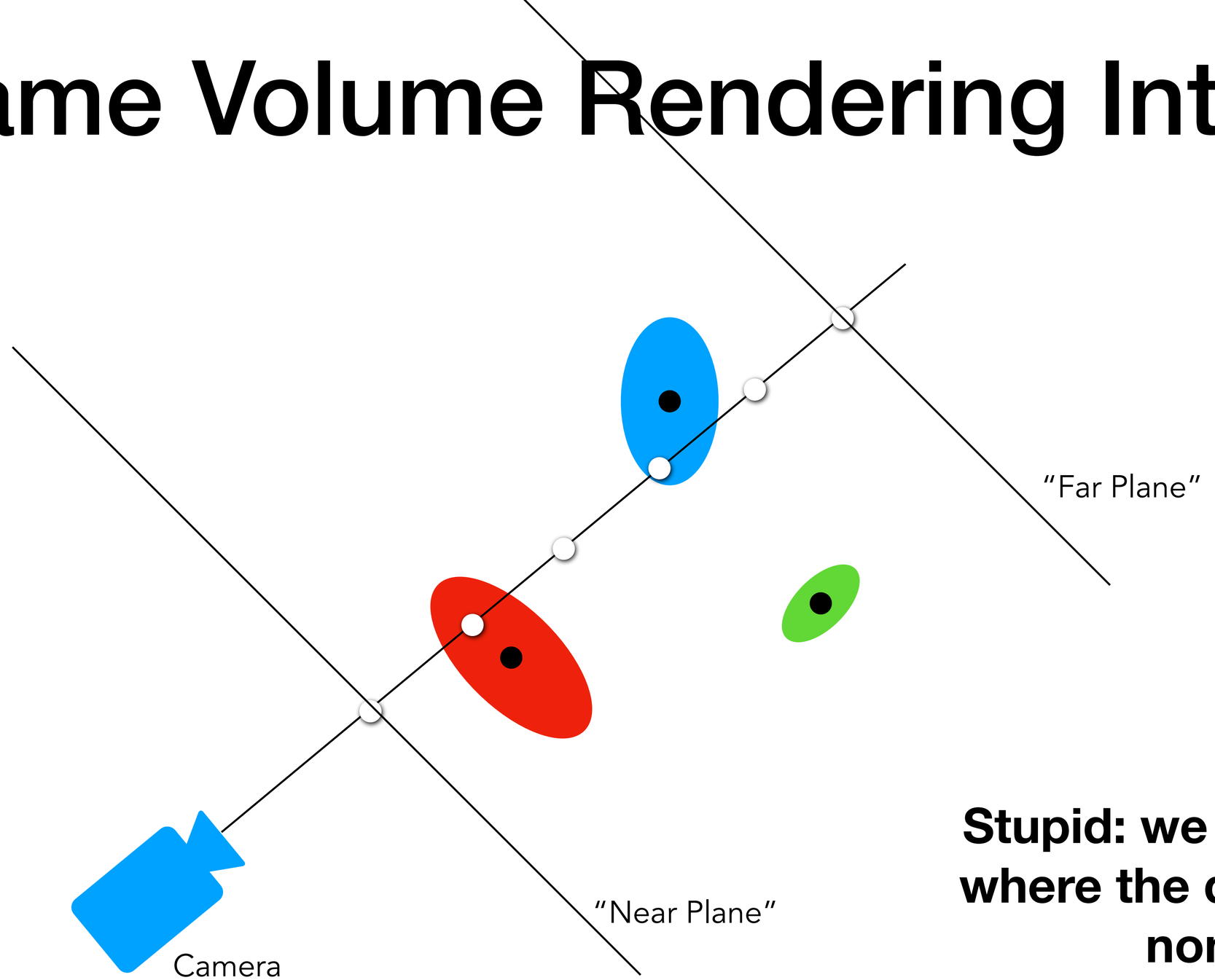


Same Volume Rendering Integral!



Still sampling lots of empty space…

Same Volume Rendering Integral!

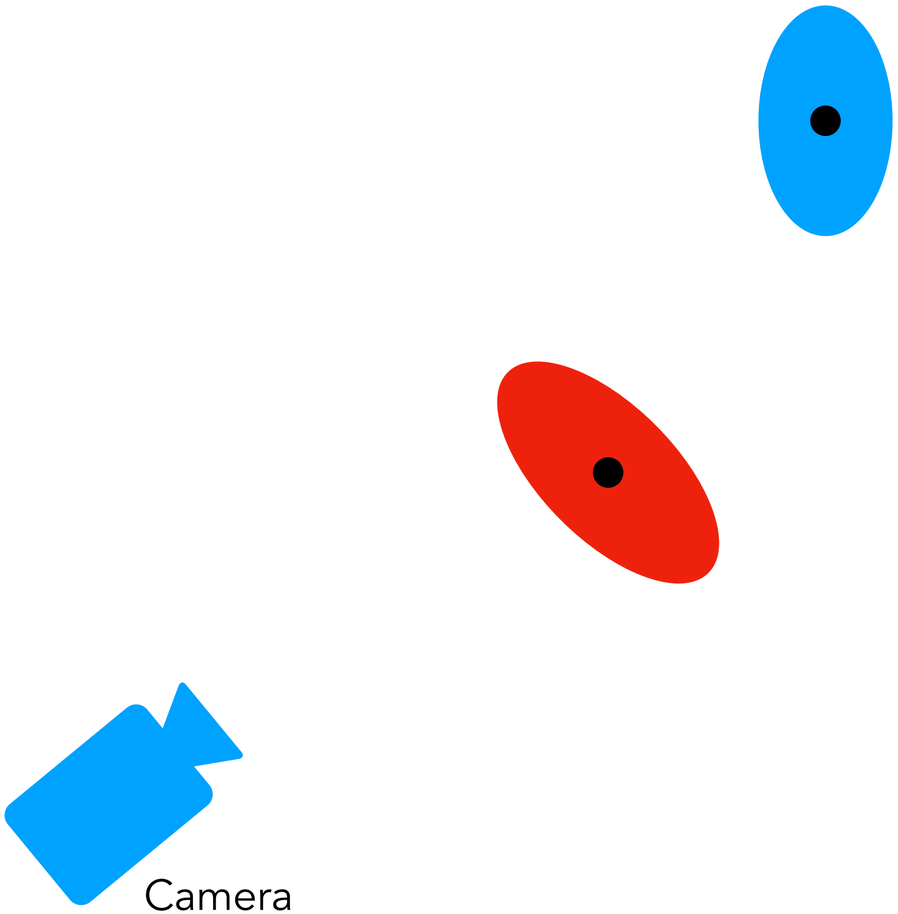


Stupid: we *already know* where the density will be nonzero!

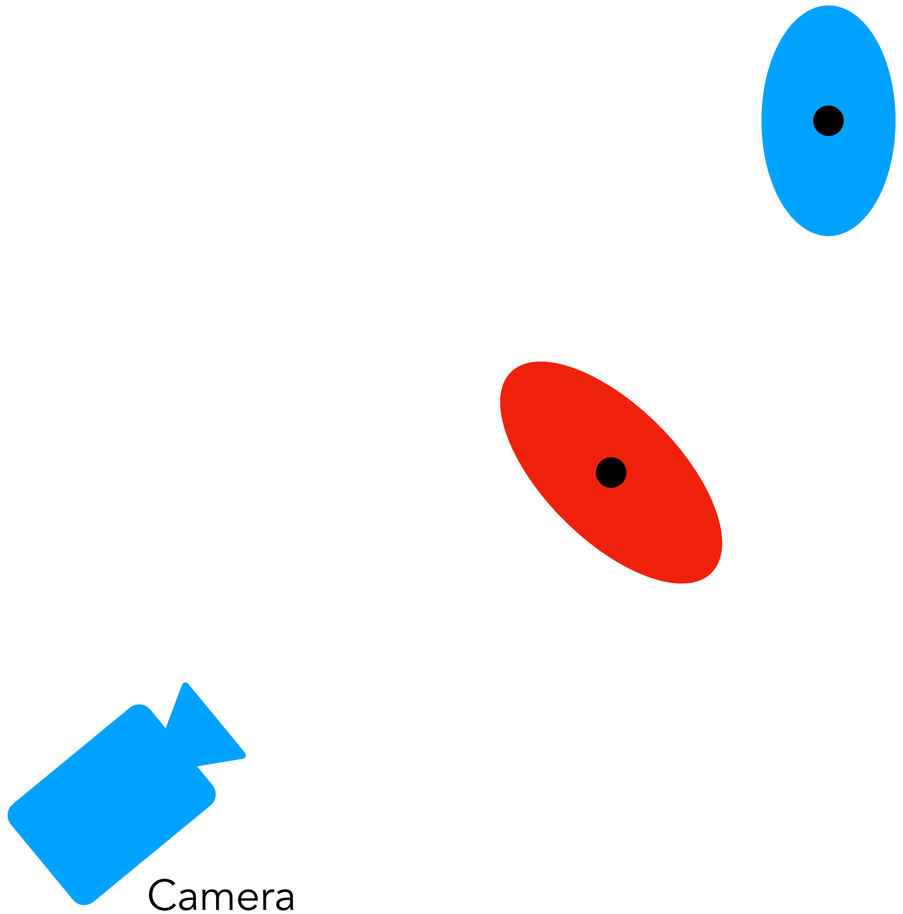
Gaussians are closed under affine transforms, integration

$$\mathcal{G}_{\mathbf{V}}(\mathbf{x}-\mathbf{p}) = \frac{1}{2\pi|\mathbf{V}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{p})^T \mathbf{V}^{-1}(\mathbf{x}-\mathbf{p})}$$

3D Covariance!



Gaussians are closed under affine transforms, integration



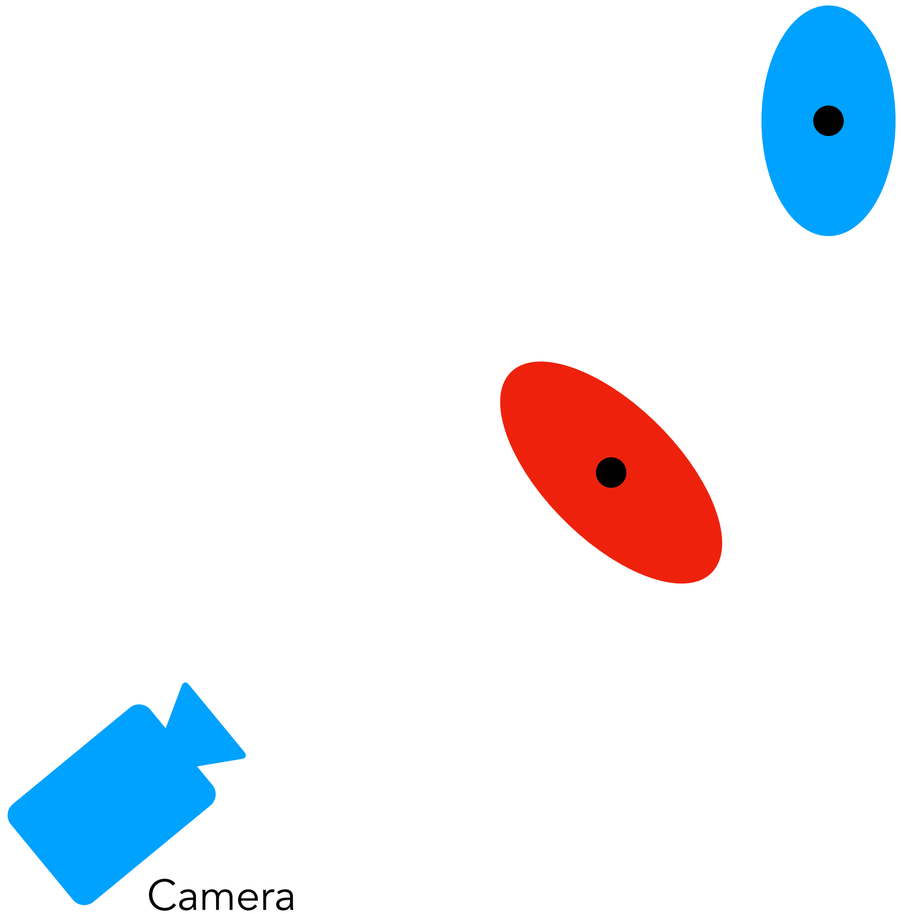
$$\mathcal{G}_{\mathbf{V}}(\mathbf{x}-\mathbf{p}) = \frac{1}{2\pi|\mathbf{V}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{p})^T \mathbf{V}^{-1}(\mathbf{x}-\mathbf{p})}$$

3D Covariance!

