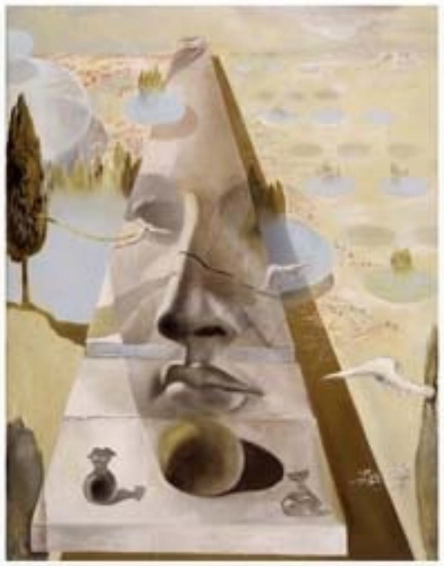
CS231A

Computer Vision: From 3D Reconstruction to Recognition

Gaussian Splatting for Novel View Synthesis



The problem of novel view synthesis

Inputs: sparsely sampled images of scene

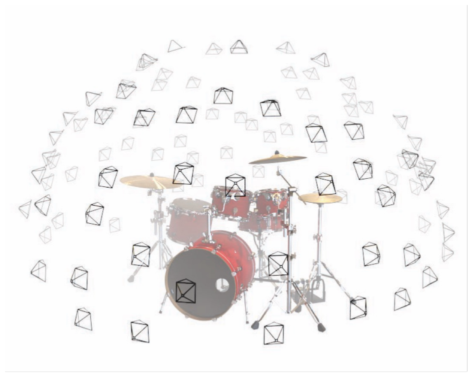
Outputs: *new* views of same scene
(rendered by our method)

2

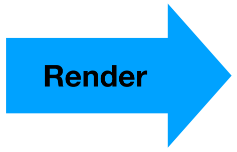
Mildenhall et al. ECCV 2020. https://www.matthewtancik.com/nerf

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

3D Scene



Camera Poses

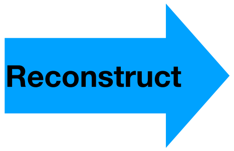


Images

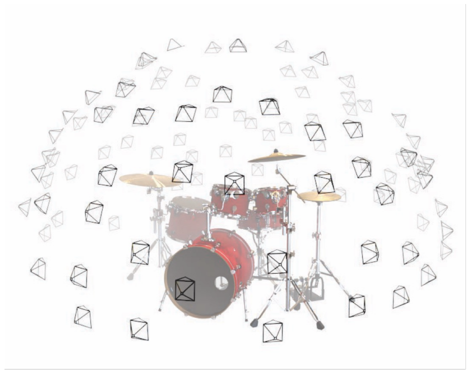
Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

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

Images



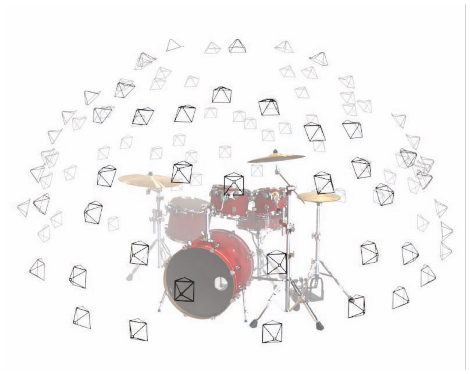
3D Scene



Camera Poses

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

How to get camera poses?



Images

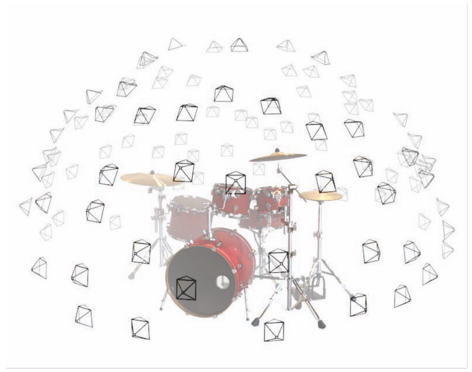
3D Scene

Camera Poses

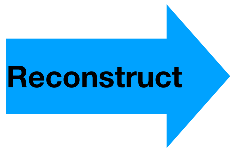
Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Can assume we know the camera poses.

Images



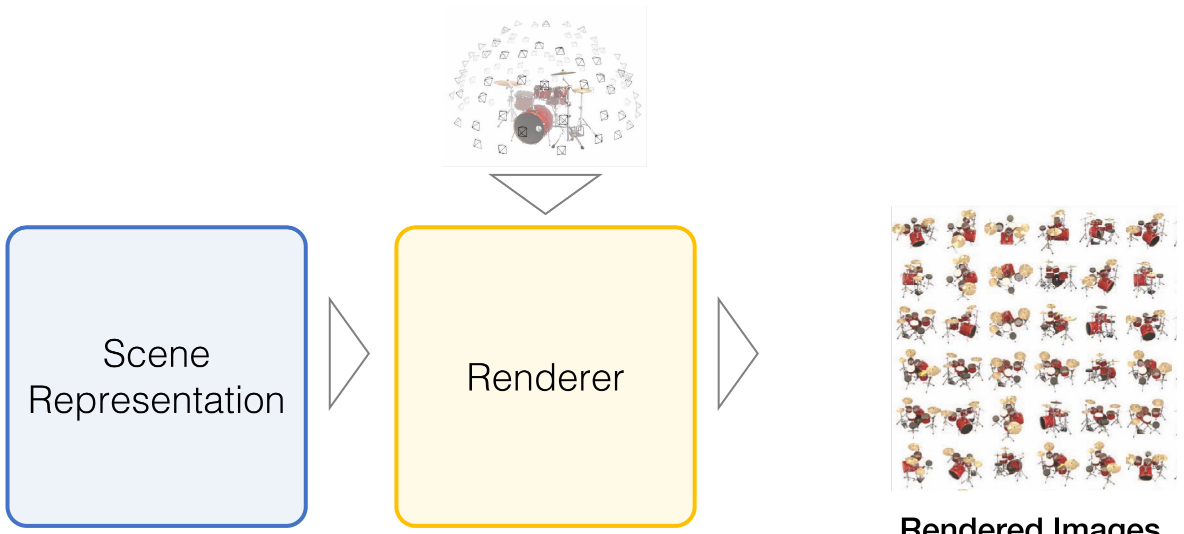
Camera Poses



3D Scene

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

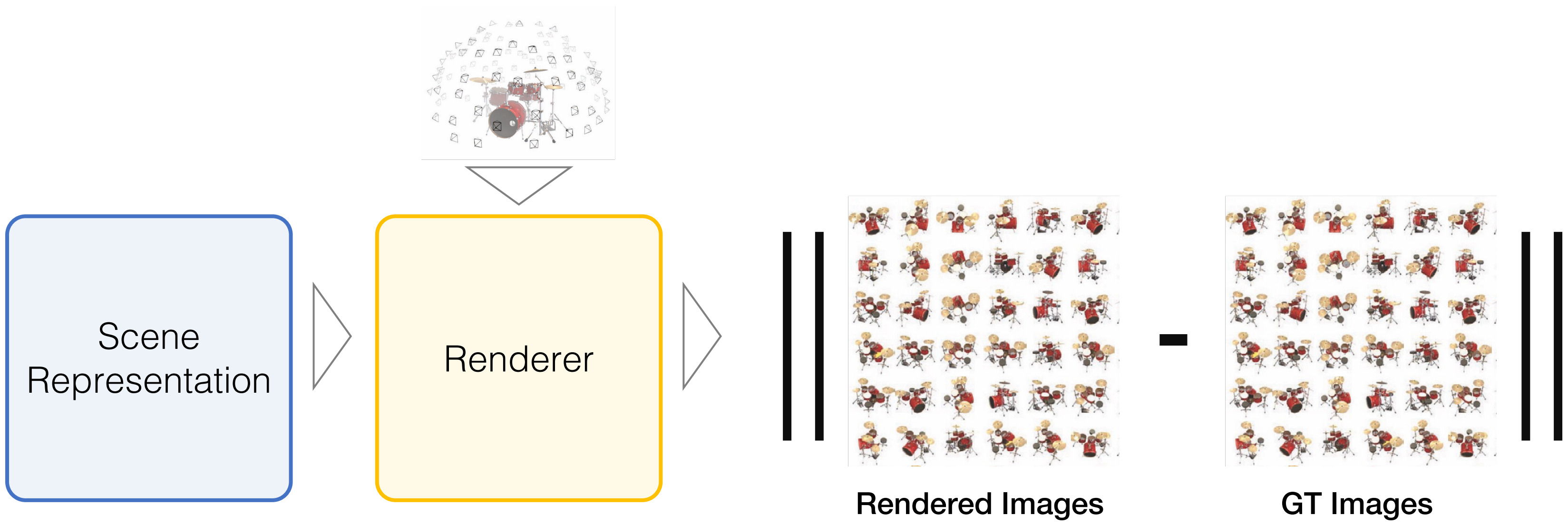
Differentiable Rendering



Rendered Images

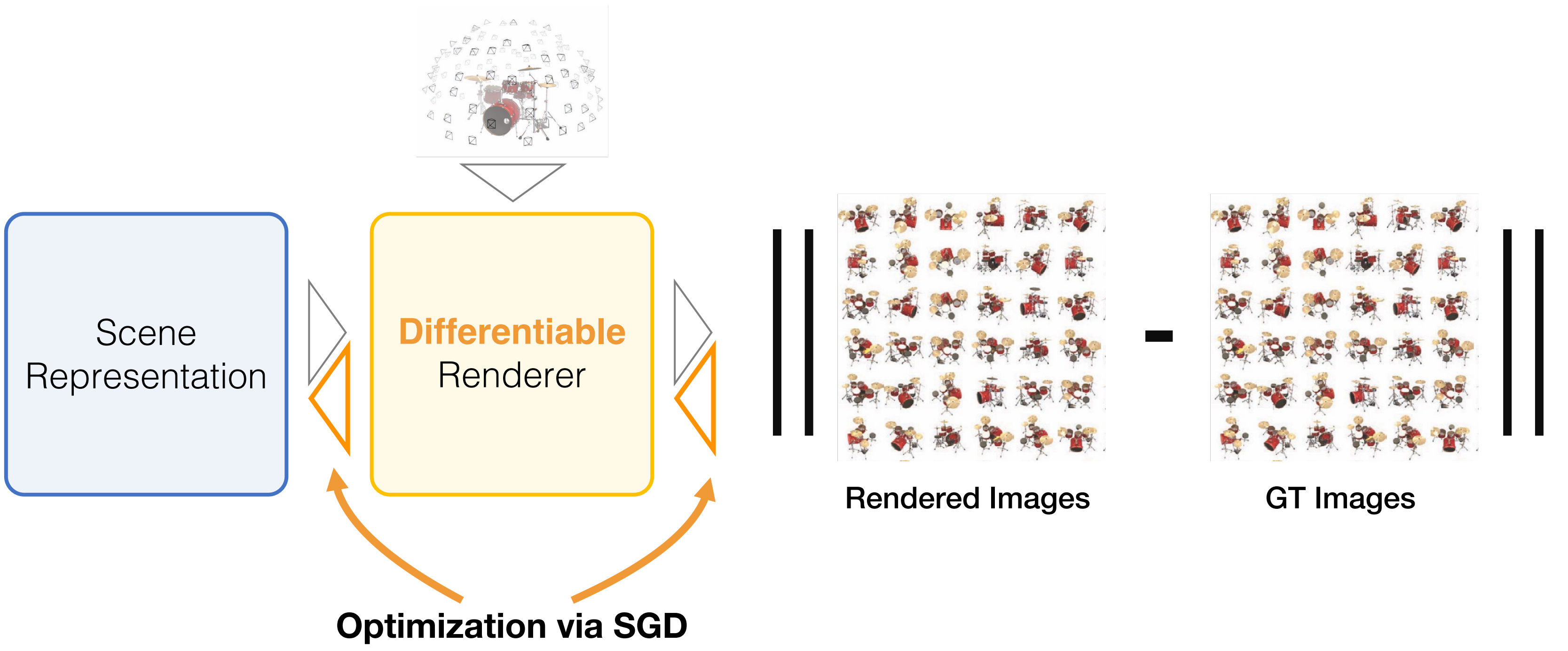
Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Differentiable Rendering



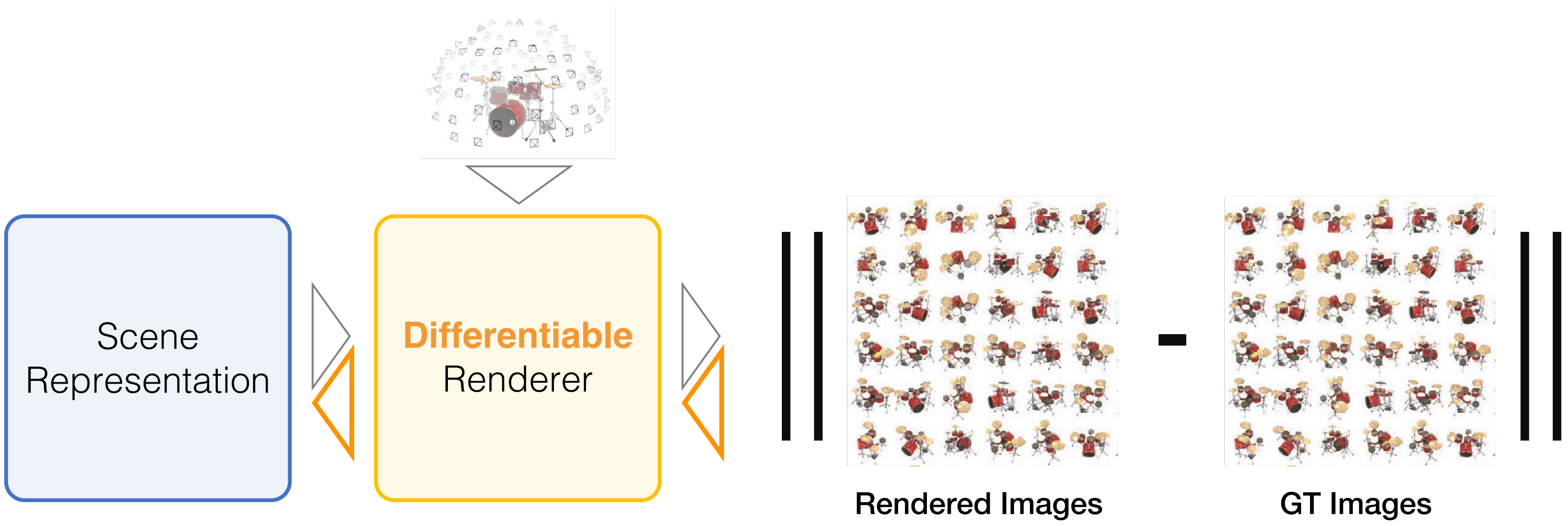
Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Differentiable Rendering



Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

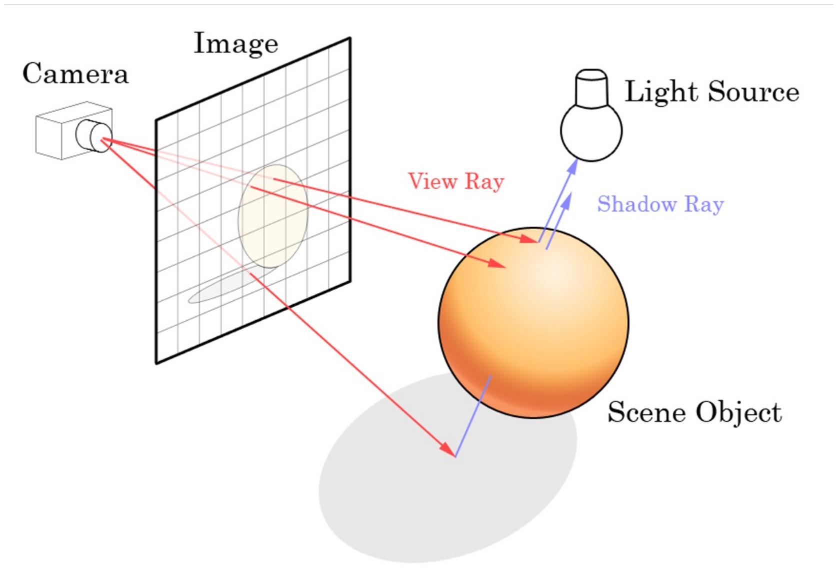
Differentiable Rendering



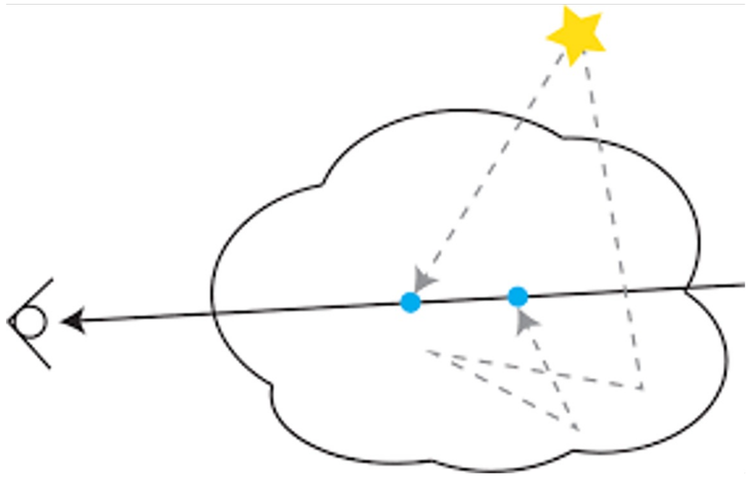
Given an *observable* variable (pixel colors), we will build a differentiable forward model that we then use to estimate *unobserved (latent) variables* (geometry, appearance)!

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Ways to Render



Surface rendering



Volume rendering

Volume rendering equation

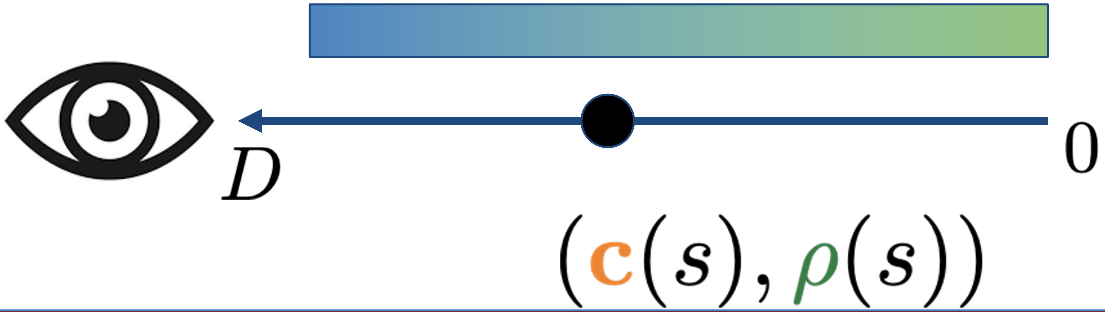
$$\mathbf{I}(D) = \mathbf{I}_0 T(0) + \int_0^D \mathbf{c}(s)\rho(s)T(s)ds$$

pixel color at coordinates D

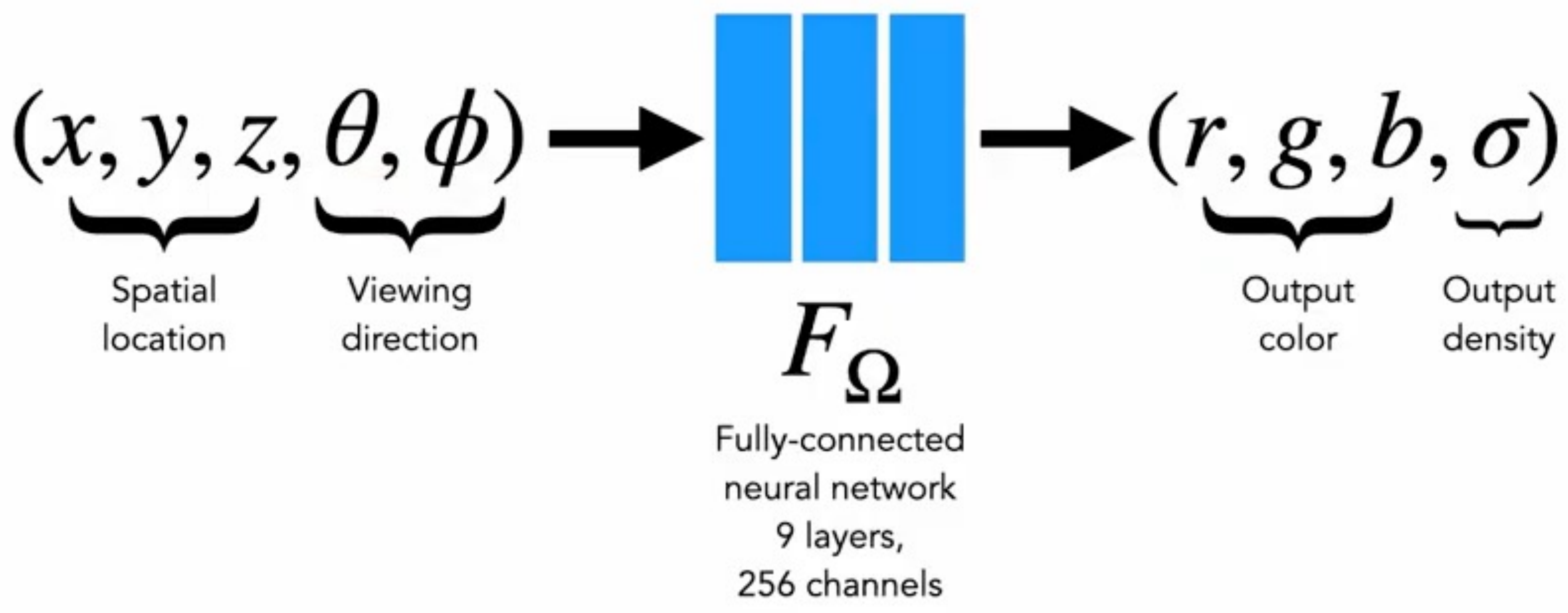
radiance **density**

$$T(s) = \exp\left(-\int_s^D \rho(t)dt\right)$$

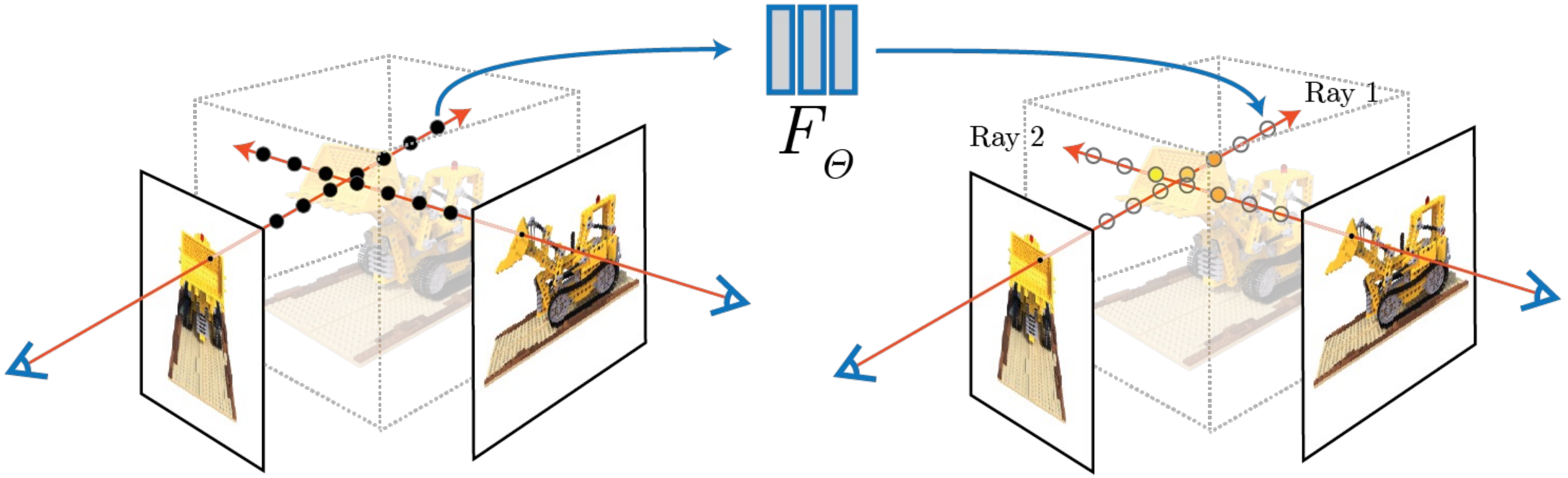
transparency



Represent a scene as a continuous 5D function


$(\underbrace{x, y, z}_{\text{Spatial location}}, \underbrace{\theta, \phi}_{\text{Viewing direction}}) \rightarrow F_\Omega \rightarrow (\underbrace{r, g, b}_{\text{Output color}}, \underbrace{\sigma}_{\text{Output density}})$
Spatial location
Viewing direction
F_Ω
Fully-connected neural network 9 layers, 256 channels
Output color
Output density

Generate views with traditional volume rendering



Mildenhall et al. ECCV 2020. https://www.matthewtancik.com/nerf

Generate views with traditional volume rendering

Rendering model for ray r(t) = o + td:

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

t = point along ray
C = Color of Pixel
c = color of point

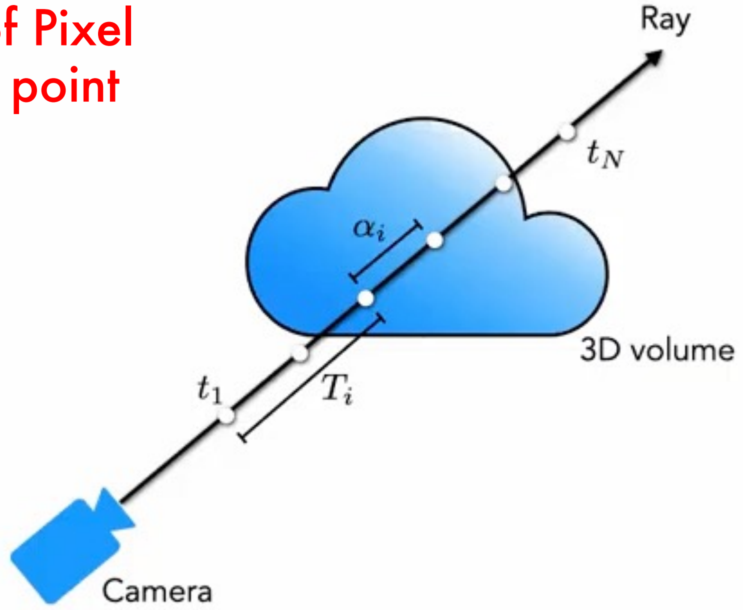
How much light is blocked earlier along ray:

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

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

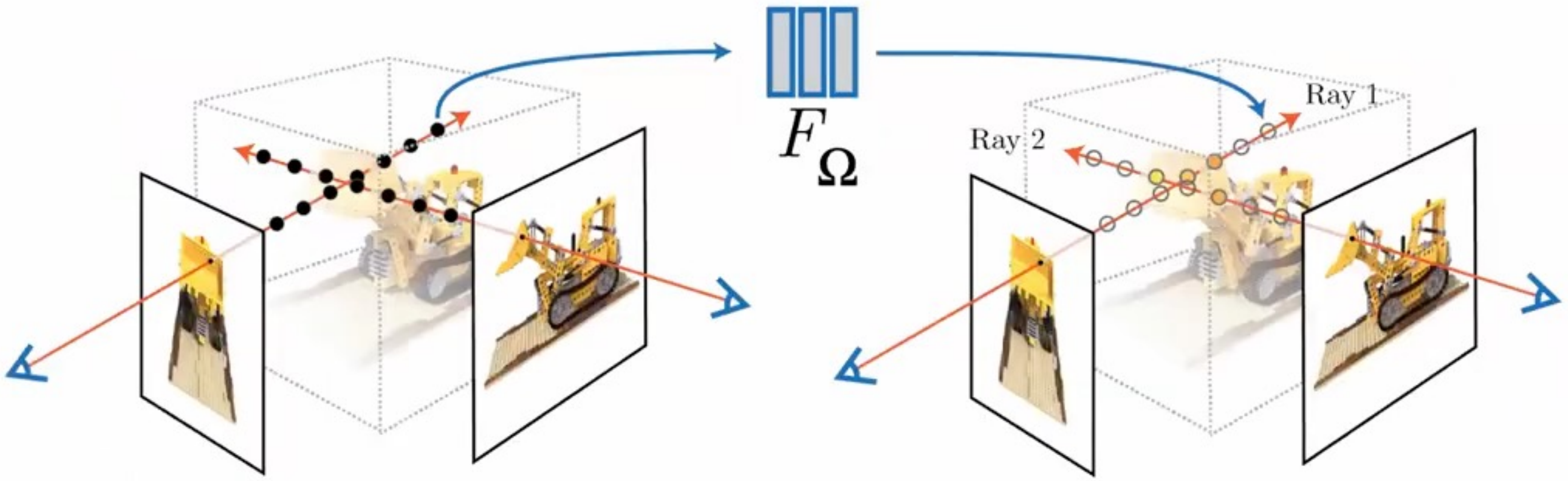
$$\alpha_i = 1 - e^{-\sigma_i \delta t_i}$$

Function of segment length δt_i and volume density σ

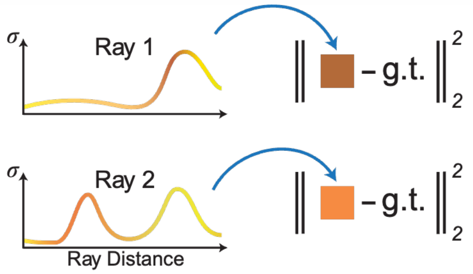


From Presentation by Matthew Tancik: Neural Radiance Fields for View Synthesis. 2020.