Affine mapping $\Phi = \mathbf{M}\mathbf{x} + \mathbf{p}$ of coordinates (such as cam2world matrix!):

$$\mathcal{G}_{\mathbf{V}}(\Phi^{-1}(\mathbf{u}) - \mathbf{p}) = \frac{1}{|\mathbf{M}^{-1}|}\mathcal{G}_{\mathbf{M}\mathbf{V}\mathbf{M}^T}(\mathbf{u} - \Phi(\mathbf{p}))$$

Gaussians are closed under affine transforms, integration



$$\mathcal{G}_{\mathbf{V}}(\mathbf{x}-\mathbf{p}) = \frac{1}{2\pi|\mathbf{V}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{p})^T \mathbf{V}^{-1}(\mathbf{x}-\mathbf{p})}$$

3D Covariance!

Affine mapping $\Phi = \mathbf{M}\mathbf{x} + \mathbf{p}$ of coordinates (such as cam2world matrix!):

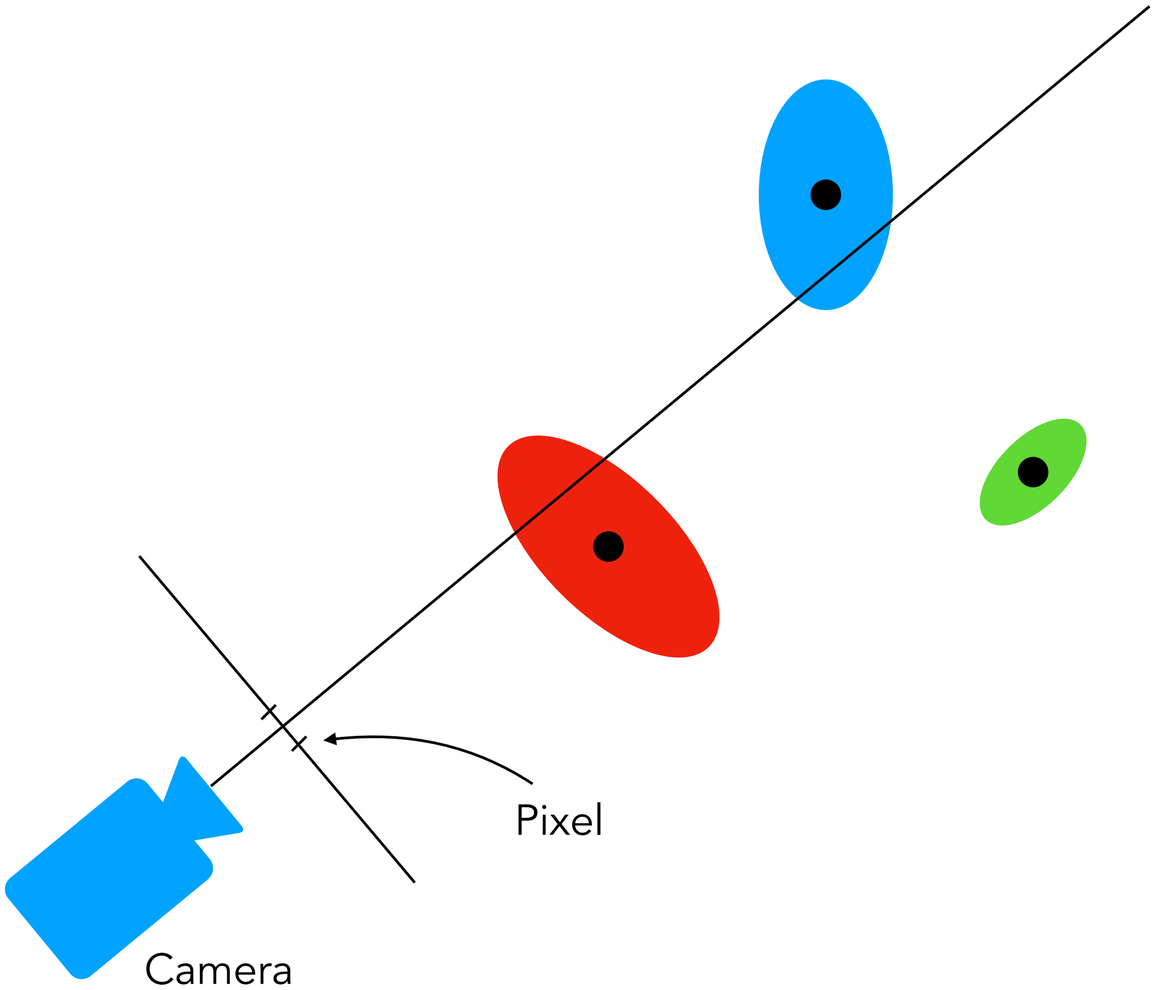
$$\mathcal{G}_{\mathbf{V}}(\Phi^{-1}(\mathbf{u}) - \mathbf{p}) = \frac{1}{|\mathbf{M}^{-1}|}\mathcal{G}_{\mathbf{M}\mathbf{V}\mathbf{M}^T}(\mathbf{u} - \Phi(\mathbf{p}))$$

Integrate along axis:

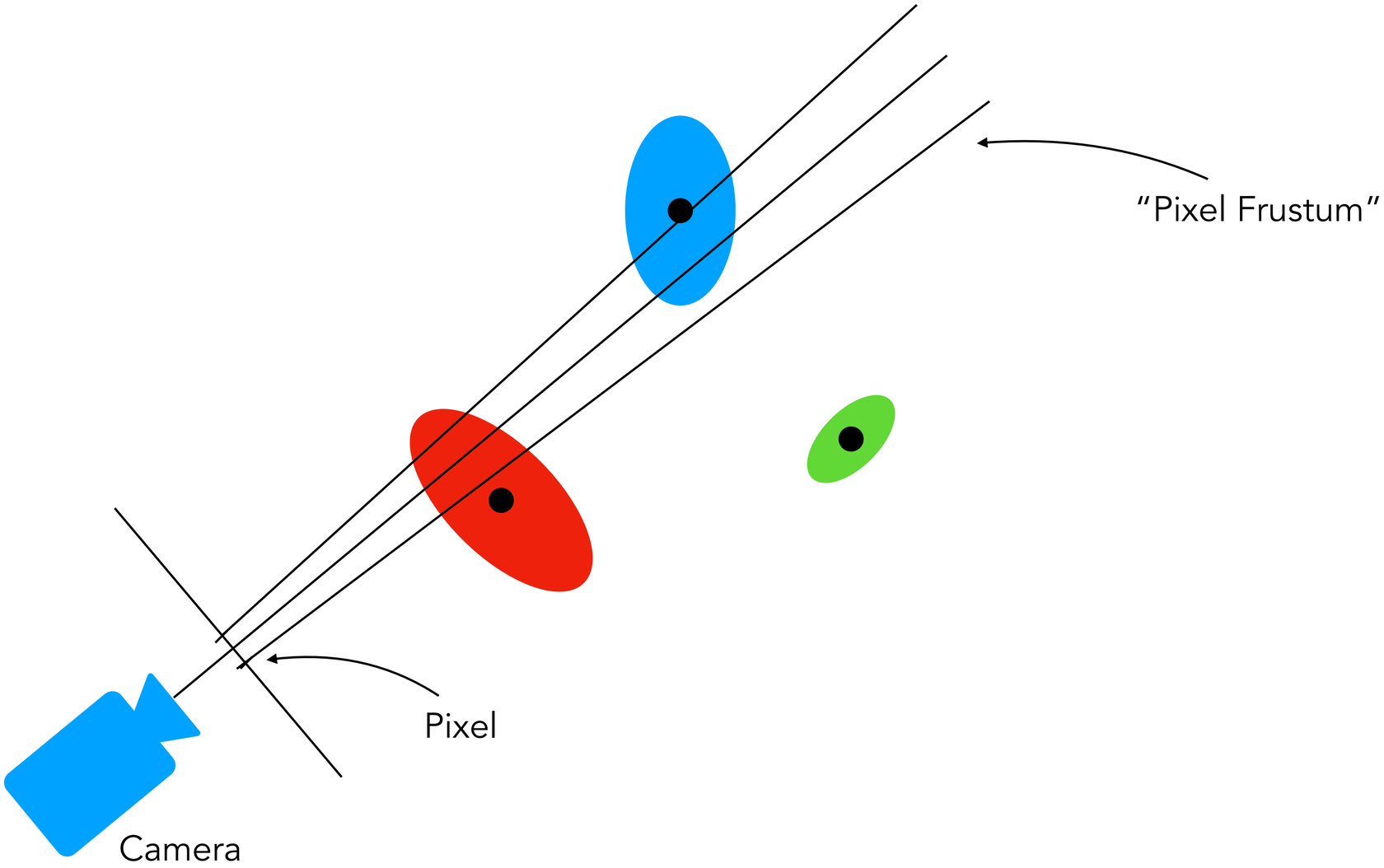
$$\int_{\mathbb{R}} \mathcal{G}^3_{\mathbf{V}}(\mathbf{x}-\mathbf{p})\, dx_2 = \mathcal{G}^2_{\hat{\mathbf{V}}}(\hat{\mathbf{x}}-\hat{\mathbf{p}})$$

$$\mathbf{V} = \begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix} \Leftrightarrow \begin{pmatrix} a & b \\ b & d \end{pmatrix} = \hat{\mathbf{V}}$$

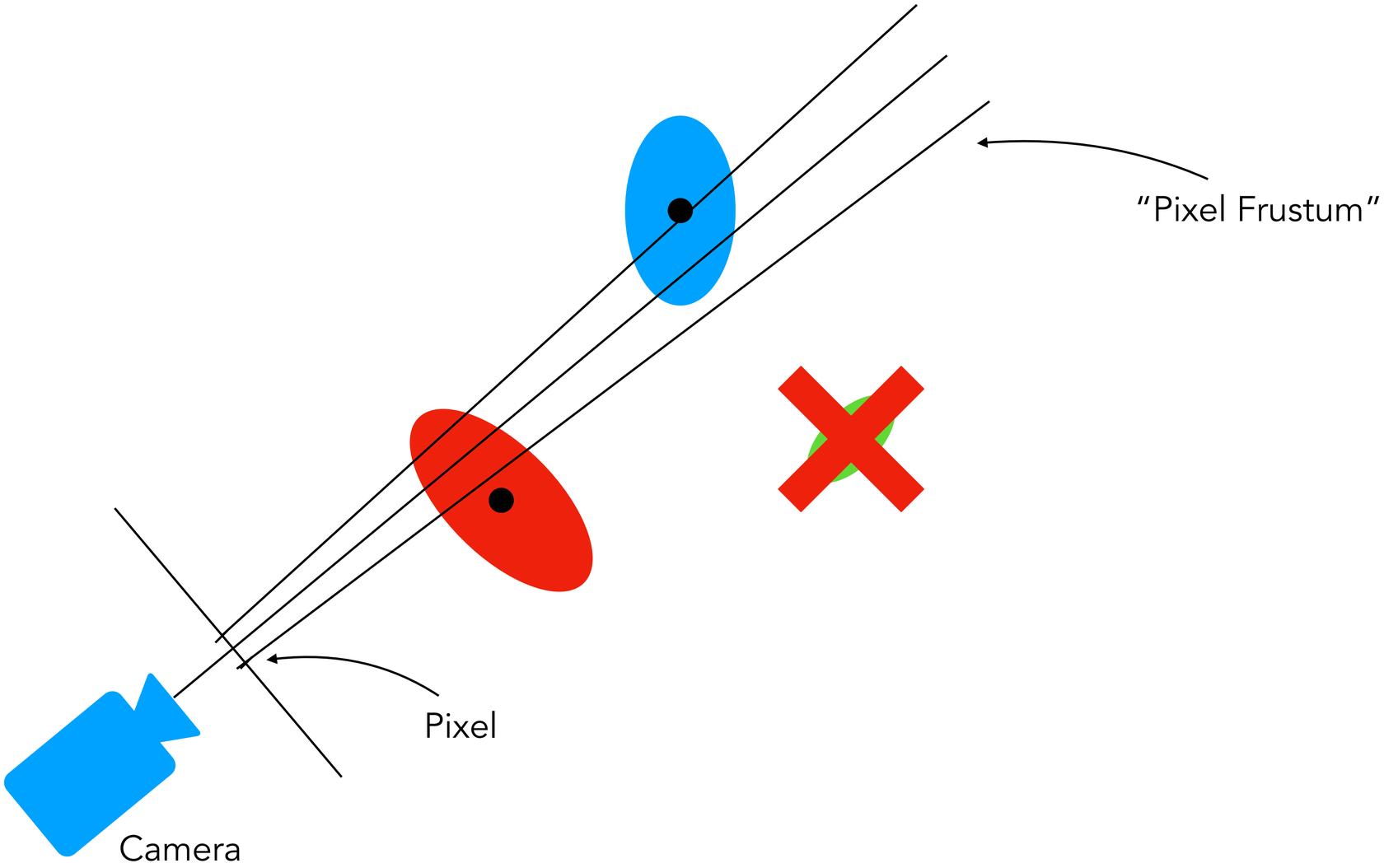
Instead: Rasterization



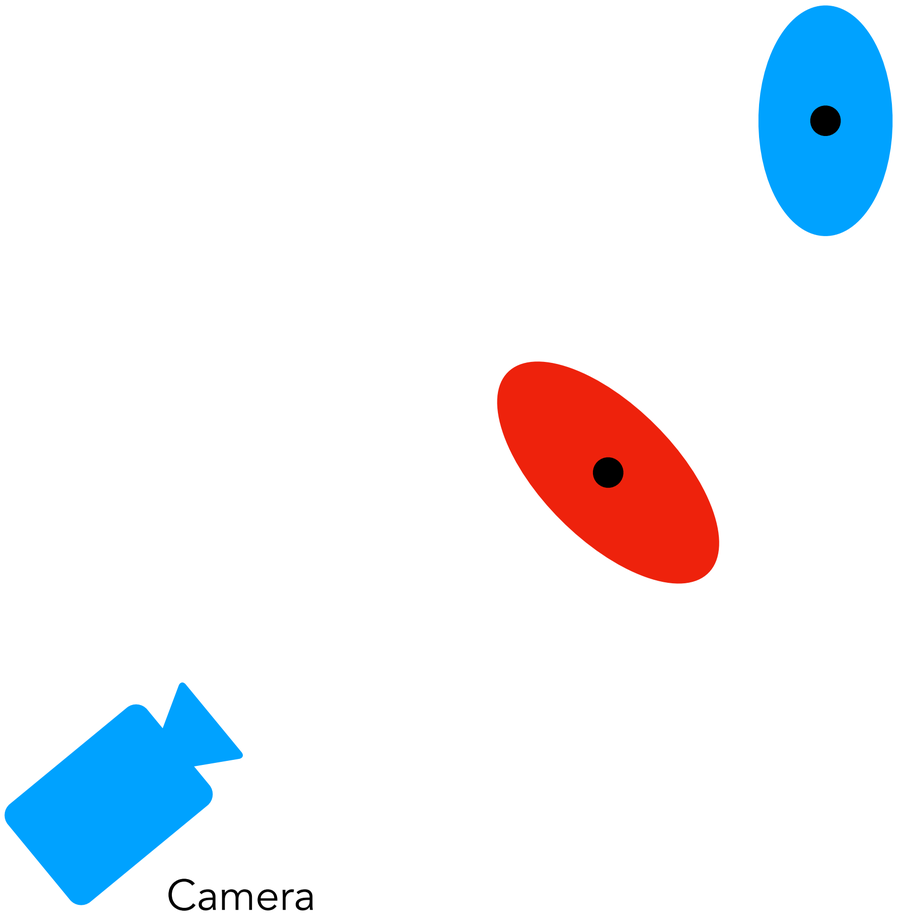
Instead: Rasterization



“Cull” Gaussians with less than 99% confidence relative to view frustum



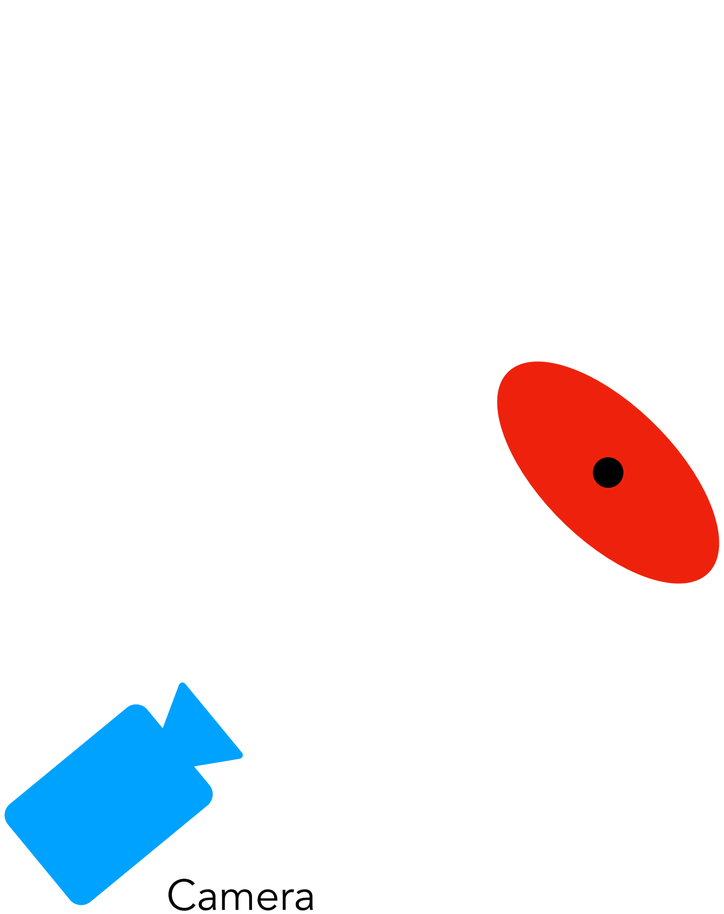
Step 1: Transform Gaussians into Camera Coordinates



Cam2world is affine mapping $\phi(x) = \mathbf{W}\mathbf{x} + \mathbf{p}$:

$$\mathcal{G}_{\mathbf{V}_k''}(\varphi^{-1}(\mathbf{u}) - \mathbf{t}_k) = \frac{1}{|\mathbf{W}^{-1}|}\mathcal{G}_{\mathbf{V}_k'}(\mathbf{u} - \mathbf{u}_k) = r_k'(\mathbf{u})$$

Step 1: Transform Gaussians into Camera Coordinates



Cam2world is affine mapping $\phi(x) = \mathbf{W}\mathbf{x} + \mathbf{p}$:

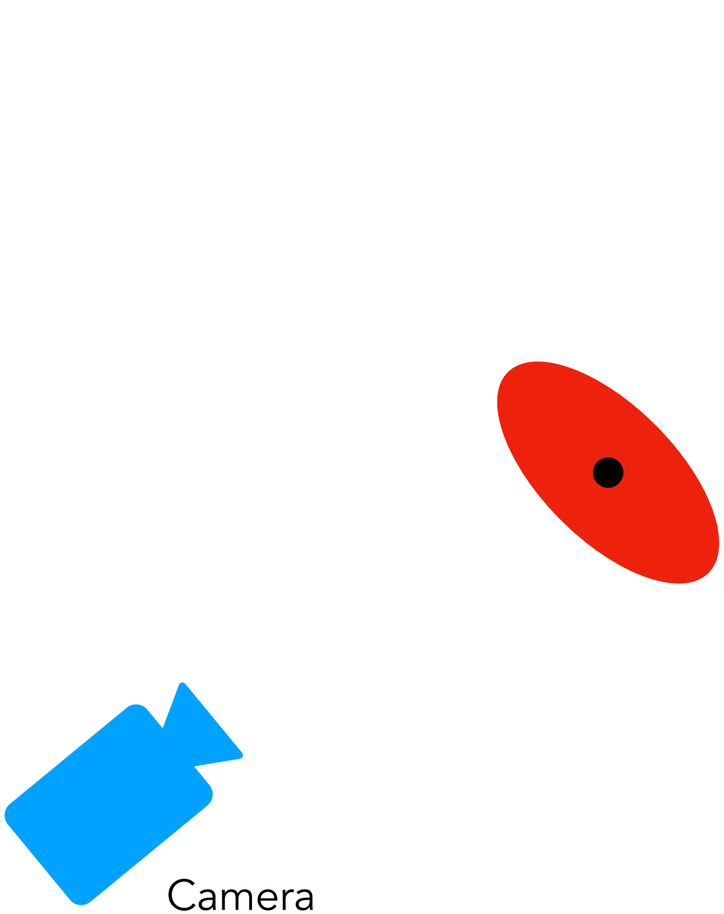
$$\mathcal{G}_{\mathbf{V}_k''}(\varphi^{-1}(\mathbf{u}) - \mathbf{t}_k) = \frac{1}{|\mathbf{W}^{-1}|}\mathcal{G}_{\mathbf{V}_k'}(\mathbf{u} - \mathbf{u}_k) = r_k'(\mathbf{u})$$

Projection $\mathbf{m}(u)$ is *not* an affine mapping :/

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \mathbf{m}(\mathbf{u}) = \begin{pmatrix} u_0/u_2 \\ u_1/u_2 \\ \|(u_0, u_1, u_2)^T\| \end{pmatrix}$$

$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} = \mathbf{m}^{-1}(\mathbf{x}) = \begin{pmatrix} x_0/l \cdot x_2 \\ x_1/l \cdot x_2 \\ 1/l \cdot x_2 \end{pmatrix},$$

Step 1: Transform Gaussians into Camera Coordinates



Cam2world is affine mapping $\phi(x) = \mathbf{W}\mathbf{x} + \mathbf{p}$:

$$\mathcal{G}_{\mathbf{V}_k''}(\varphi^{-1}(\mathbf{u}) - \mathbf{t}_k) = \frac{1}{|\mathbf{W}^{-1}|}\mathcal{G}_{\mathbf{V}_k'}(\mathbf{u} - \mathbf{u}_k) = r_k'(\mathbf{u})$$

Projection $\mathbf{m}(u)$ is *not* an affine mapping :/

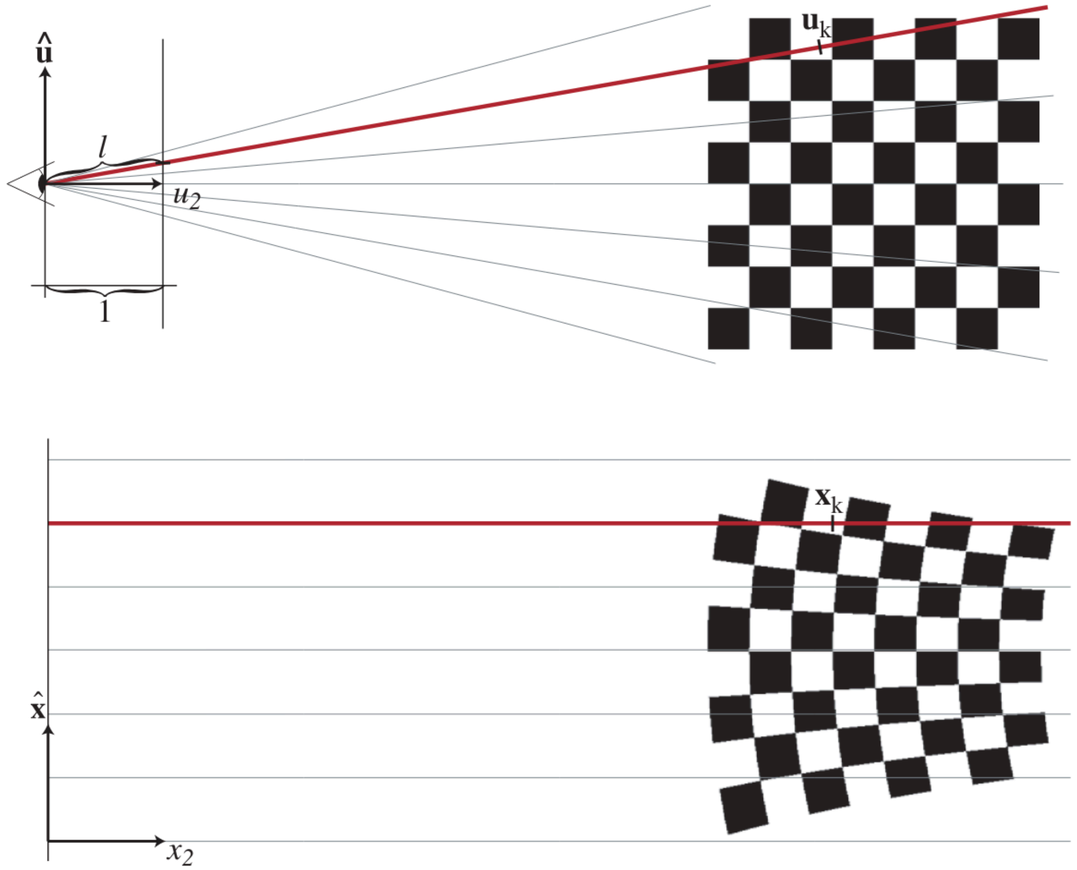
$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \mathbf{m}(\mathbf{u}) = \begin{pmatrix} u_0/u_2 \\ u_1/u_2 \\ \|(u_0, u_1, u_2)^T\| \end{pmatrix}$$

$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} = \mathbf{m}^{-1}(\mathbf{x}) = \begin{pmatrix} x_0/l \cdot x_2 \\ x_1/l \cdot x_2 \\ 1/l \cdot x_2 \end{pmatrix},$$