Optimize with gradient descent on rendering loss

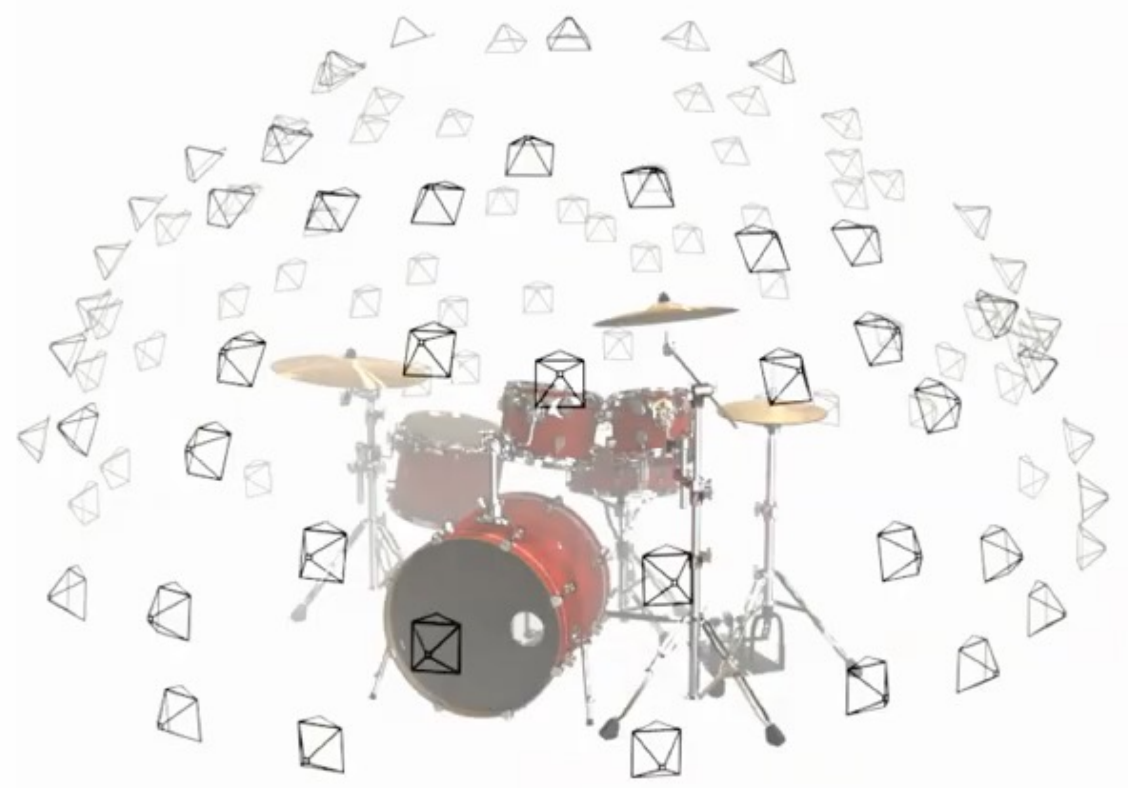


$$\min_{\Omega} \sum_i \|\text{render}^{(i)}(F_\Omega) - I_{\text{gt}}^{(i)}\|^2$$



From Presentation by Matthew Tancik: Neural Radiance Fields for View Synthesis. 2020.

Training network to reproduce all input views of the scene



From Presentation by Matthew Tancik: Neural Radiance Fields for View Synthesis. 2020.

The problem of novel view synthesis

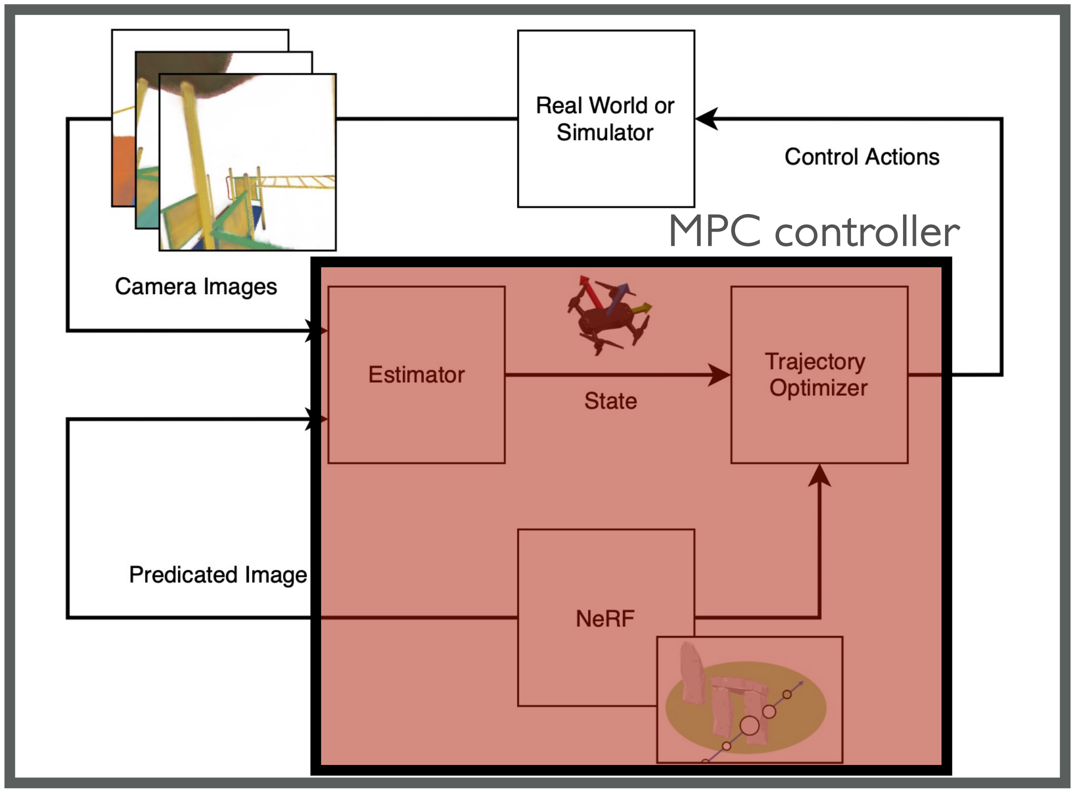
Inputs: sparsely sampled images of scene

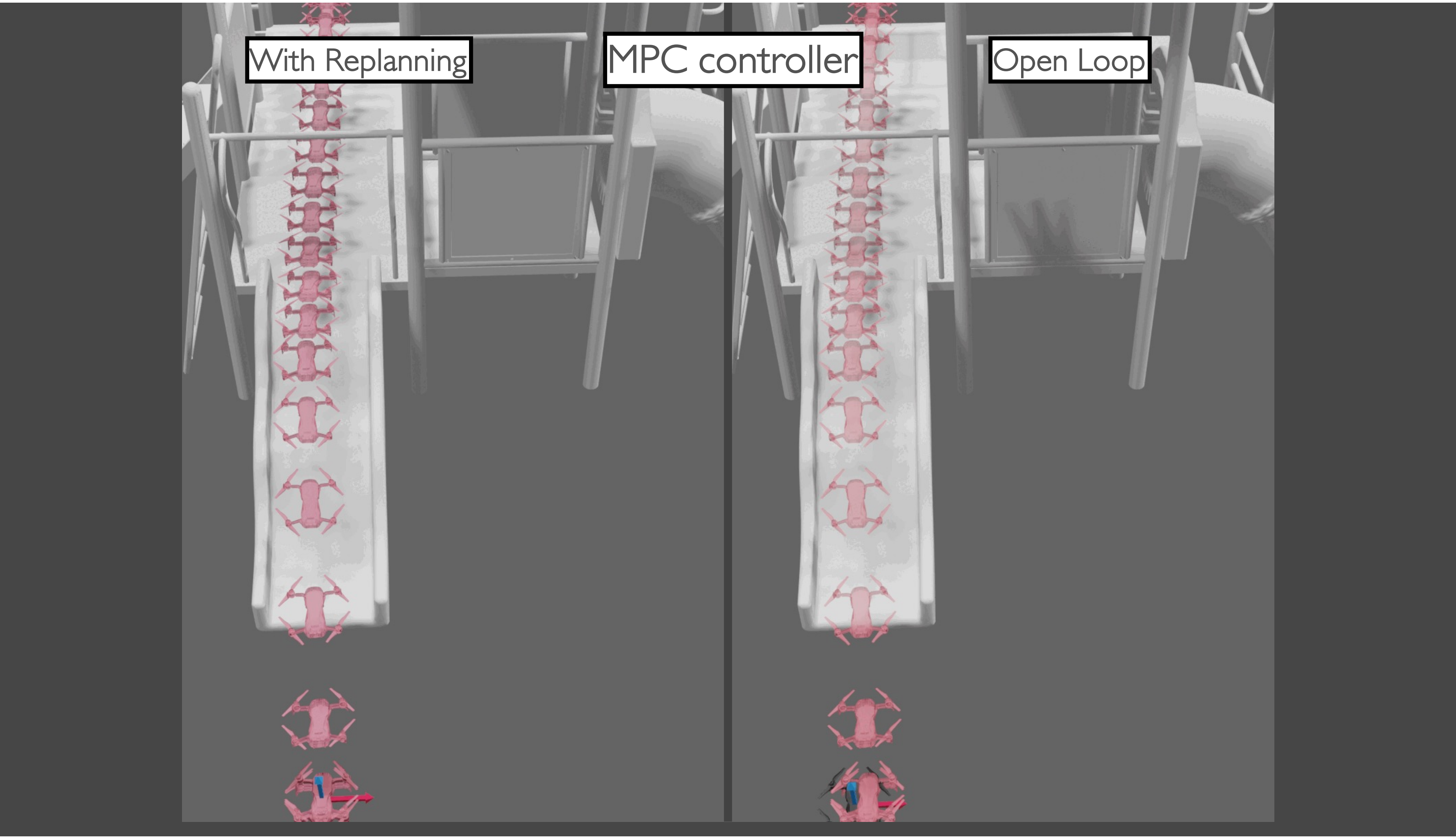
Outputs: *new* views of same scene
(rendered by our method)