But can approximate with first-order Taylor Expansion as:

$$\mathbf{m}_{\mathbf{u}_k}(\mathbf{u}) = \mathbf{x}_k + \mathbf{J}_{\mathbf{u}_k} \cdot (\mathbf{u} - \mathbf{u}_k) \qquad \mathbf{J}_{\mathbf{u}_k} = \frac{\partial \mathbf{m}}{\partial \mathbf{u}}(\mathbf{u}_k)$$

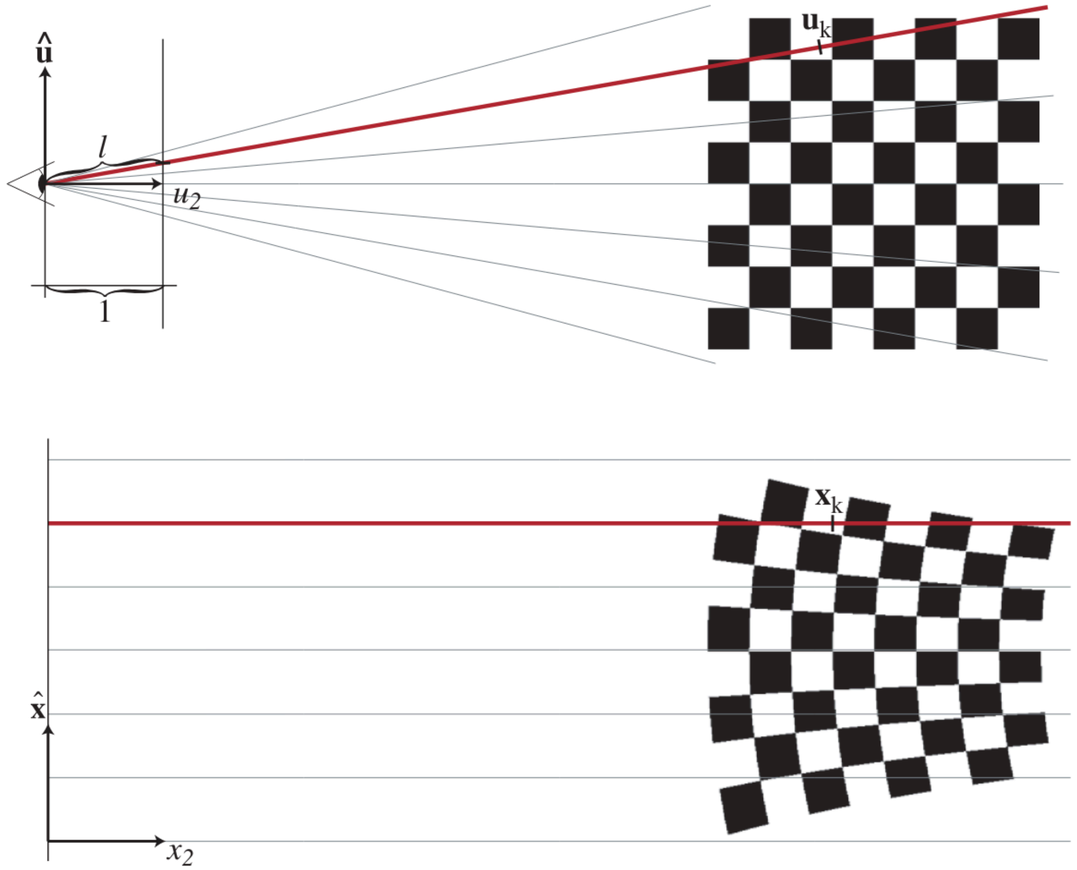
Step 1: Transform Gaussians into Camera Coordinates



But can approximate with first-order Taylor Expansion as:

$$\mathbf{m}_{\mathbf{u}_k}(\mathbf{u}) = \mathbf{x}_k + \mathbf{J}_{\mathbf{u}_k} \cdot (\mathbf{u} - \mathbf{u}_k) \qquad \mathbf{J}_{\mathbf{u}_k} = \frac{\partial \mathbf{m}}{\partial \mathbf{u}}(\mathbf{u}_k)$$

Step 1: Transform Gaussians into Camera Coordinates



But can approximate with first-order Taylor Expansion as:

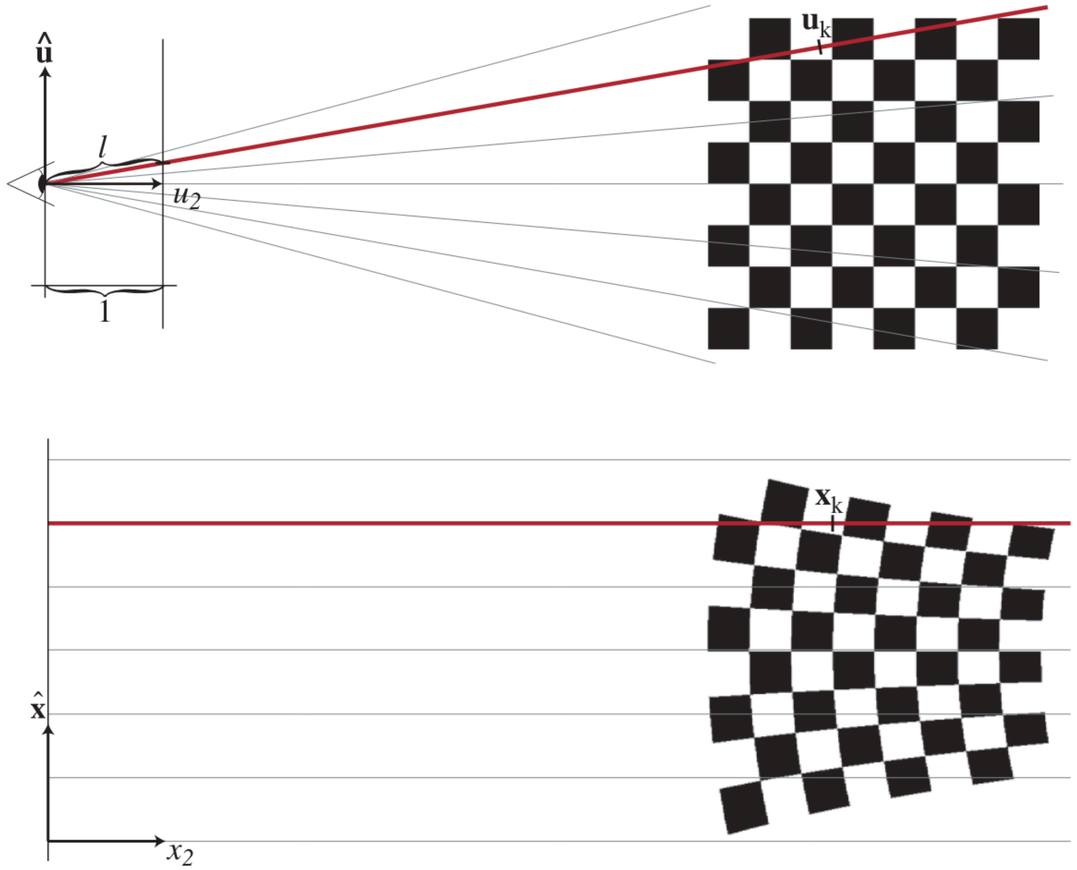
$$\mathbf{m}_{\mathbf{u}_k}(\mathbf{u}) = \mathbf{x}_k + \mathbf{J}_{\mathbf{u}_k} \cdot (\mathbf{u} - \mathbf{u}_k) \qquad \mathbf{J}_{\mathbf{u}_k} = \frac{\partial \mathbf{m}}{\partial \mathbf{u}}(\mathbf{u}_k)$$

Projected, 2D Gaussians are then:

$$\frac{1}{|\mathbf{W}^{-1}||\mathbf{J}^{-1}|}\mathcal{G}_{\mathbf{V}_k}(\mathbf{x} - \mathbf{x}_k)$$

$$\begin{aligned} \mathbf{V}_k &= \mathbf{J}\mathbf{V}'_k\,\mathbf{J}^T \\ &= \mathbf{J}\mathbf{W}\mathbf{V}''_k\mathbf{W}^T\mathbf{J}^T. \end{aligned}$$

Step 1: Transform Gaussians into Camera Coordinates



But can approximate with first-order Taylor Expansion as:

$$\mathbf{m}_{\mathbf{u}_k}(\mathbf{u}) = \mathbf{x}_k + \mathbf{J}_{\mathbf{u}_k} \cdot (\mathbf{u} - \mathbf{u}_k) \qquad \mathbf{J}_{\mathbf{u}_k} = \frac{\partial \mathbf{m}}{\partial \mathbf{u}}(\mathbf{u}_k)$$

Projected, 2D Gaussians are then:

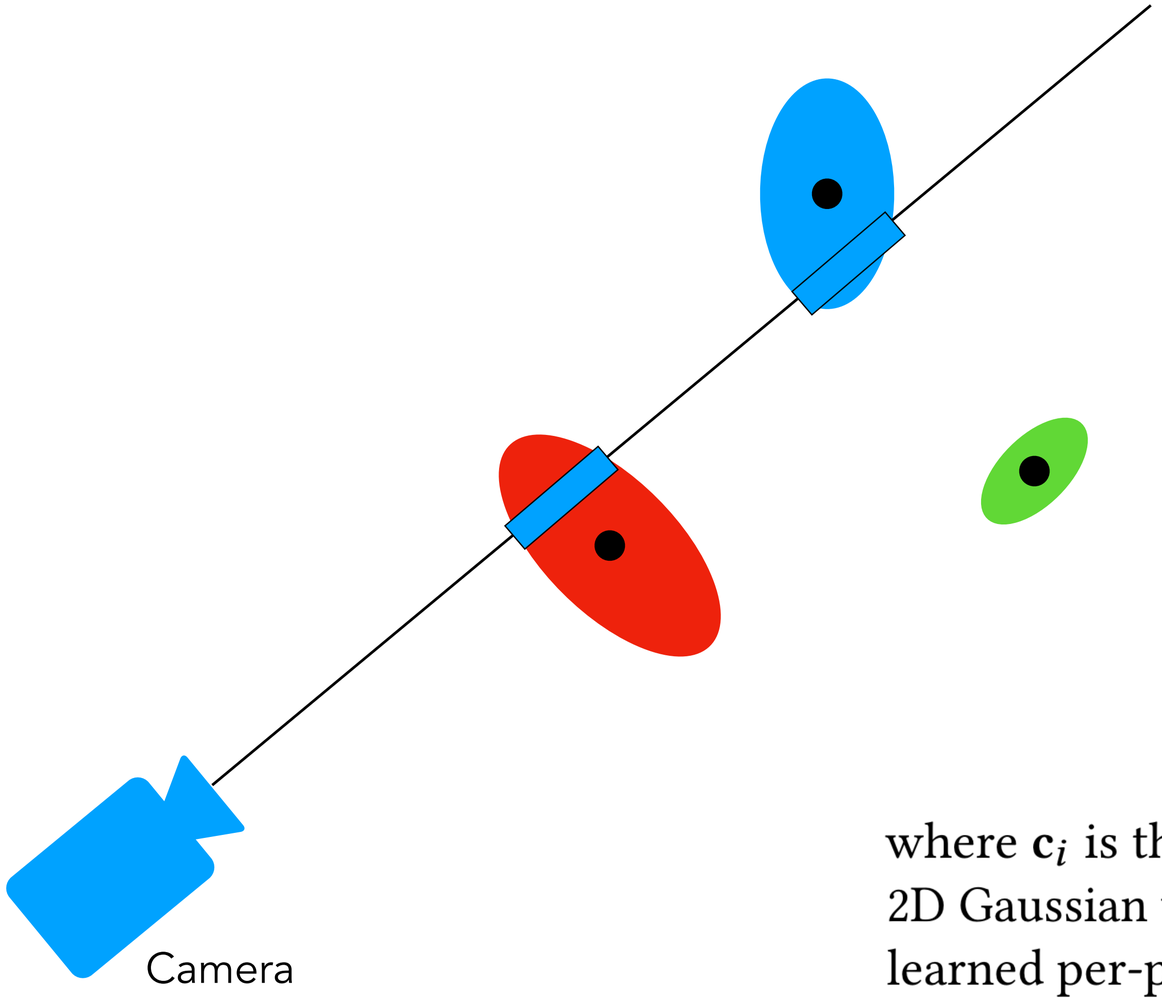
$$\frac{1}{|\mathbf{W}^{-1}||\mathbf{J}^{-1}|}\mathcal{G}_{\mathbf{V}_k}(\mathbf{x} - \mathbf{x}_k)$$

$$\begin{aligned} \mathbf{V}_k &= \mathbf{J}\mathbf{V}'_k\,\mathbf{J}^T \\ &= \mathbf{J}\mathbf{W}\mathbf{V}''_k\mathbf{W}^T\mathbf{J}^T. \end{aligned}$$

Finally, can integrate along rays:

$$\begin{aligned} q_k(\hat{\mathbf{x}}) &= \int_{\mathbb{R}} \frac{1}{|\mathbf{J}^{-1}||\mathbf{W}^{-1}|}\mathcal{G}_{\mathbf{V}_k}(\hat{\mathbf{x}} - \hat{\mathbf{x}}_k, x_2 - x_{k2})\,dx_2 \\ &= \frac{1}{|\mathbf{J}^{-1}||\mathbf{W}^{-1}|}\mathcal{G}_{\hat{\mathbf{V}}_k}(\hat{\mathbf{x}} - \hat{\mathbf{x}}_k) \end{aligned}$$

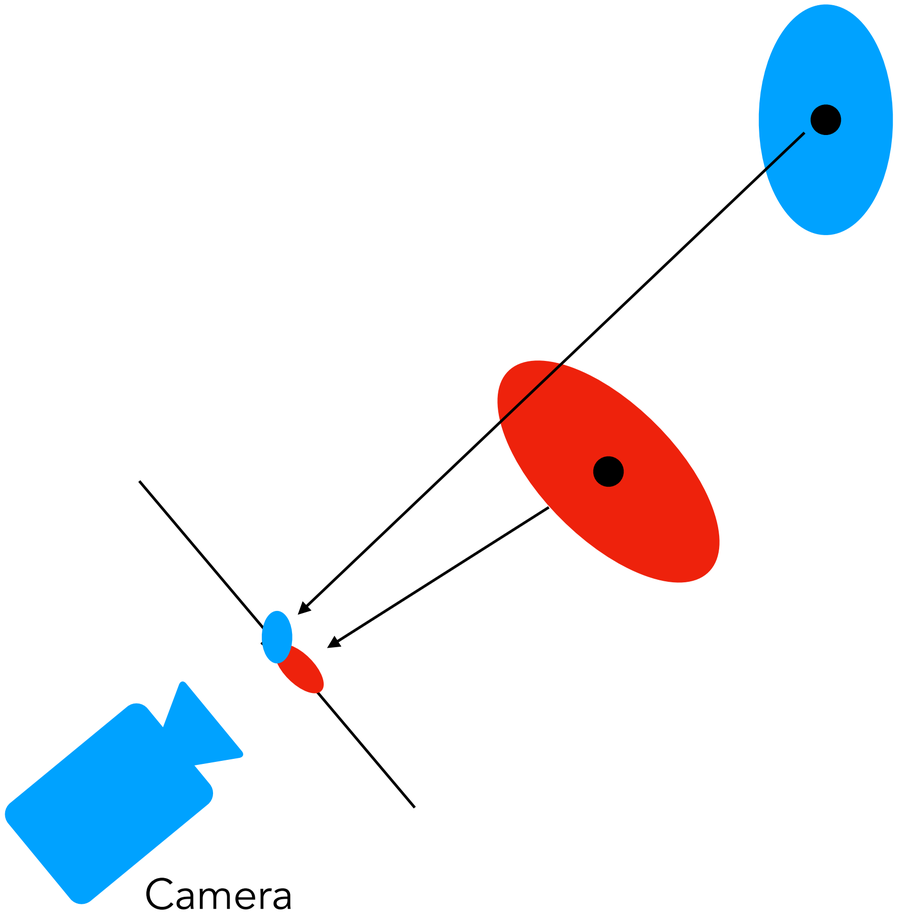
Can compute volume rendering integral without ever sampling a single 3D point in space!

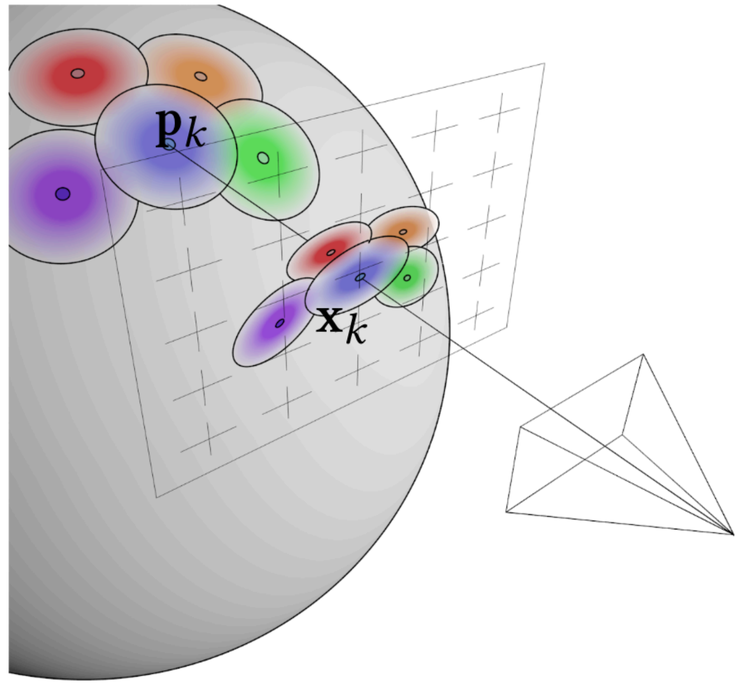


$$C = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \tag{3}$$

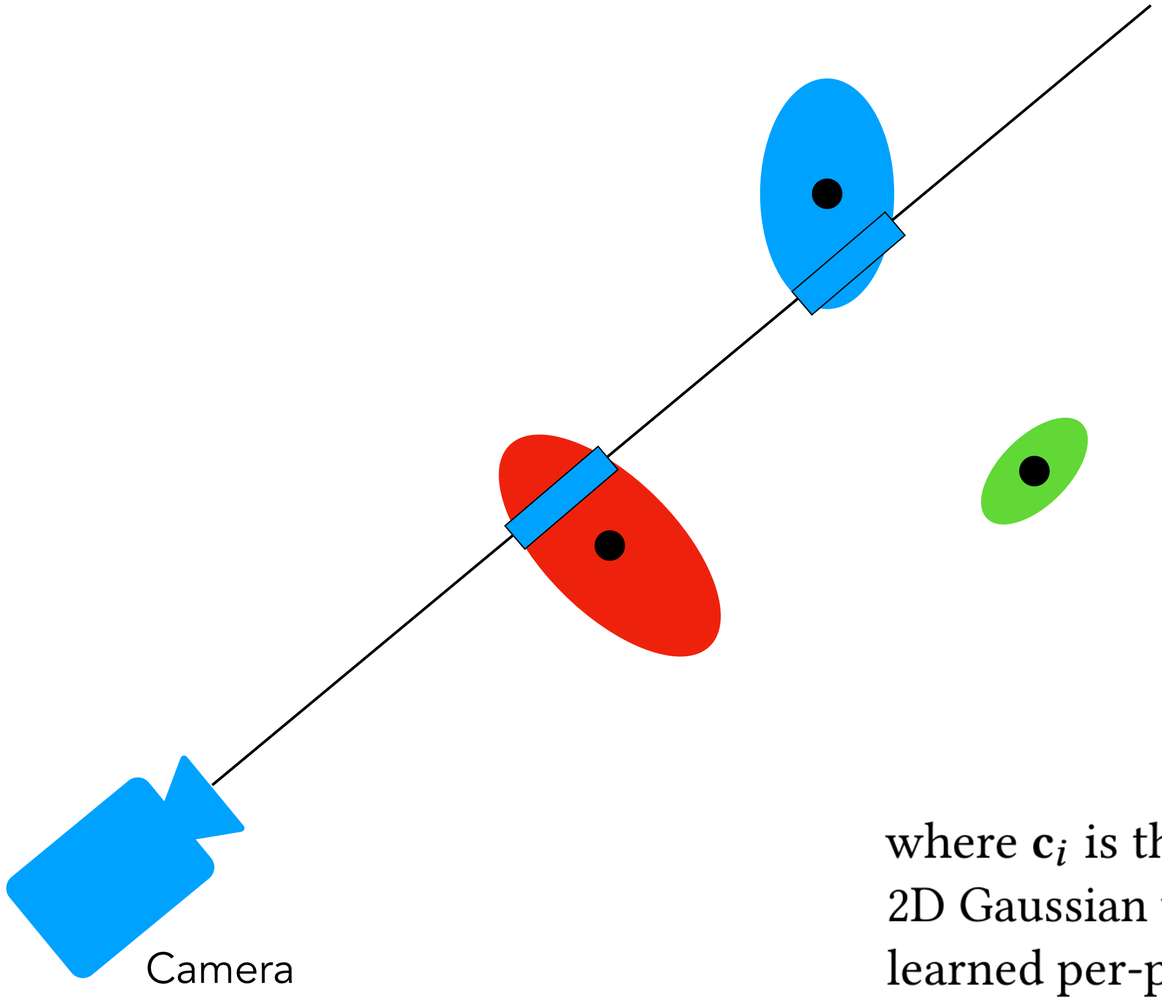
where $\mathbf{c}_i$ is the color of each point and α_i is given by evaluating a 2D Gaussian with covariance Σ [Yifan et al. 2019] multiplied with a learned per-point opacity.

Projected 3D Gaussian makes 2D Gaussian!





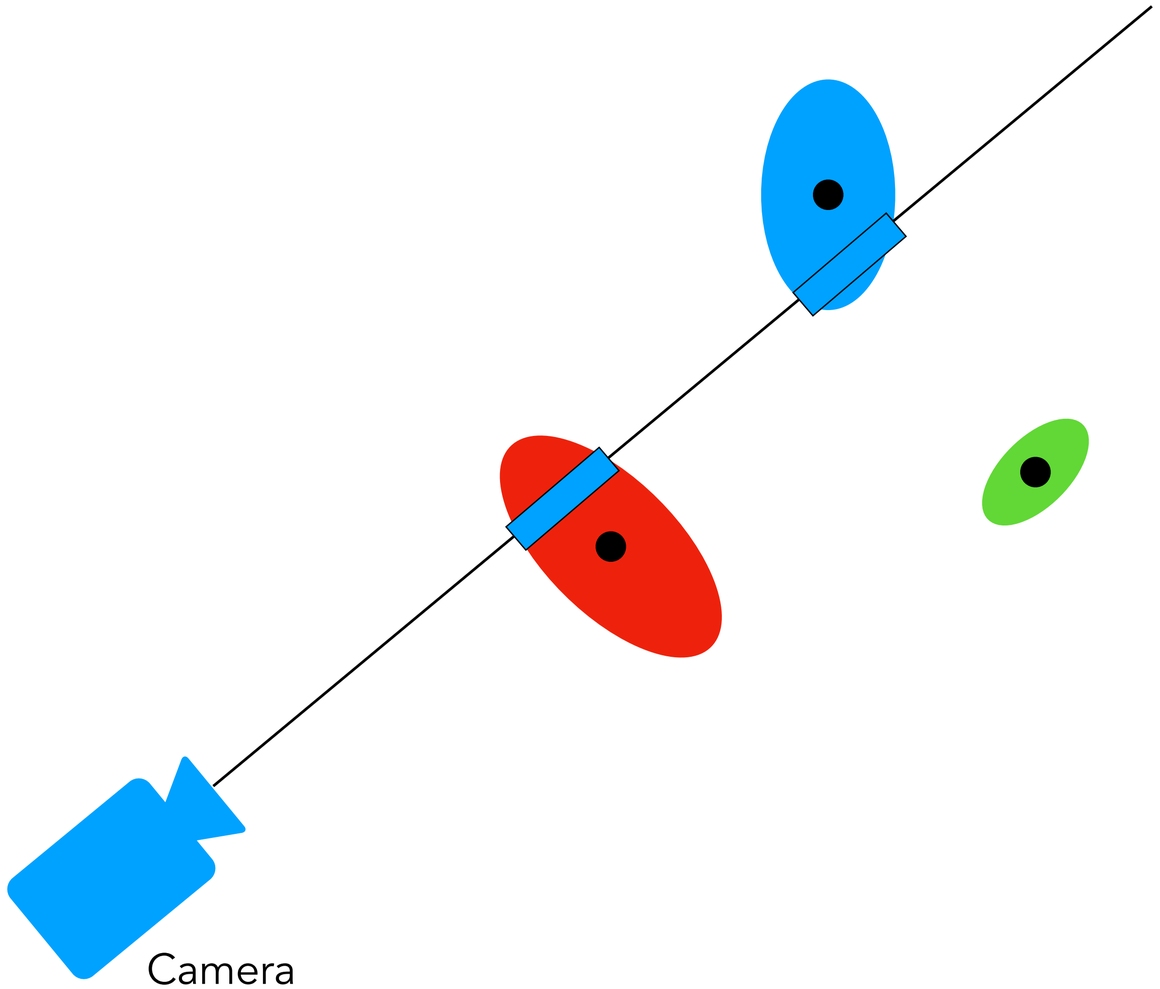
Any problems for inverse graphics, though…?