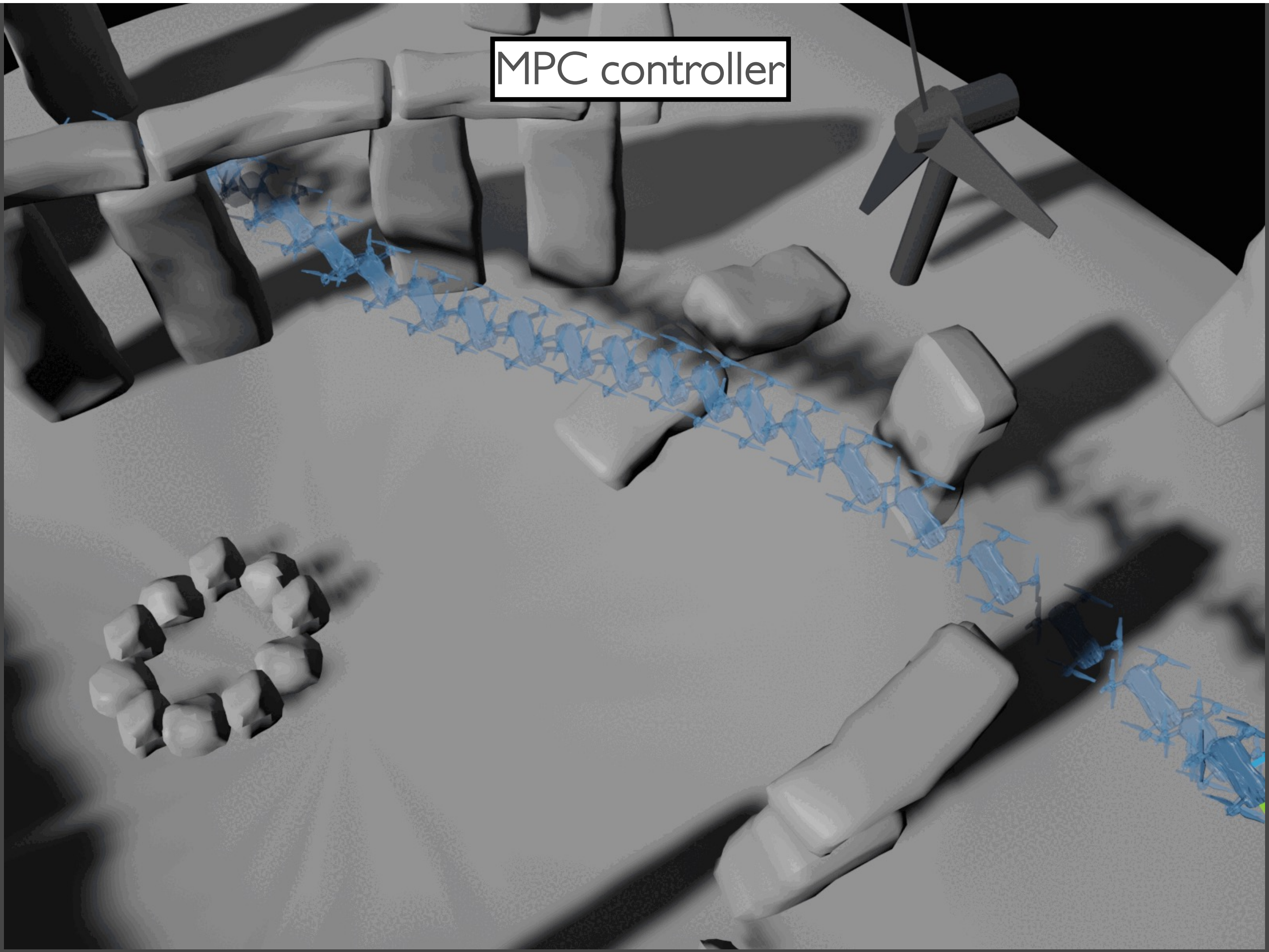
2

Mildenhall et al. ECCV 2020. https://www.matthewtancik.com/nerf

Vision-Only Navigation

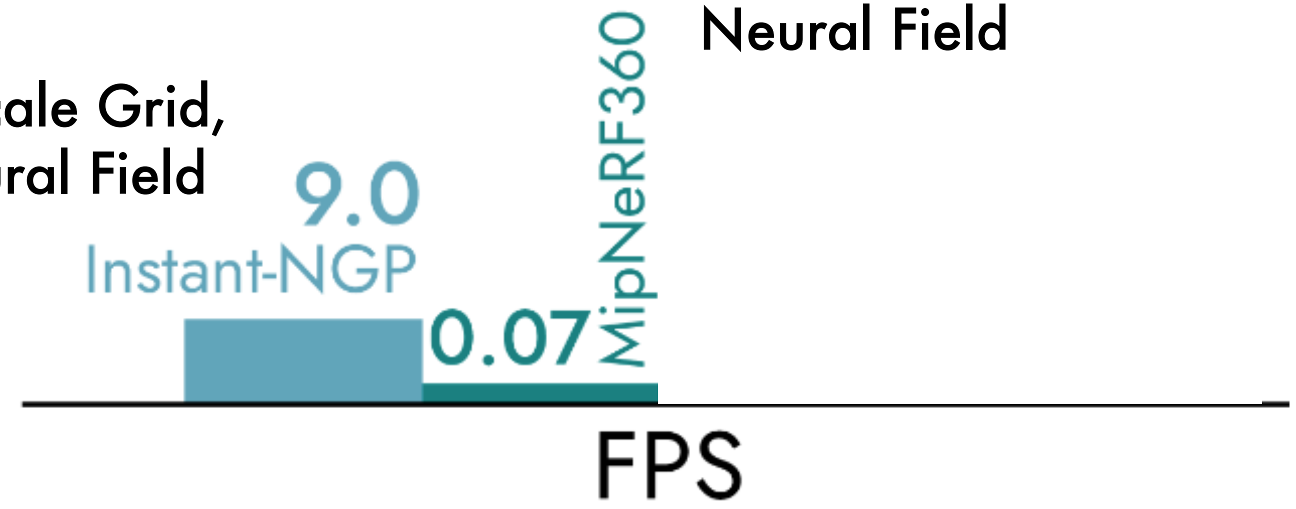



With Replanning
MPC controller
Open Loop

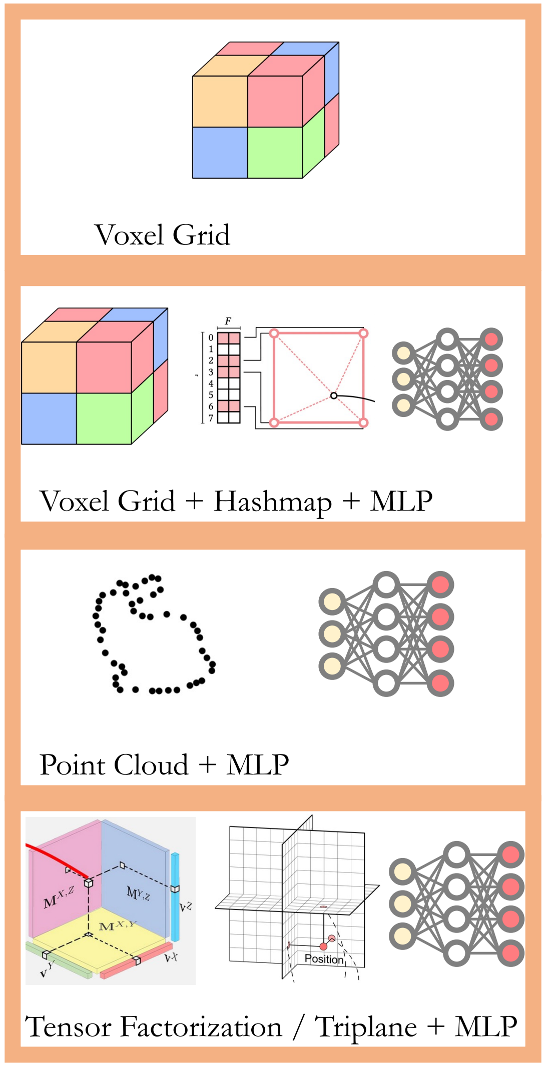

MPC controller

Neural Field

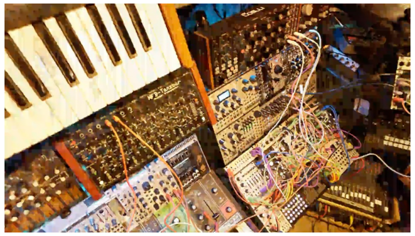
Hybrid Multi-Scale Grid, HashMap, Neural Field



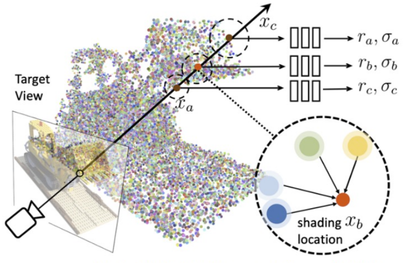
Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann


Voxel Grid
Voxel Grid + Hashmap + MLP
Point Cloud + MLP
Tensor Factorization / Triplane + MLP

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



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



(b) Point-NeRF Representation with Volum

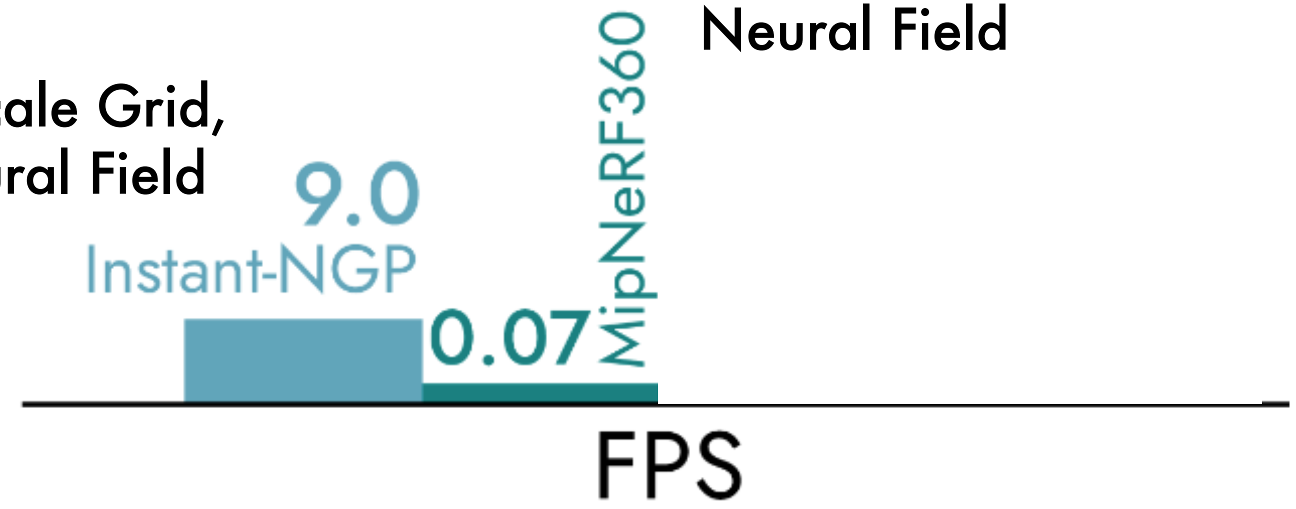
PointNeRF: Point-based Neural …
[Xu et al. 2022]

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

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Neural Field

Hybrid Multi-Scale Grid,
HashMap, Neural Field



Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

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

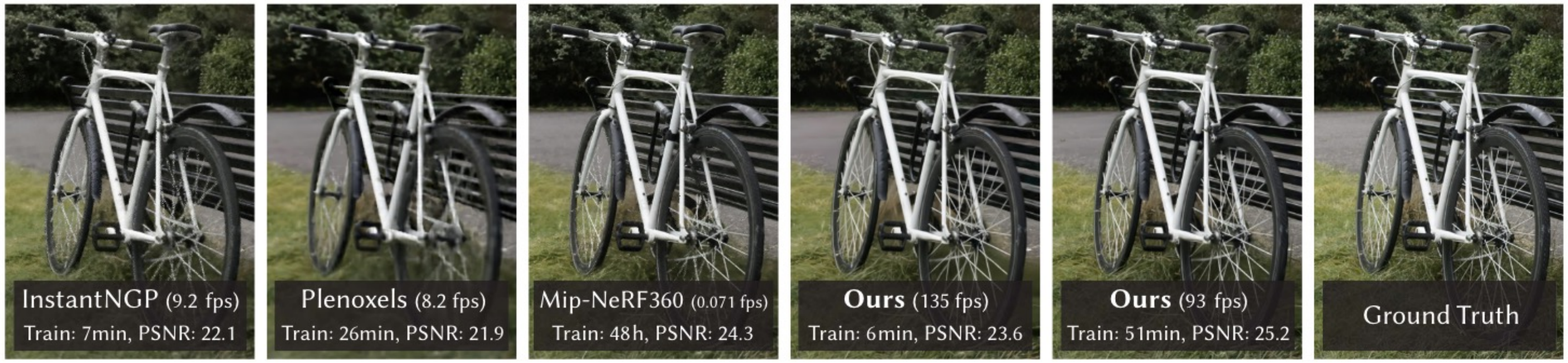
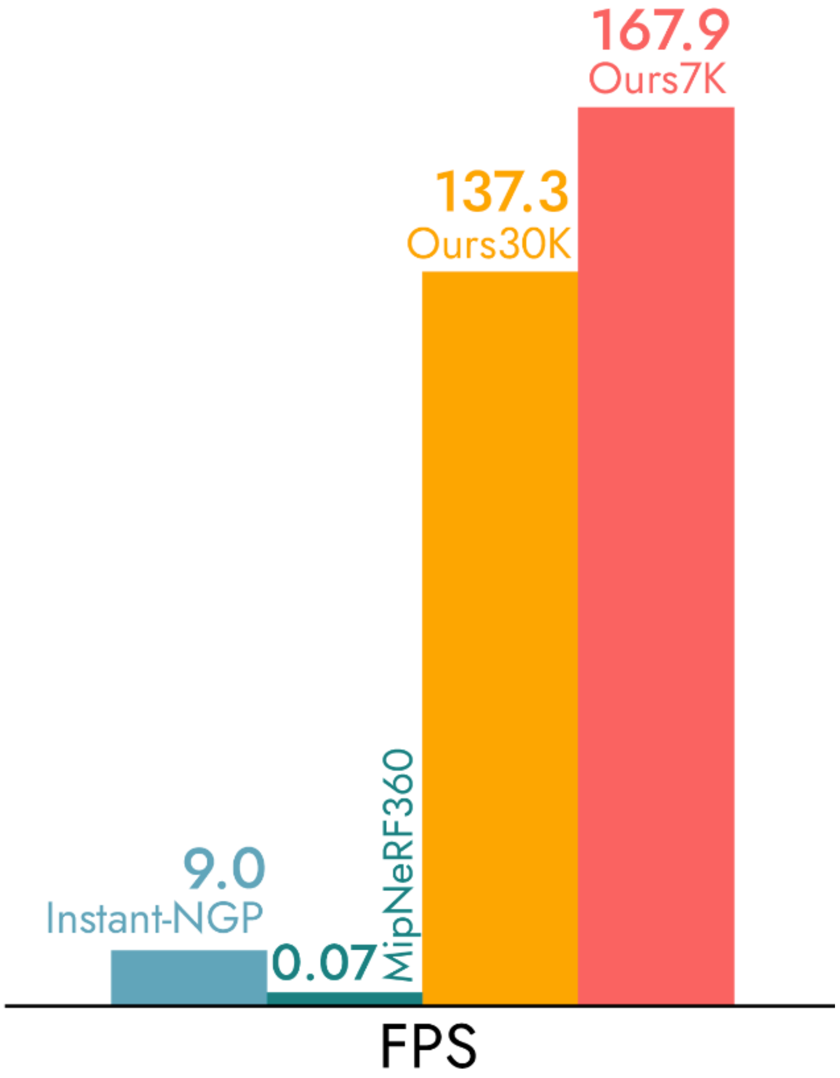


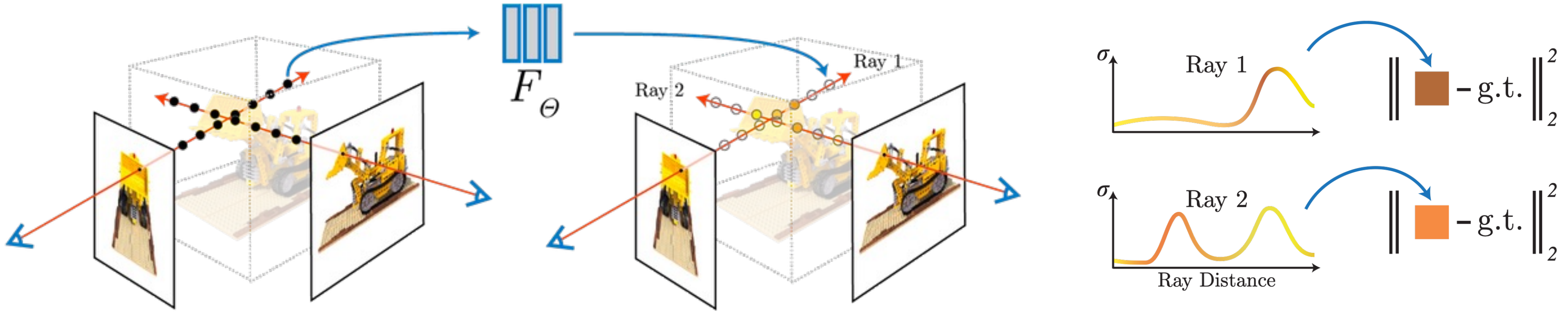
Fig. 1. Our method achieves real-time rendering of radiance fields with quality that equals the previous method with the best quality [Barron et al. 2022], while only requiring optimization times competitive with the fastest previous methods [Fridovich-Keil and Yu et al. 2022; Müller et al. 2022]. Key to this performance is a novel 3D Gaussian scene representation coupled with a real-time differentiable renderer, which offers significant speedup to both scene optimization and novel view synthesis. Note that for comparable training times to InstantNGP [Müller et al. 2022], we achieve similar quality to theirs; while this is the maximum quality they reach, by training for 51min we achieve state-of-the-art quality, even slightly better than Mip-NeRF360 [Barron et al. 2022].



Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

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

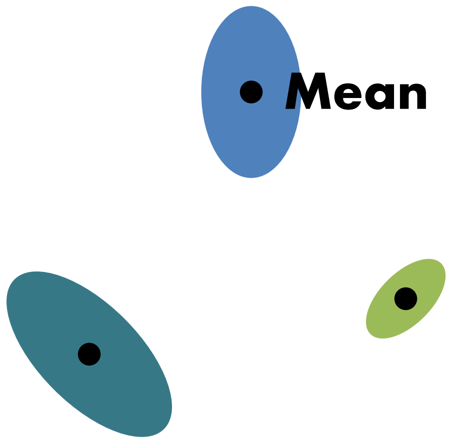
$$\begin{tikzcd}(x,y,z,\theta,\phi) \arrow[r] & F_\Theta \arrow[r] & (RGB\sigma)\end{tikzcd}$$


F_Θ
Ray 2
Ray 1
σ
Ray 1
$\| \blacksquare - \text{g.t.} \|_2^2$
σ
Ray 2
Ray Distance
$\| \blacksquare - \text{g.t.} \|_2^2$

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

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

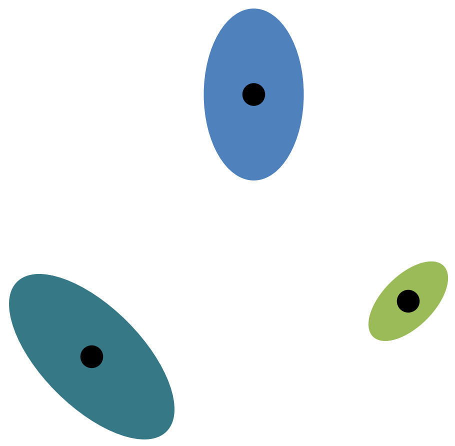
3D Gaussian Blobs floating in Space



Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

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

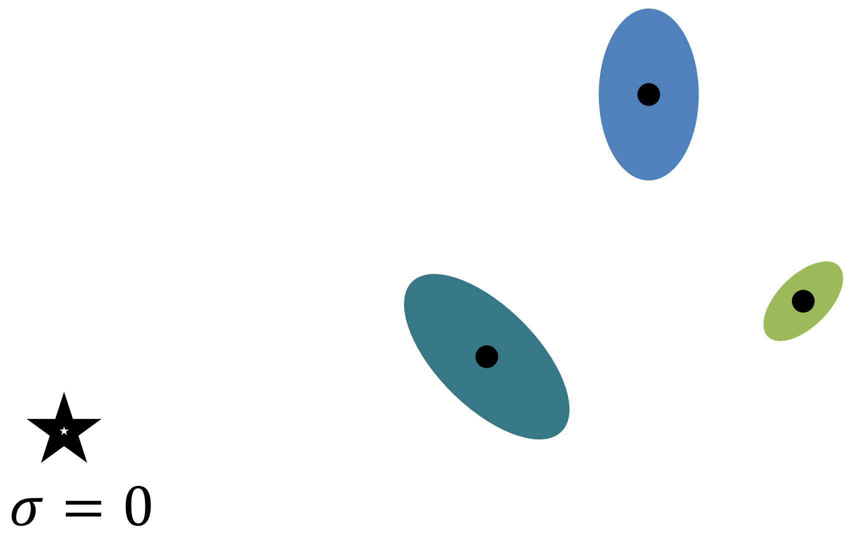
3D Gaussian Blobs floating in Space



Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

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

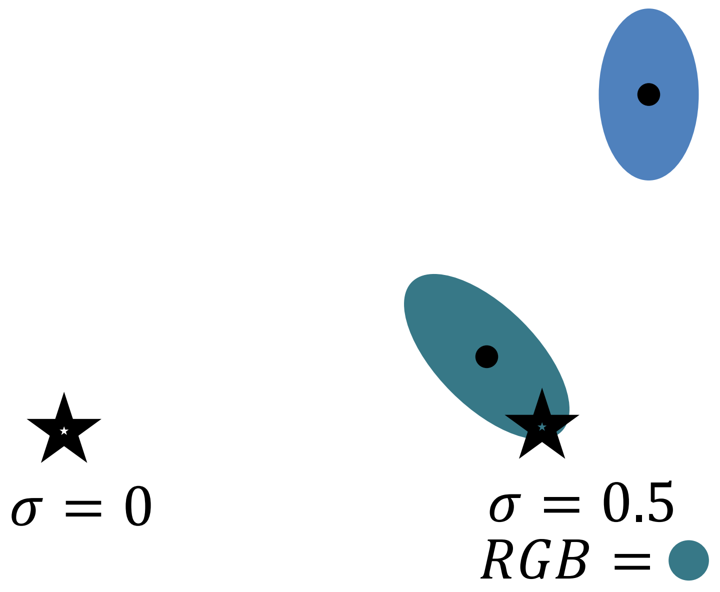
3D Gaussian Blobs floating in Space



Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

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

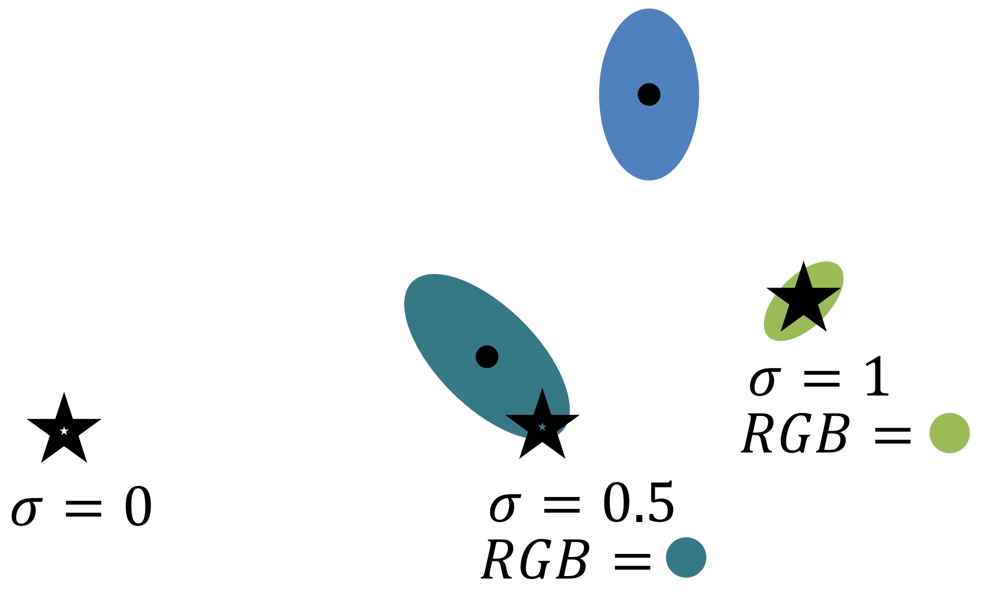
3D Gaussian Blobs floating in Space



Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

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

3D Gaussian Blobs floating in Space



Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

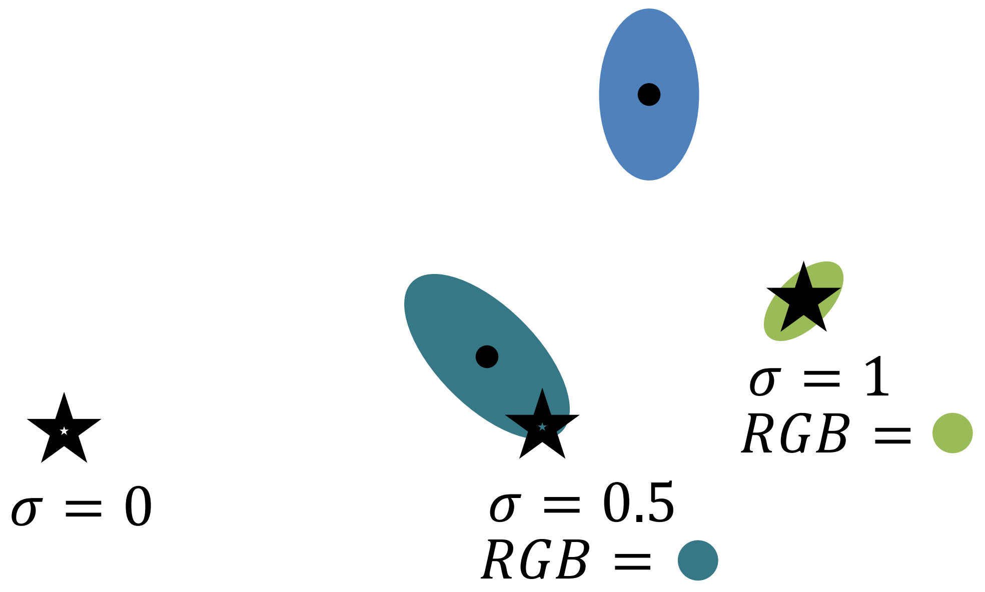
Anisotropic Volumetric 3D Gaussians

Final Rendering

3D Gaussian Visualization

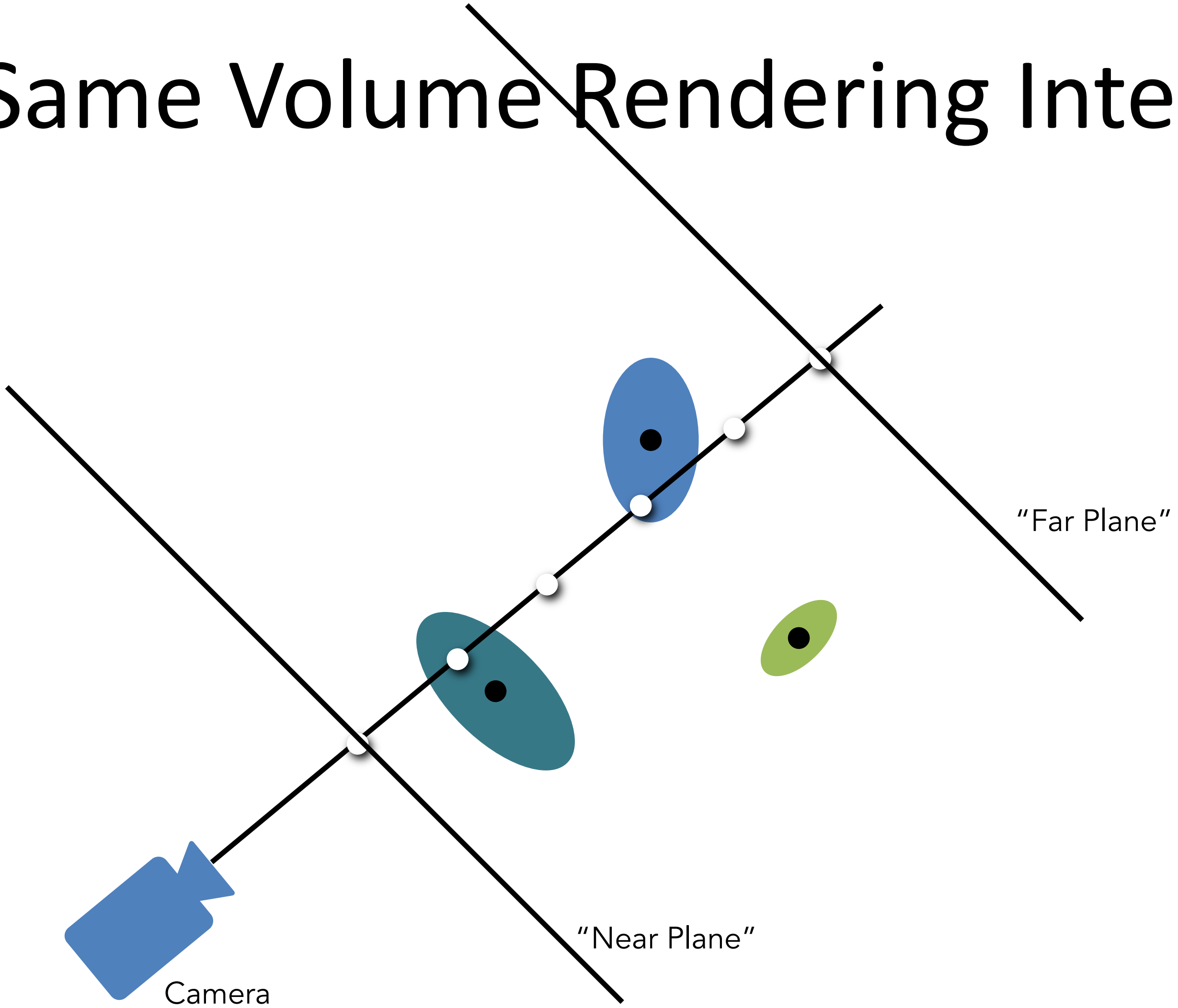
How to Render?

3D Gaussian Blobs floating in Space



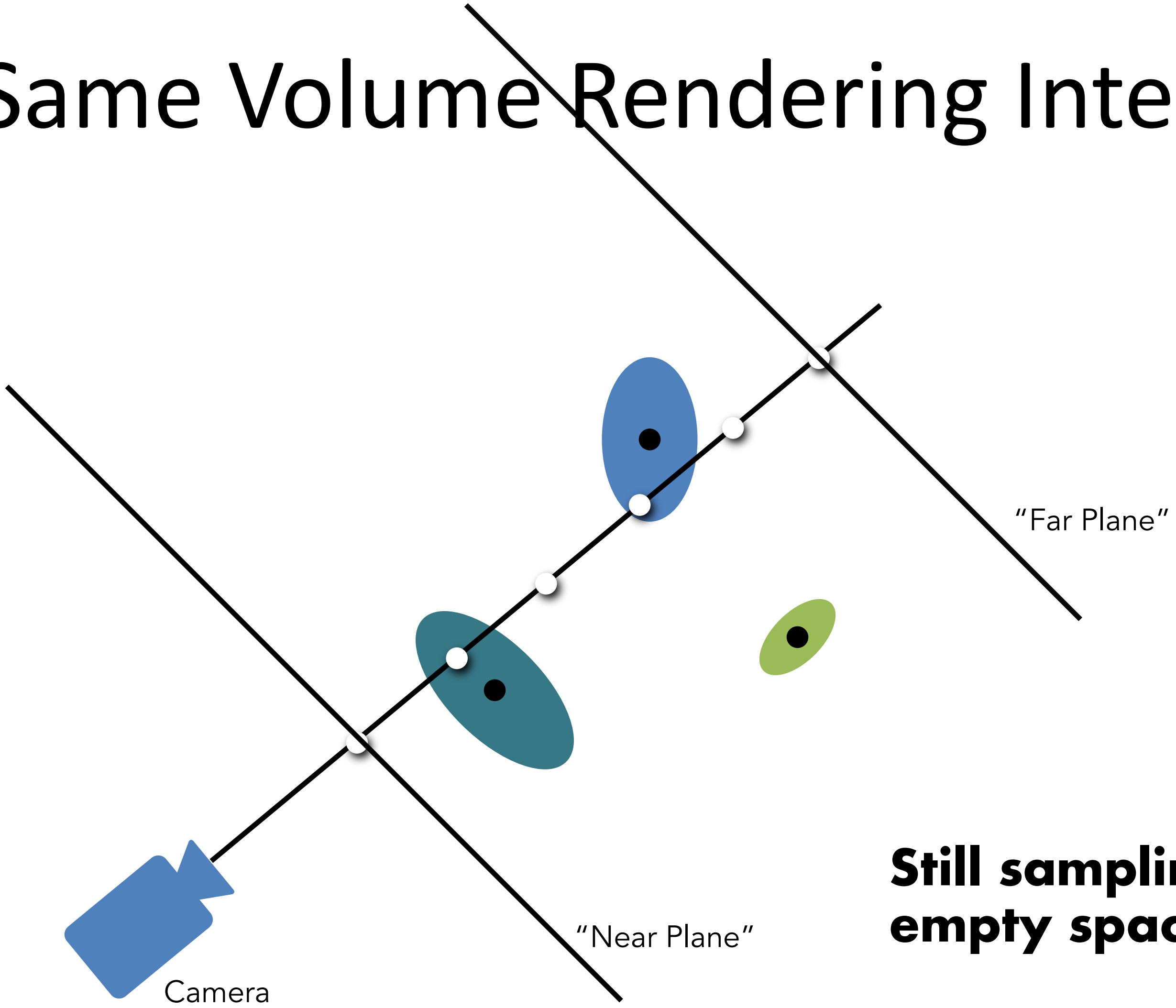
Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Same Volume Rendering Integral!



Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

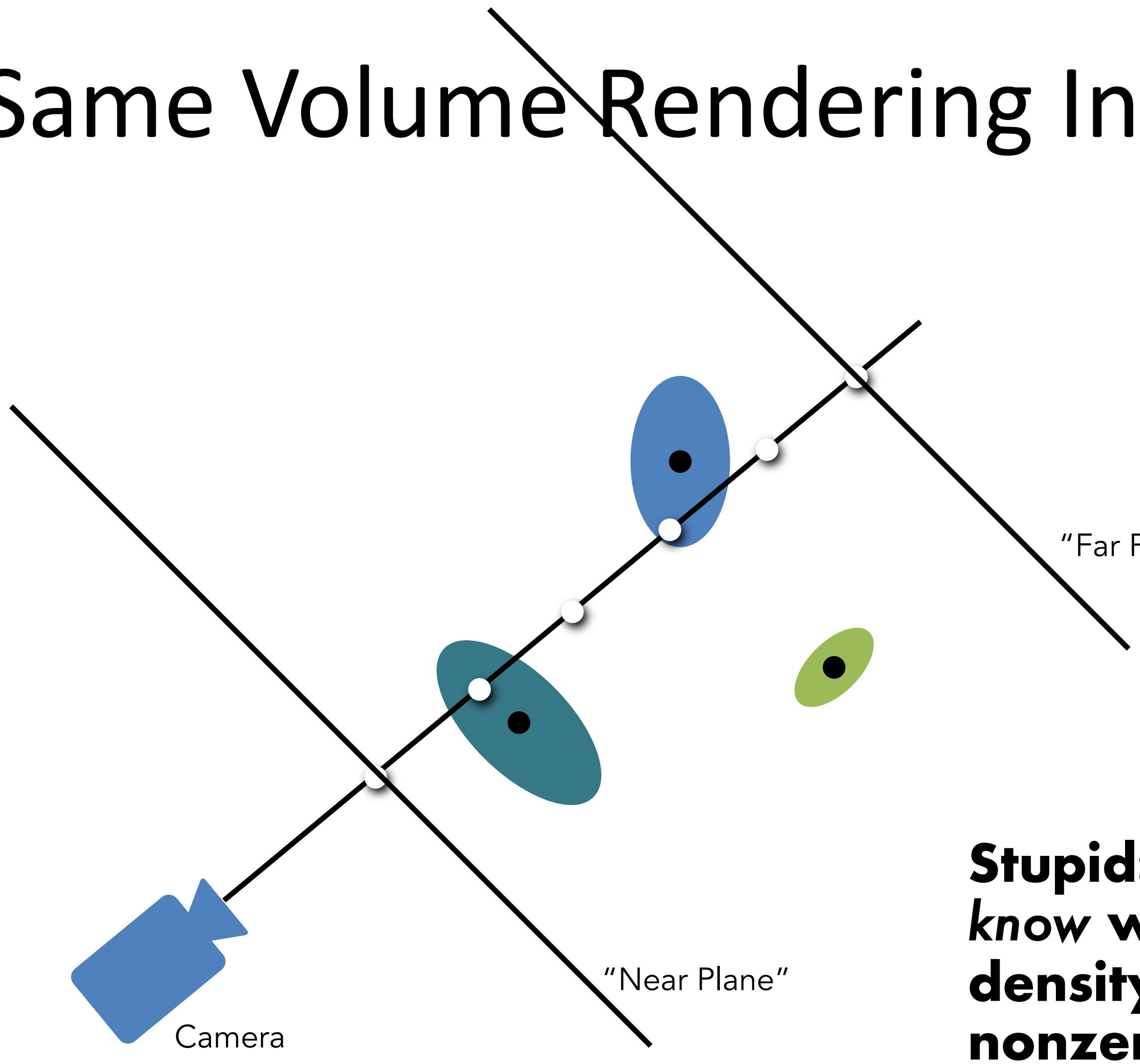
Same Volume Rendering Integral!



Still sampling lots of empty space...

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Same Volume Rendering Integral!



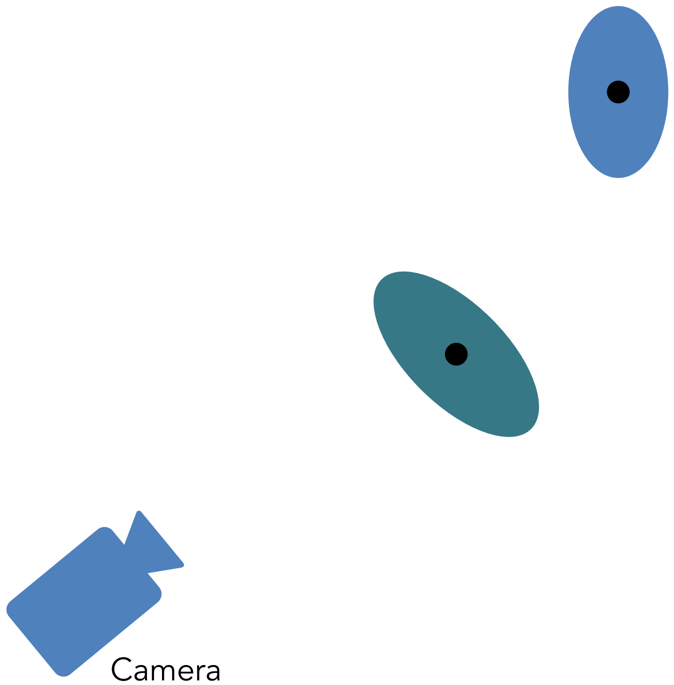
Stupid: we *already know* where the density will be nonzero!

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Gaussians are closed under affine transforms, integration

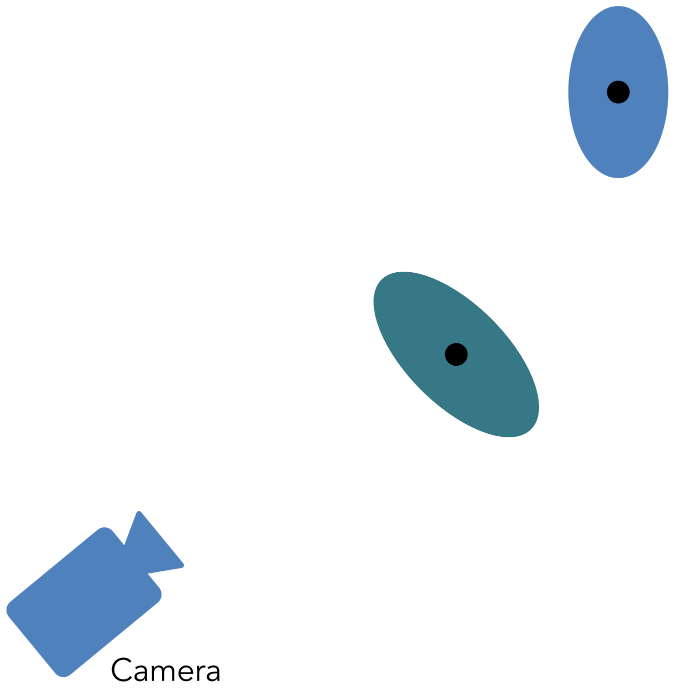
$$\mathcal{G}_{\mathbf{V}}(\mathbf{x}-\mathbf{p}) = \frac{1}{2\pi|\mathbf{V}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{p})^T \mathbf{V}^{-1}(\mathbf{x}-\mathbf{p})}$$

$\uparrow$ 3D Covariance!



Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Gaussians are closed under affine transforms, integration



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

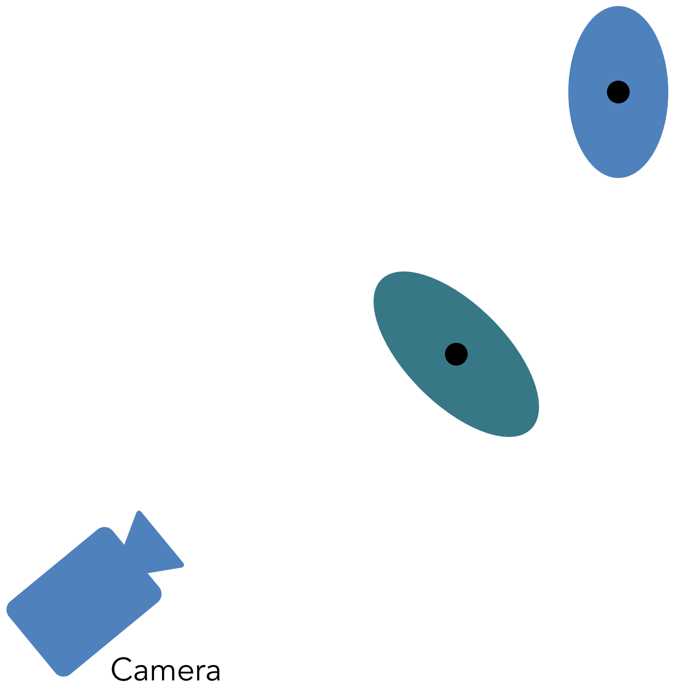
$\uparrow$ 3D Covariance!

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

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

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Gaussians are closed under affine transforms, integration



$$\mathcal{G}_{\mathbf{V}}(\mathbf{x}-\mathbf{p}) = \frac{1}{2\pi|\mathbf{V}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{p})^T \underset{\substack{\uparrow \\ \text{3D Covariance!}}}{\mathbf{V}^{-1}} (\mathbf{x}-\mathbf{p})}$$

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

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

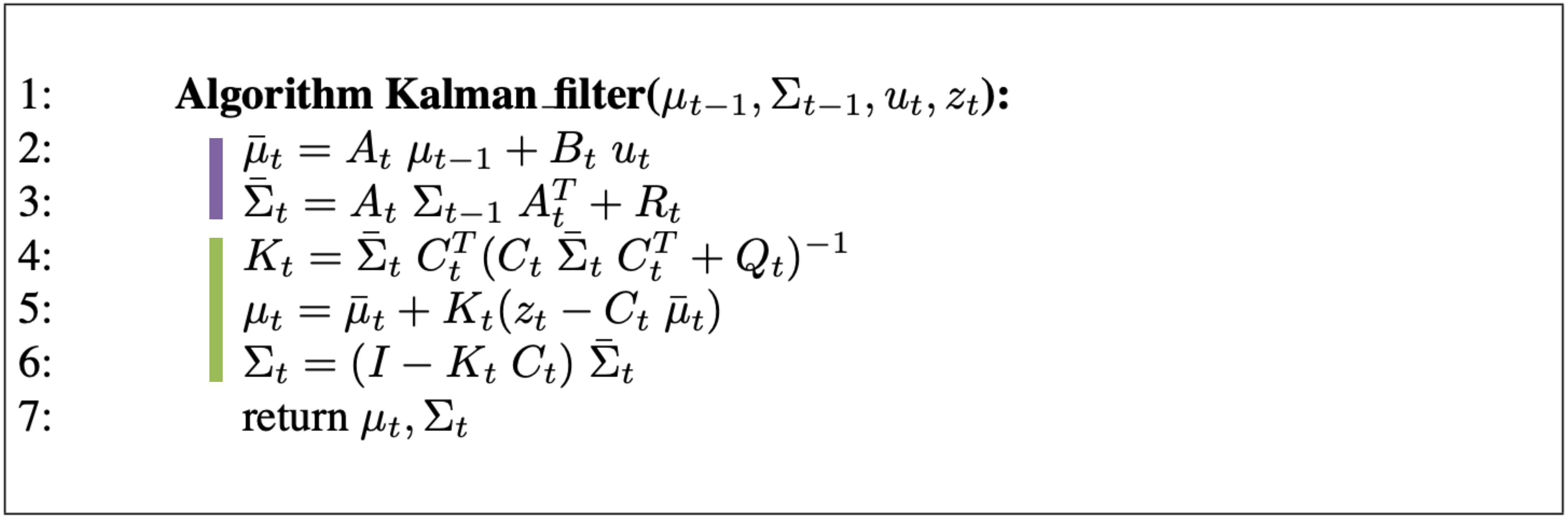
Integrate along axis:

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

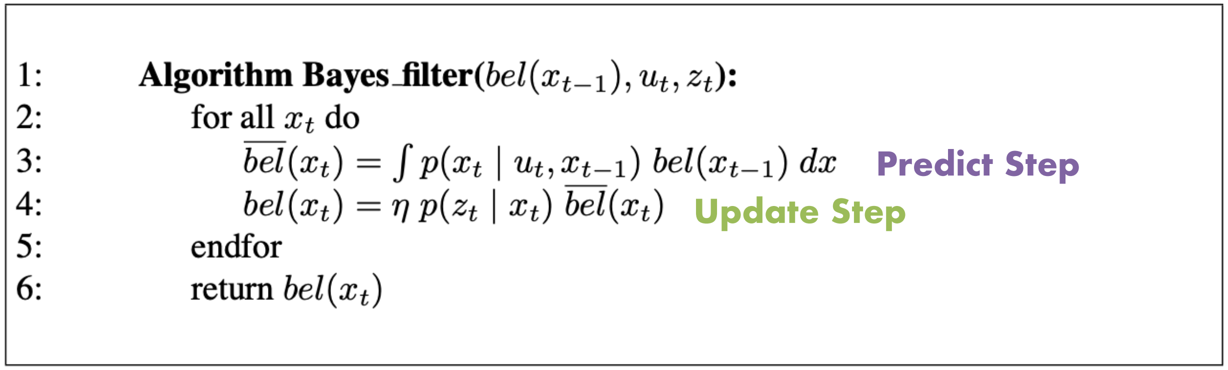
$$\mathbf{V} = \begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix} \Leftrightarrow \begin{pmatrix} a & b \\ b & d \end{pmatrix} = \hat{\mathbf{V}}$$