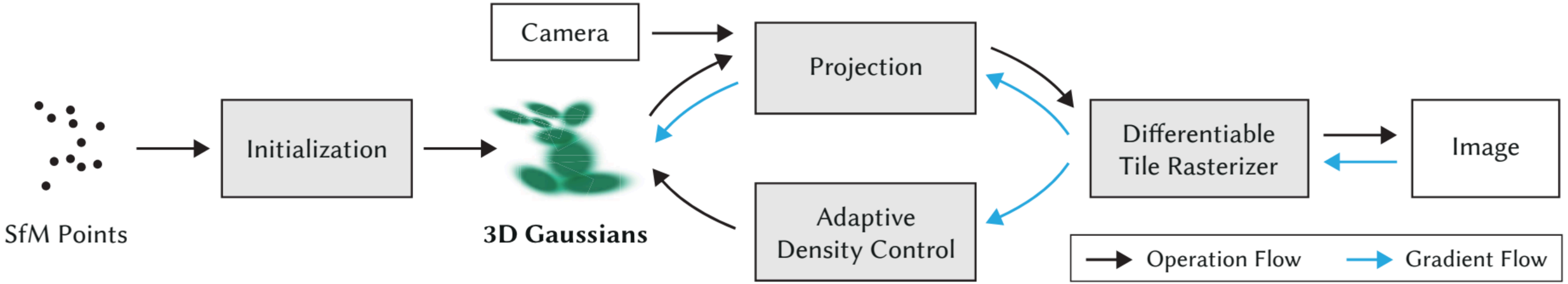
$$C = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \tag{3}$$

where $\mathbf{c}_i$ is the color of each point and α_i is given by evaluating a 2D Gaussian with covariance Σ [Yifan et al. 2019] multiplied with a learned per-point opacity.

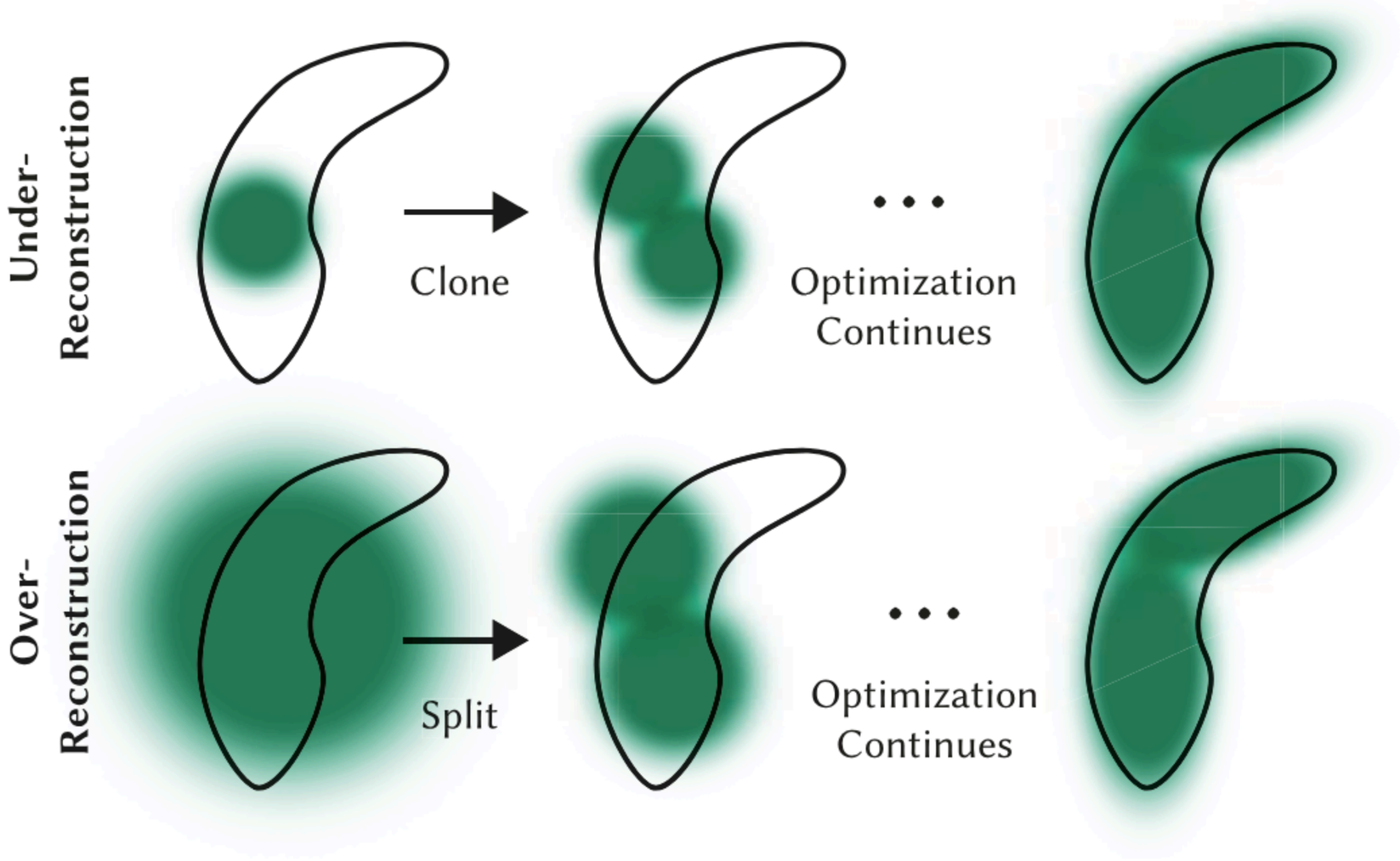
Problem: Local minima…



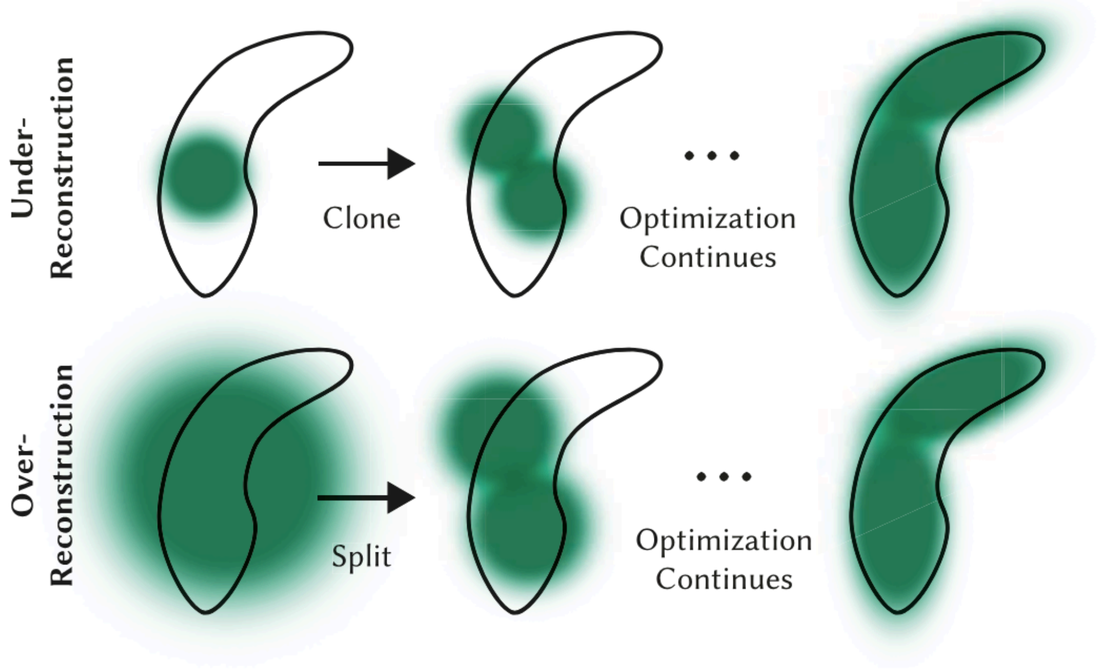
Fix 1: Start from SFM point cloud.



Fix 2: Heuristic *pruning* and *spawning* operations

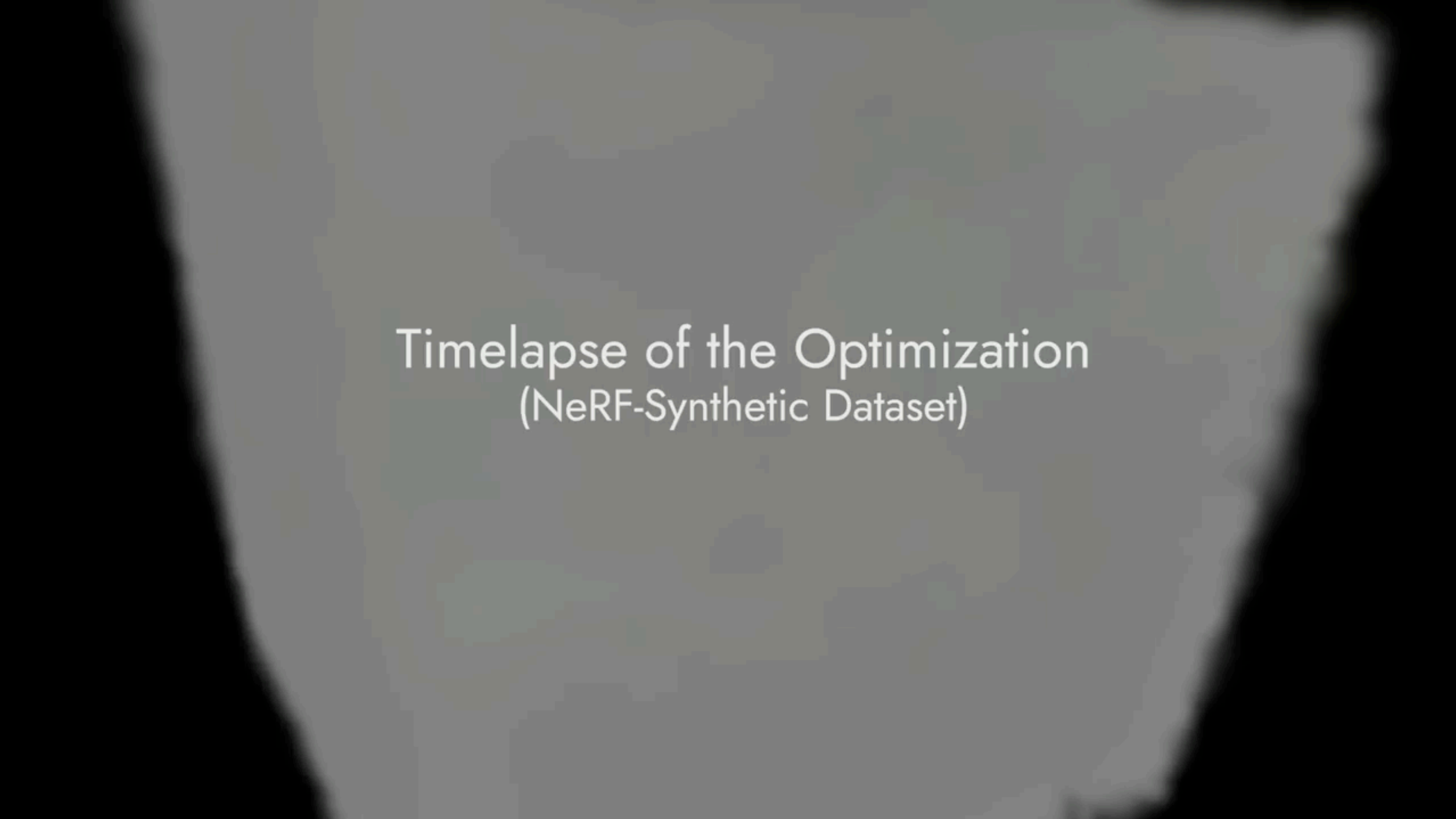


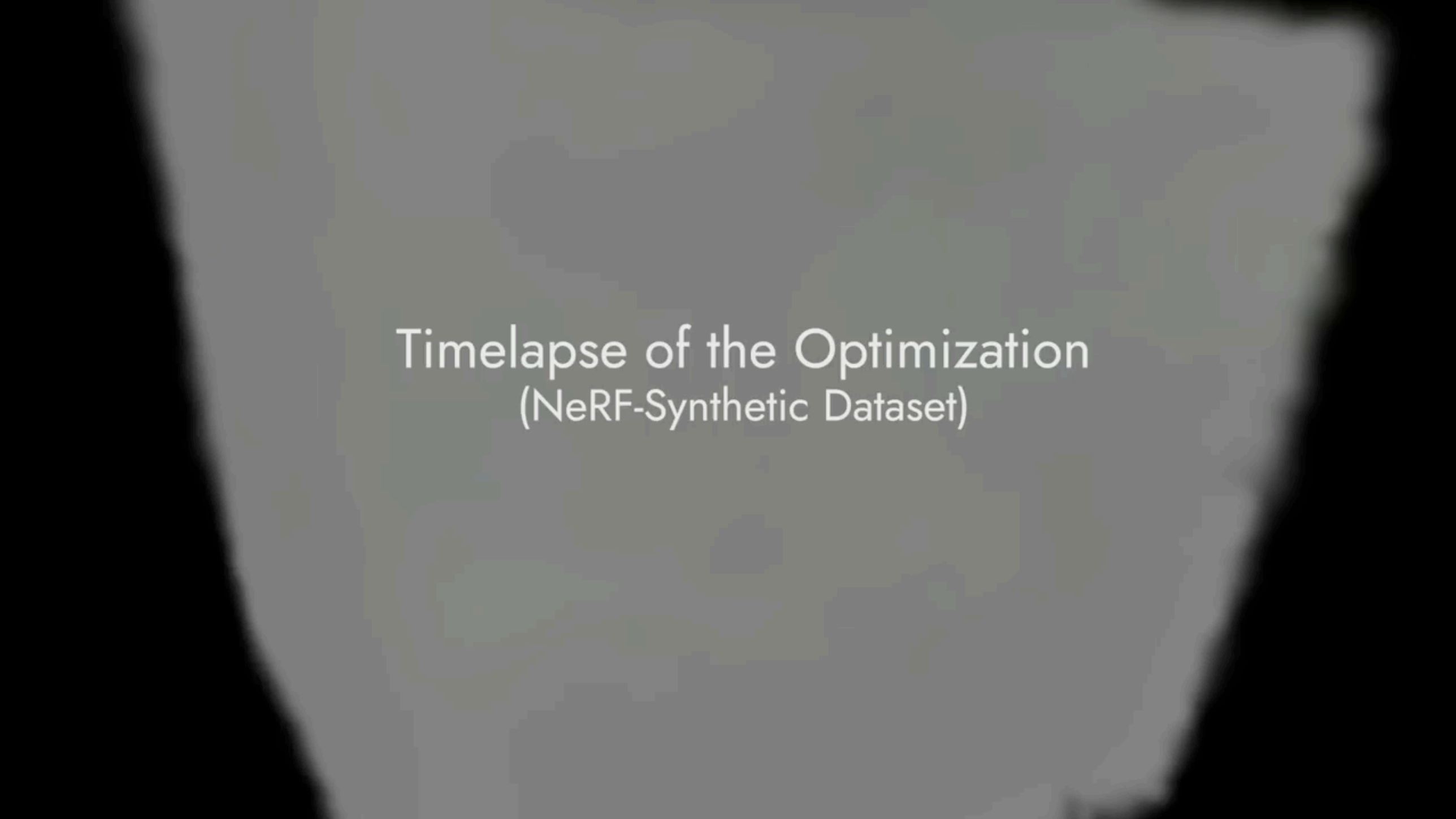
Fix 2: Heuristic *pruning* and *spawning* operations



Our adaptive control of the Gaussians needs to populate empty areas. It focuses on regions with missing geometric features ("under-reconstruction"), but also in regions where Gaussians cover large areas in the scene (which often correspond to "over-reconstruction"). We observe that both have *large* view-space positional gradients. Intuitively, this is likely because they correspond to regions that are not yet well reconstructed, and the optimization tries to move the Gaussians to correct this.

Since both cases are good candidates for densification, we densify Gaussians with an average magnitude of view-space position gradients above a threshold τ_{pos}, which we set to 0.0002 in our tests.

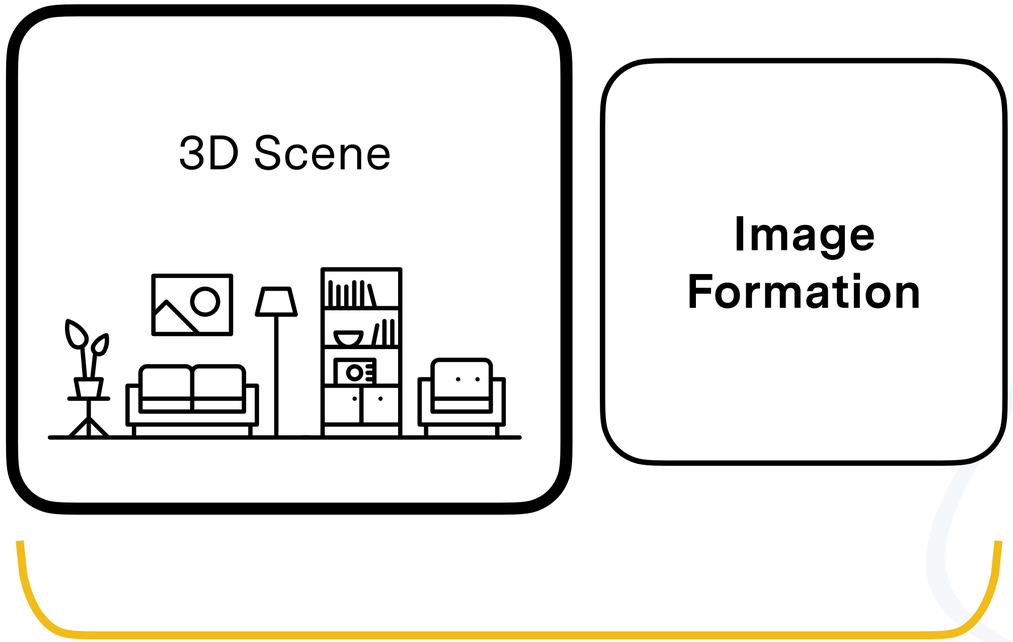

Timelapse of the Optimization
(NeRF-Synthetic Dataset)


Timelapse of the Optimization
(NeRF-Synthetic Dataset)


Menu Views Capture
3D Gaussians
Camera Point view
All interactive sessions are recorded at 1080p with an A6000
Metrics
92.95 (10.76 ms)
Playroom


Menu Views Capture
3D Gaussians
Camera Point view
All interactive sessions are recorded at 1080p with an A6000
Metrics
92.95 (10.76 ms)
Playroom

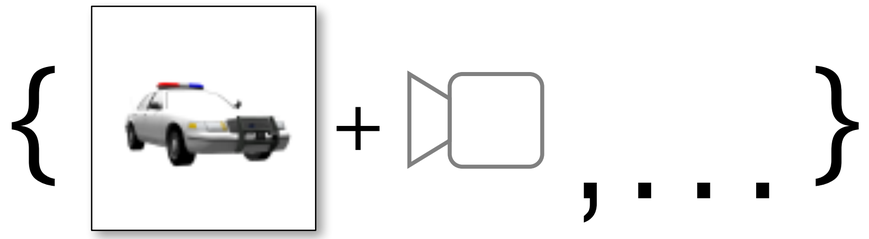
Today: Differentiable Rendering, AKA “Inverse Graphics”



Overfitting case: Inference = Fitting via Gradient Descent

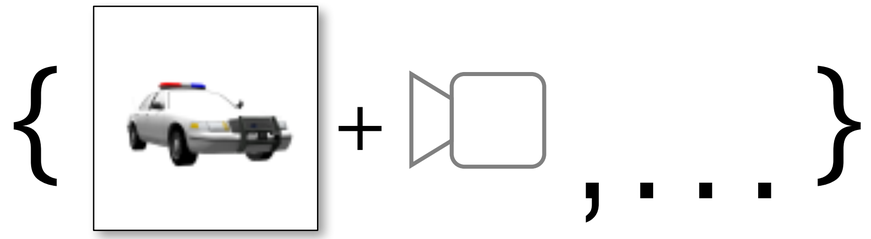
Sitzmann et al: Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations, NeurIPS 2020.

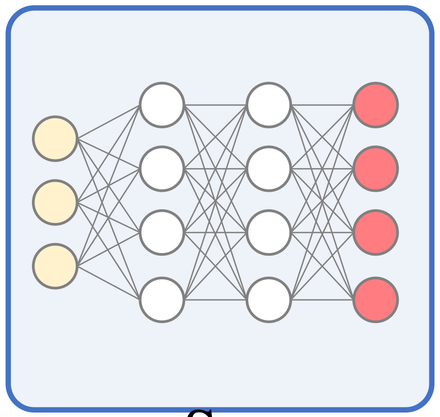
Overfitting case: Inference = Fitting via Gradient Descent



Sitzmann et al: Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations, NeurIPS 2020.

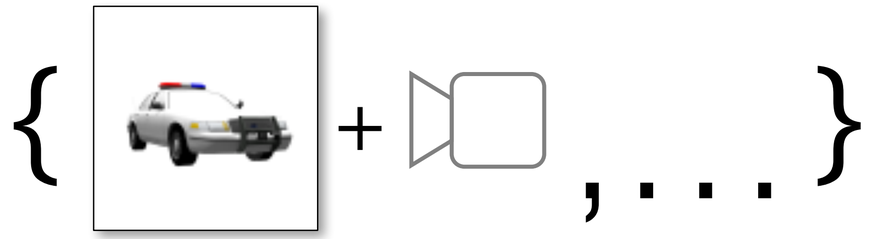
Overfitting case: Inference = Fitting via Gradient Descent

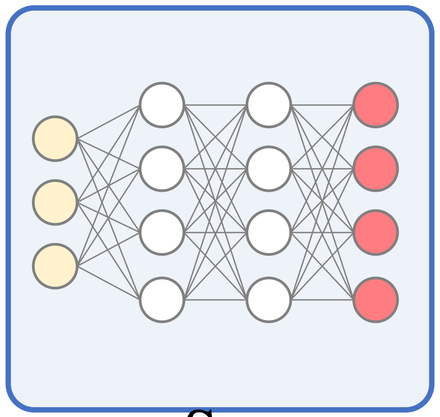




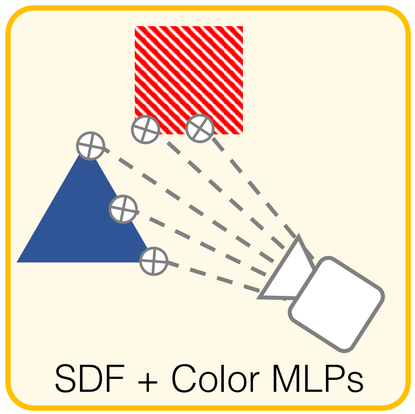
Sitzmann et al: Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations, NeurIPS 2020.

Overfitting case: Inference = Fitting via Gradient Descent





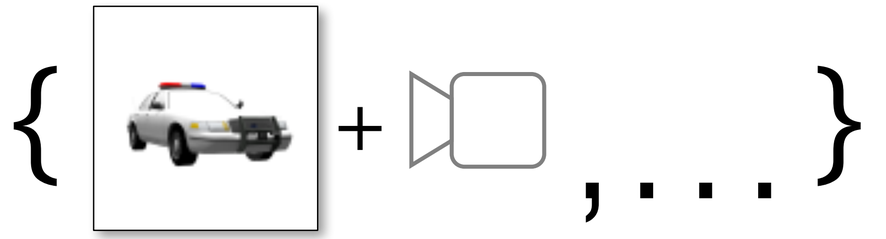
Srn_{ϕ}

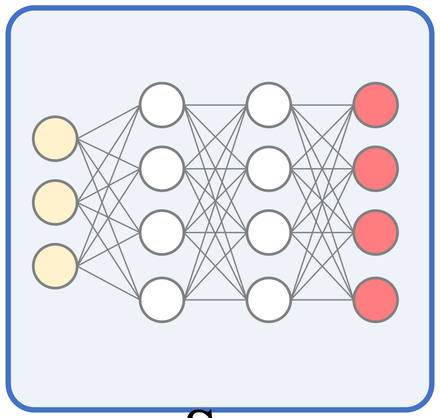


Render_{θ}

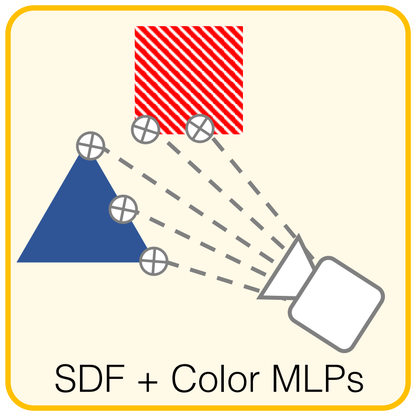
Sitzmann et al: Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations, NeurIPS 2020.