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Throwback: The Kalman Filter Algorithm

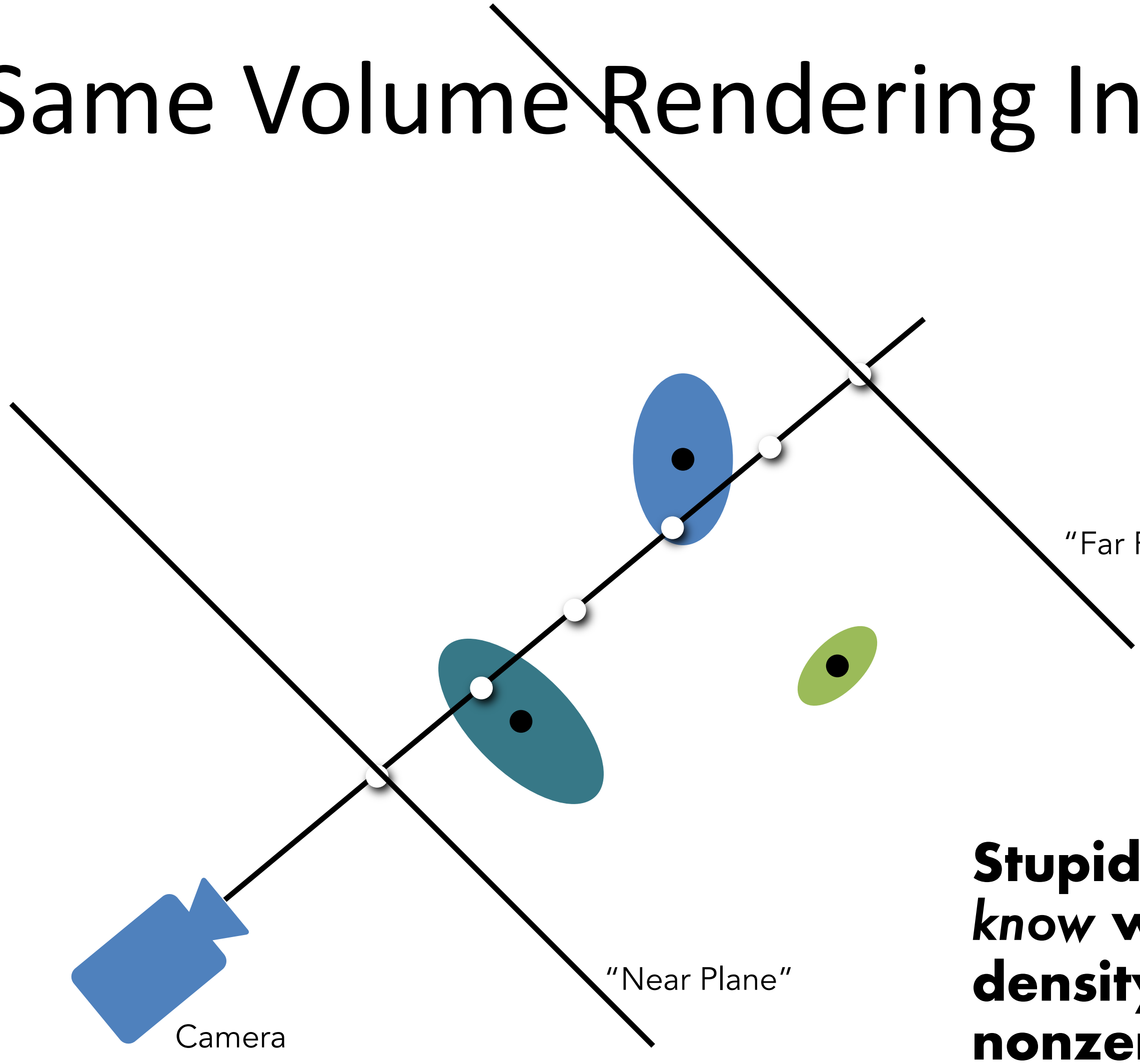


```
Algorithm Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
    $\bar{\mu}_t = A_t\, \mu_{t-1} + B_t\, u_t$
    $\bar{\Sigma}_t = A_t\, \Sigma_{t-1}\, A_t^T + R_t$
    $K_t = \bar{\Sigma}_t\, C_t^T (C_t\, \bar{\Sigma}_t\, C_t^T + Q_t)^{-1}$
    $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\, \bar{\mu}_t)$
    $\Sigma_t = (I - K_t\, C_t)\, \bar{\Sigma}_t$
    return $\mu_t, \Sigma_t$
```



```
Algorithm Bayes_filter($bel(x_{t-1}), u_t, z_t$):
    for all $x_t$ do
        $\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\, bel(x_{t-1})\, dx$    Predict Step
        $bel(x_t) = \eta\, p(z_t \mid x_t)\, \overline{bel}(x_t)$    Update Step
    endfor
    return $bel(x_t)$
```

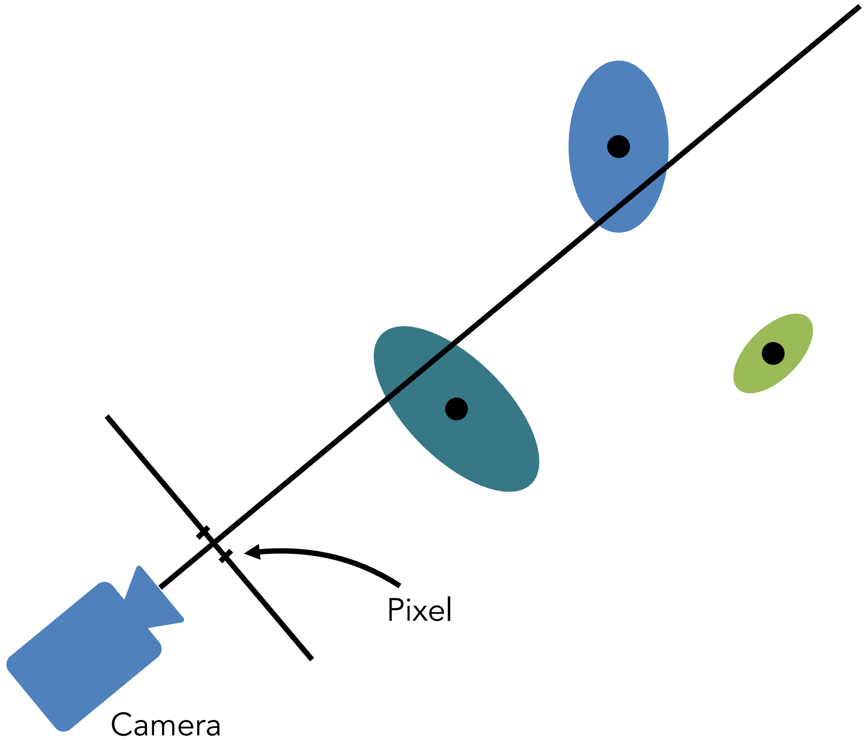
Same Volume Rendering Integral!



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

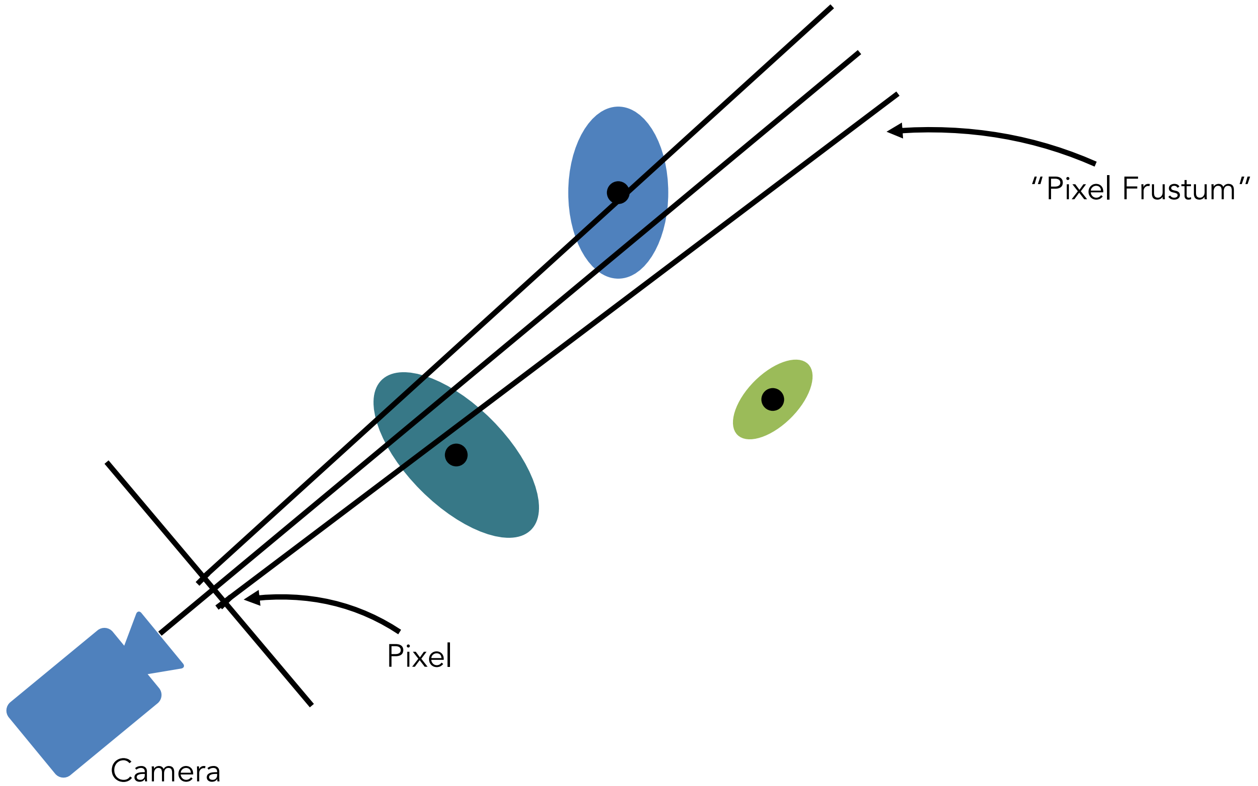
Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Instead: Rasterization



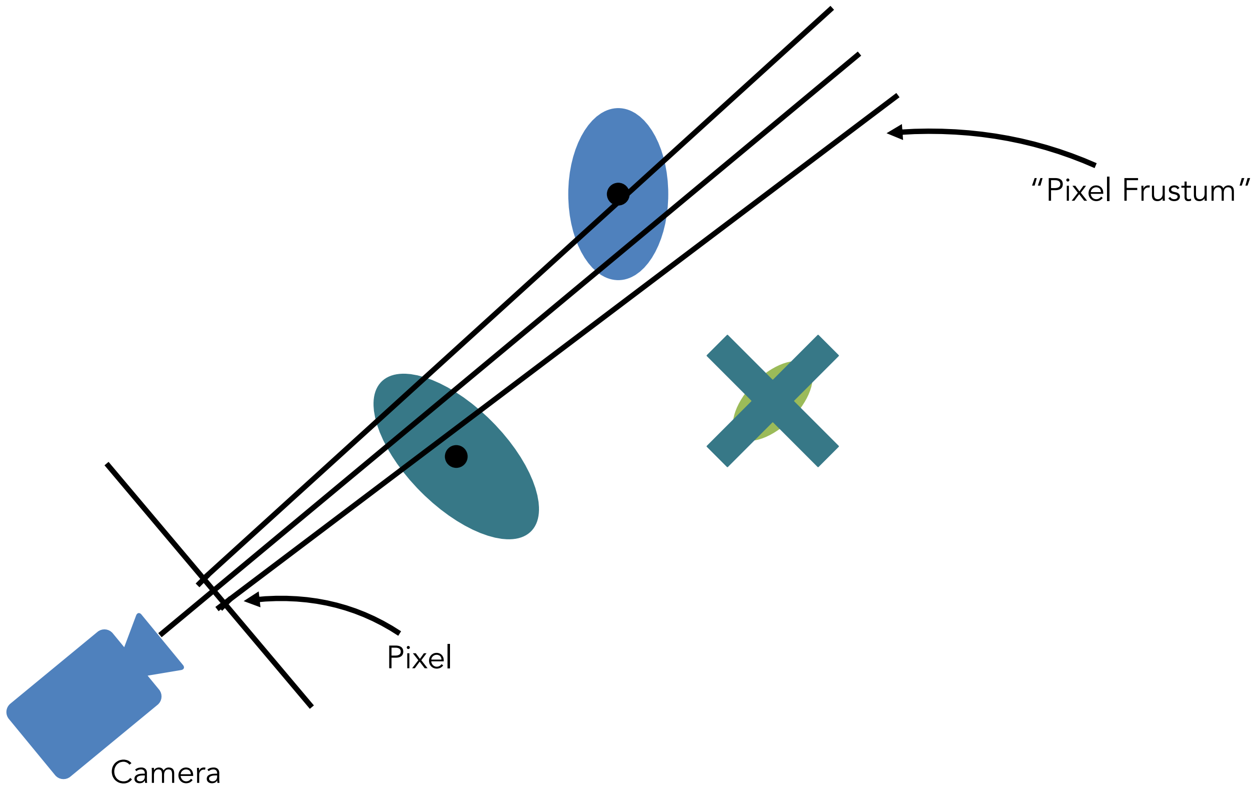
Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Instead: Rasterization



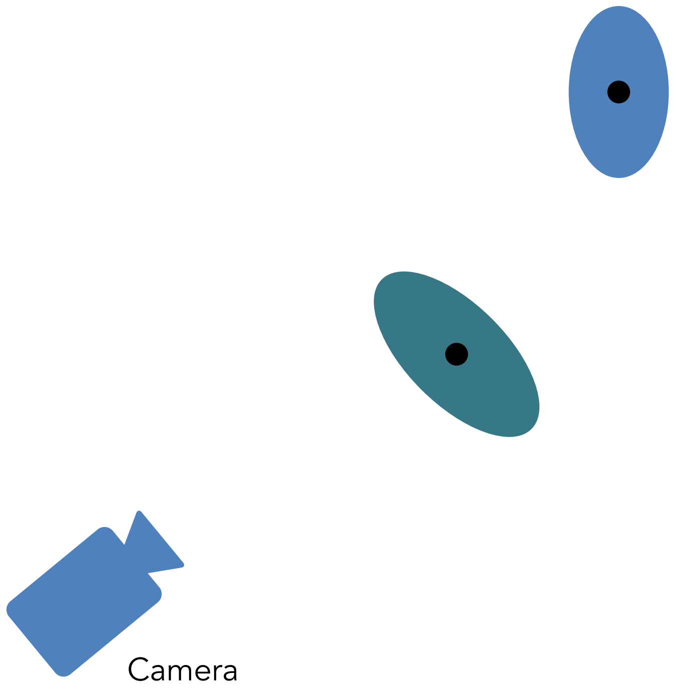
Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