Overfitting case: Inference = Fitting via Gradient Descent





Srn_{ϕ}

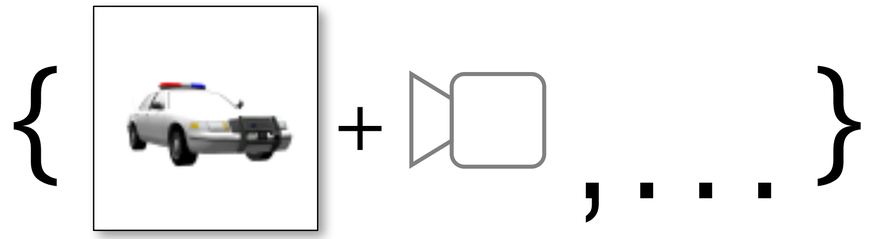


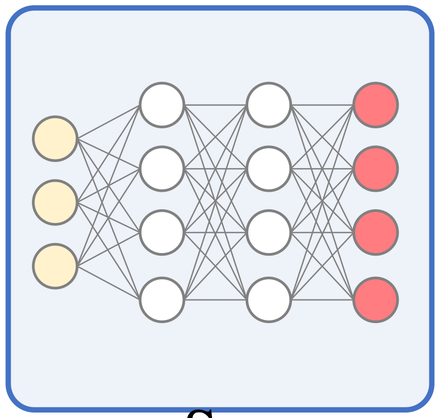
Render_{θ}

$$\min \left\| \text{Render}_{\theta}(\text{Srn}_{\phi}, \xi_i) - \mathcal{I}_i \right\|$$

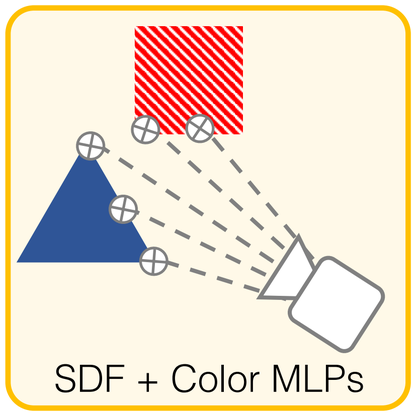
Sitzmann et al: Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations, NeurIPS 2020.

Overfitting case: Inference = Fitting via Gradient Descent





Srn_{ϕ}



Render_{θ}

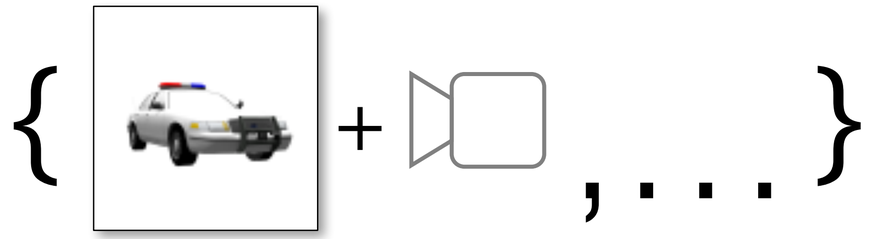
$$\min \left\| \text{Render}_{\theta}(\text{Srn}_{\phi}, \xi_i) - \mathcal{I}_i \right\|$$

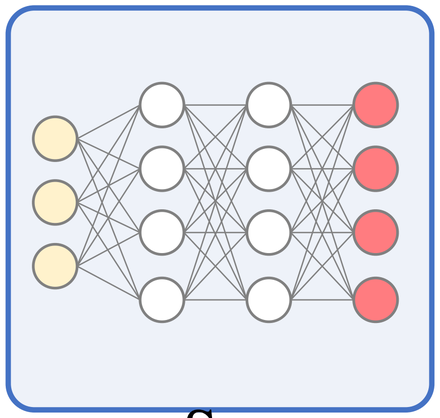
Optimization



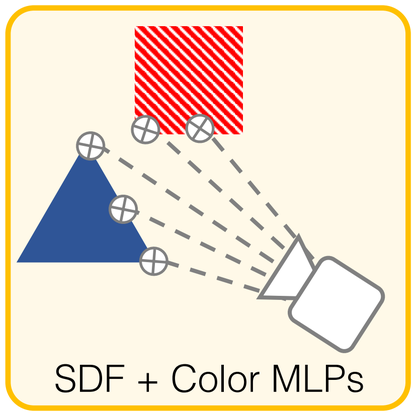
Sitzmann et al: Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations, NeurIPS 2020.

Overfitting case: Inference = Fitting via Gradient Descent





Srn_{ϕ}



Render_{θ}

$$\min \left\| \text{Render}_{\theta}(\text{Srn}_{\phi}, \xi_i) - \mathcal{I}_i \right\|$$

Optimization



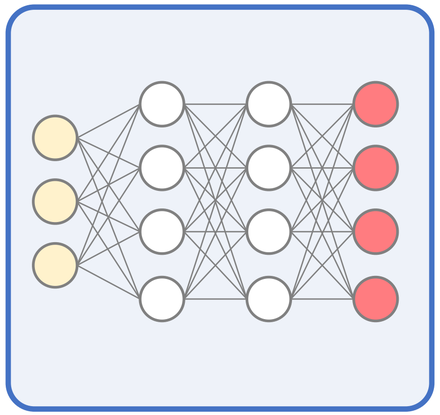
Novel View Synthesis



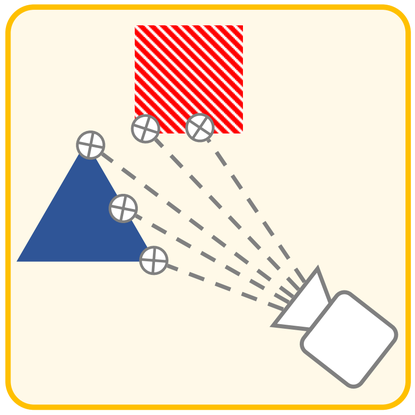
Normal map RGB

Sitzmann et al: Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations, NeurIPS 2020.

What if observations don't constrain scene representation?



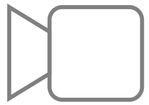
Srn_{ϕ}



Render_{θ}



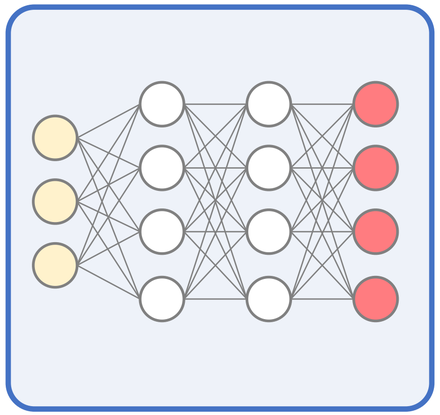
$+$



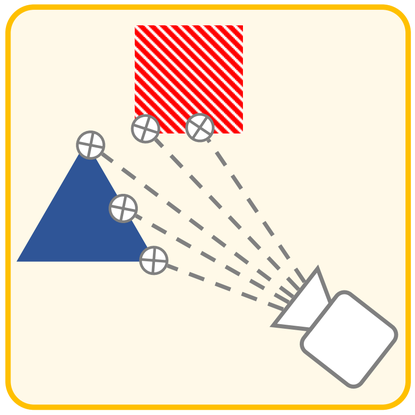
$(\mathcal{I}, \xi)$

$$\underset{\phi,\theta}{\text{argmin}} \left\| \text{Render}_{\theta}(\text{Srn}_{\phi}, \xi) - \mathcal{I} \right\|$$

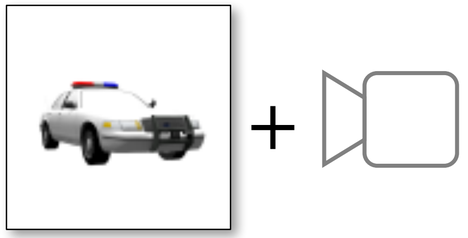
What if observations don't constrain scene representation?



Srn_{ϕ}



Render_{θ}



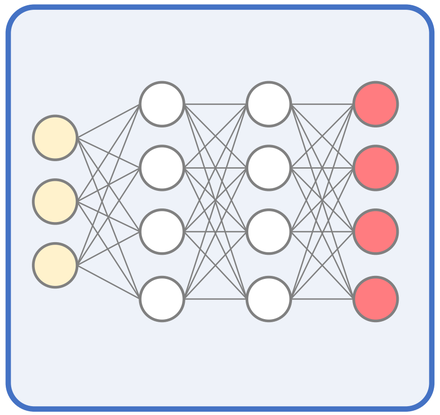
$+$

$(\mathcal{I}, \xi)$

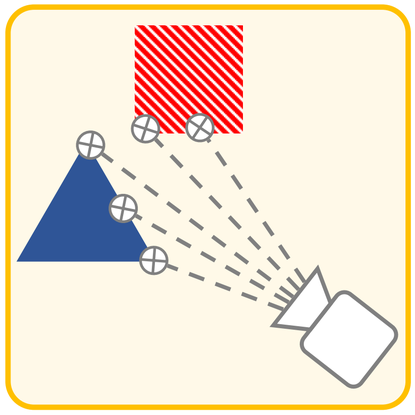
No 3D inform.

$$\underset{\phi,\theta}{\text{argmin}} \left\| \text{Render}_{\theta}(\text{Srn}_{\phi}, \xi) - \mathcal{I} \right\|$$

What if observations don't constrain scene representation?



Srn_{ϕ}



Render_{θ}



$+$

$(\mathcal{I}, \xi)$

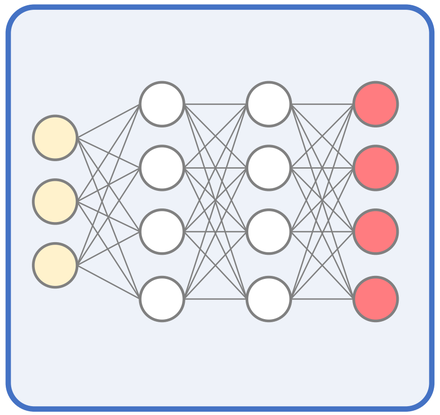
No 3D inform.



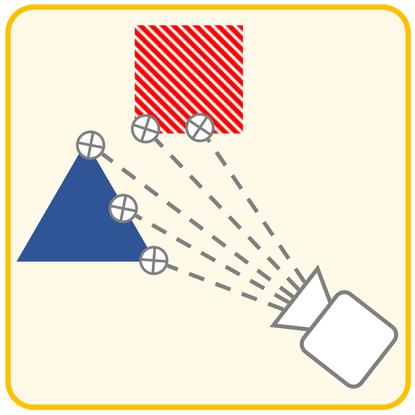
Input

$$\underset{\phi,\theta}{\text{argmin}} \left\| \text{Render}_{\theta}(\text{Srn}_{\phi}, \xi) - \mathcal{I} \right\|$$

What if observations don't constrain scene representation?



Srn_{ϕ}



Render_{θ}



$+$

$(\mathcal{I}, \xi)$

No 3D inform.

$$\underset{\phi,\theta}{\text{argmin}} \left\| \text{Render}_{\theta}(\text{Srn}_{\phi}, \xi) - \mathcal{I} \right\|$$



Input



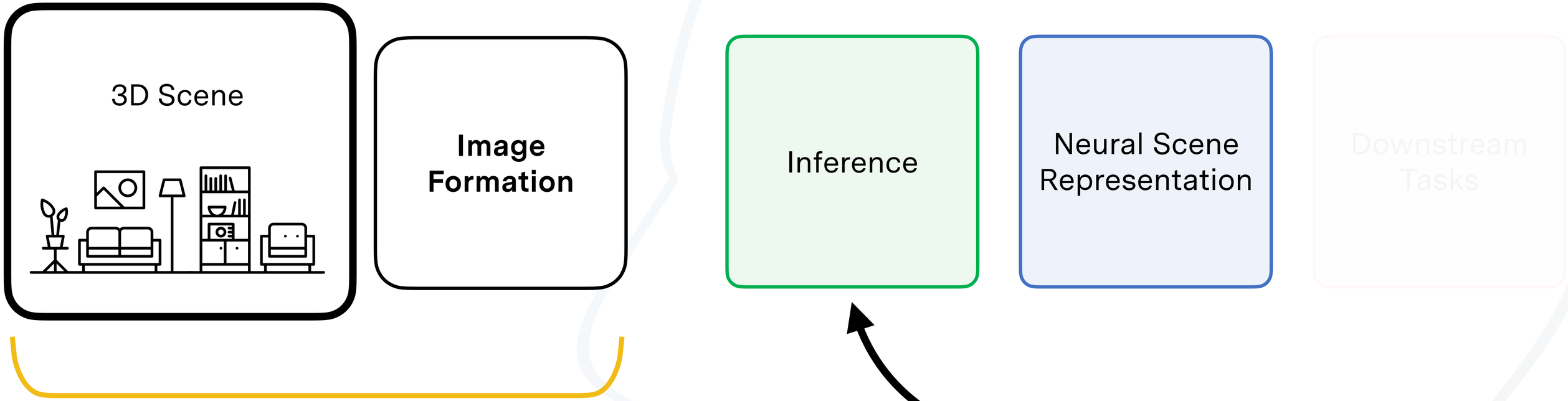
Normal map

RGB



GT

Today: Differentiable Rendering, AKA “Inverse Graphics”



Graphics

Let’s learn this!