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



Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Step 1: Transform Gaussians into Camera Coordinates

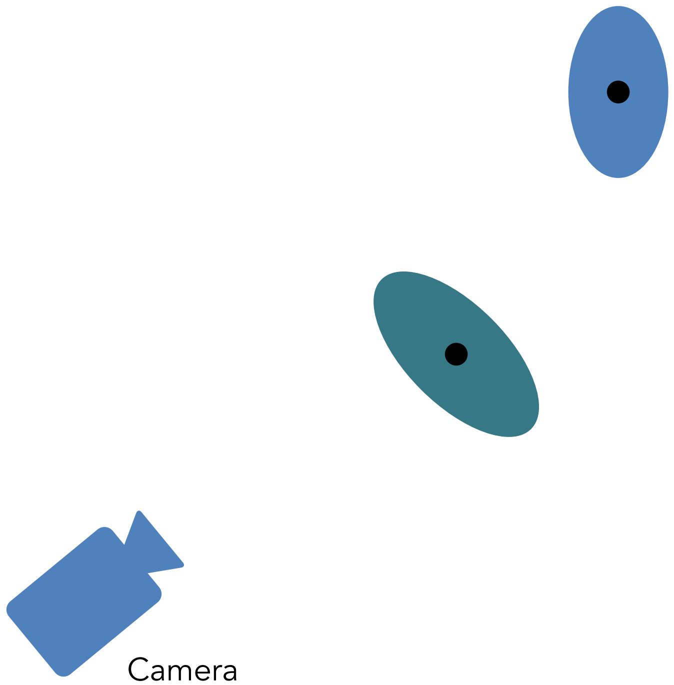


Cam2world is affine mapping $\phi(x) = \mathbf{W}\mathbf{x} + \mathbf{p}$:

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

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Step 1: Transform Gaussians into Camera Coordinates



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

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

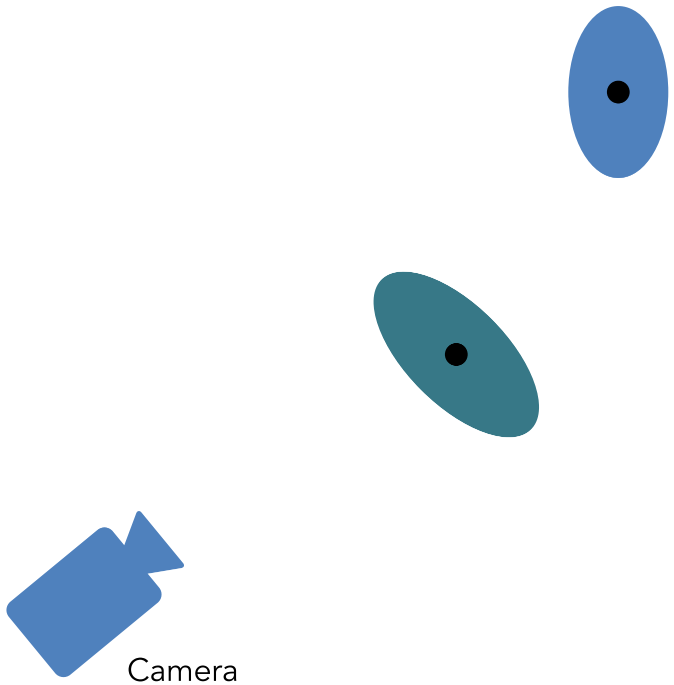
Projection $\mathbf{m}(u)$ is *not* an affine mapping :/

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

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

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

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 :/

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

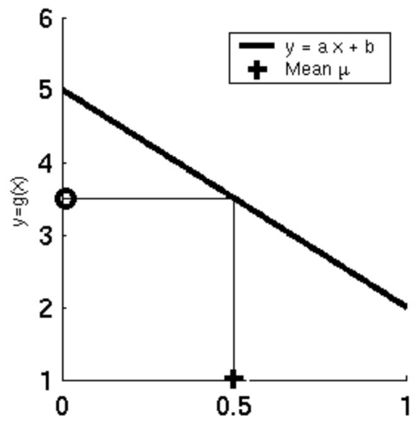
$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} = \mathbf{m}^{-1}(\mathbf{x}) = \begin{pmatrix} x_0/l \cdot x_2 \\ x_1/l \cdot x_2 \\ 1/l \cdot x_2 \end{pmatrix},$$

But can approximate with first-order Taylor Expansion as:

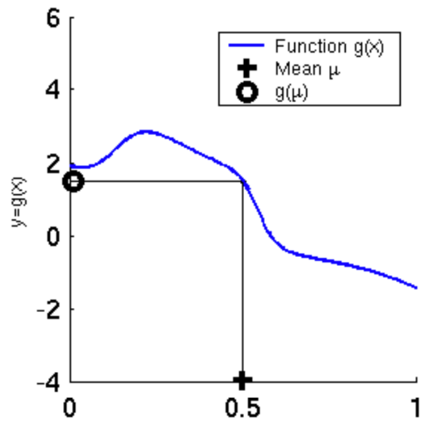
$$\mathbf{m}_{\mathbf{u}_k}(\mathbf{u}) = \mathbf{x}_k + \mathbf{J}_{\mathbf{u}_k} \cdot (\mathbf{u} - \mathbf{u}_k) \qquad \mathbf{J}_{\mathbf{u}_k} = \frac{\partial \mathbf{m}}{\partial \mathbf{u}}(\mathbf{u}_k)$$

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

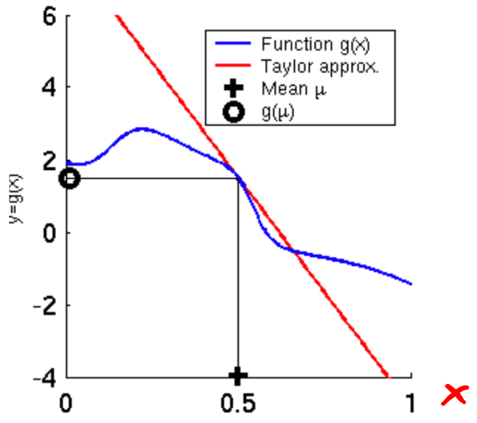
Propagating a Gaussian through a Linear Model



Propagating a Gaussian through a Non-Linear Model



Linearizing the Non-Linear Model



Throwback: The Extended Kalman Filter Algorithm

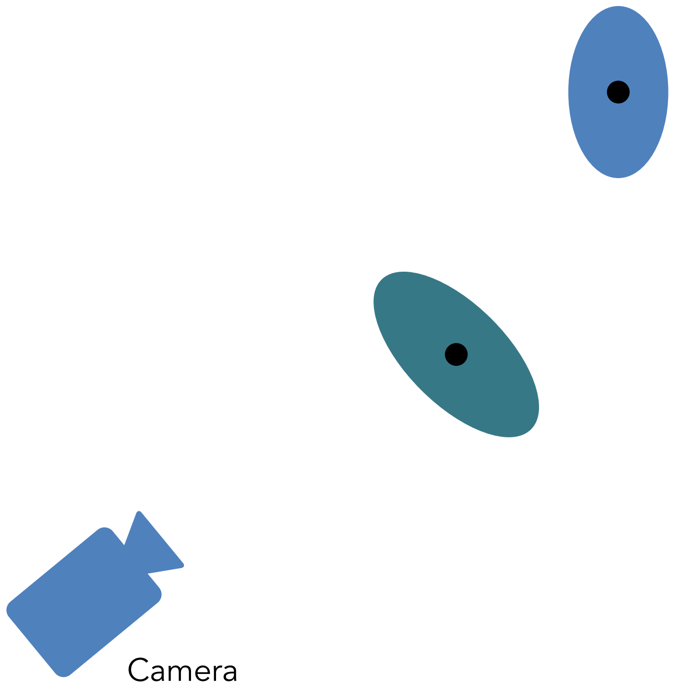
```
1:    Algorithm Extended_Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2:        $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3:        $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$
4:        $K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$
5:        $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6:        $\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$
7:        return $\mu_t, \Sigma_t$
```

Predict

Update

	Kalman filter	EKF
state prediction (Line 2)	$A_t \, \mu_{t-1} + B_t \, u_t$	$g(u_t, \mu_{t-1})$
measurement prediction (Line 5)	$C_t \, \bar{\mu}_t$	$h(\bar{\mu}_t)$

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 :/

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

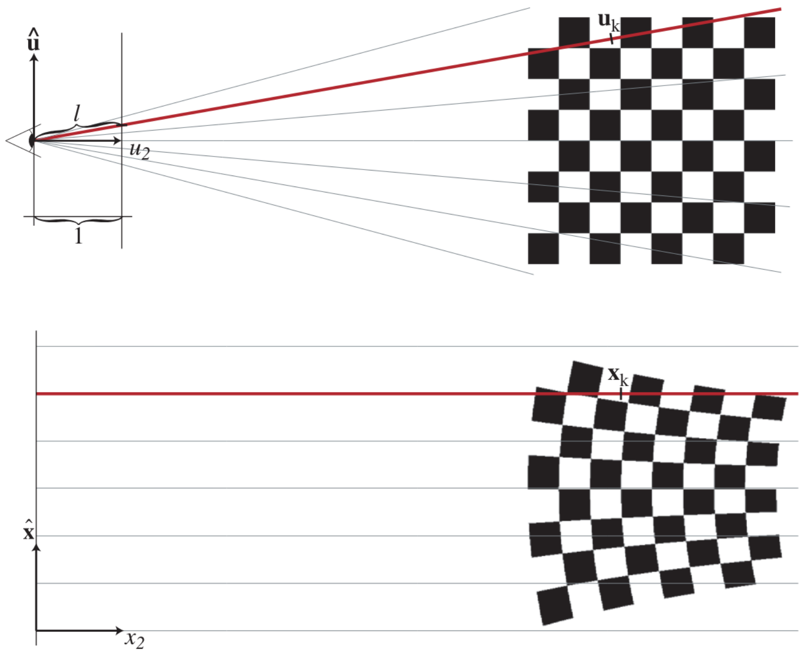
$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} = \mathbf{m}^{-1}(\mathbf{x}) = \begin{pmatrix} x_0/l \cdot x_2 \\ x_1/l \cdot x_2 \\ 1/l \cdot x_2 \end{pmatrix},$$

But can approximate with first-order Taylor Expansion as:

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

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Step 1: Transform Gaussians into Camera Coordinates

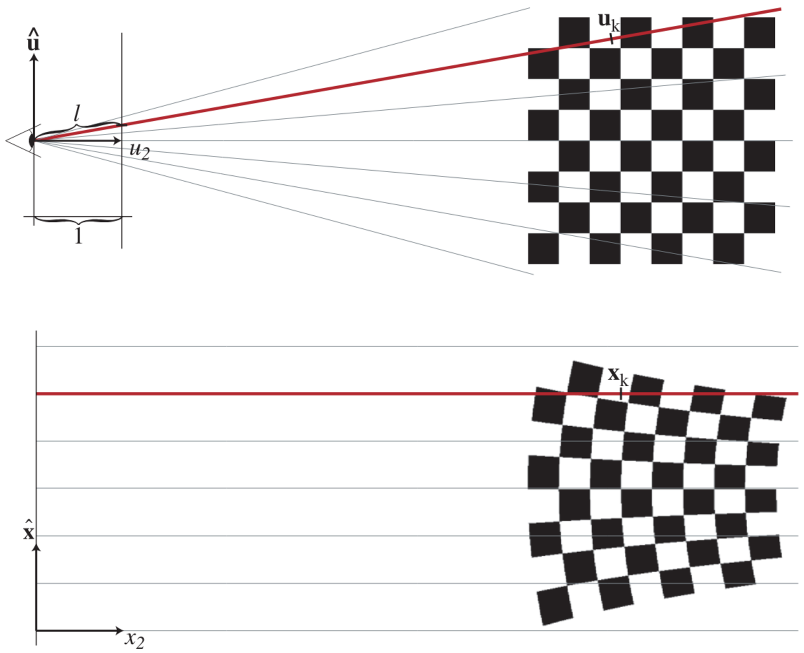


But can approximate with first-order Taylor Expansion as:

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

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Step 1: Transform Gaussians into Camera Coordinates



But can approximate with first-order Taylor Expansion as:

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

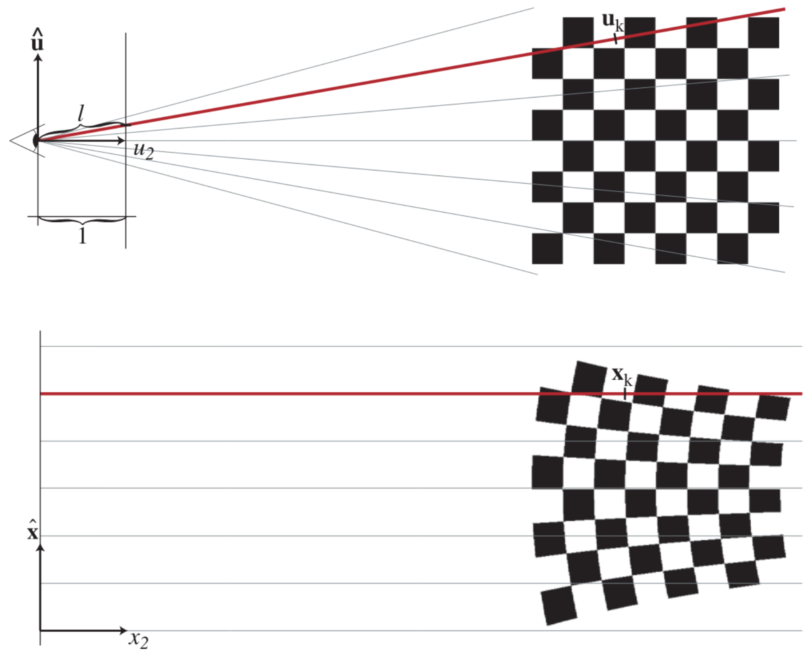
Projected, 2D Gaussians are then:

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

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

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

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:

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

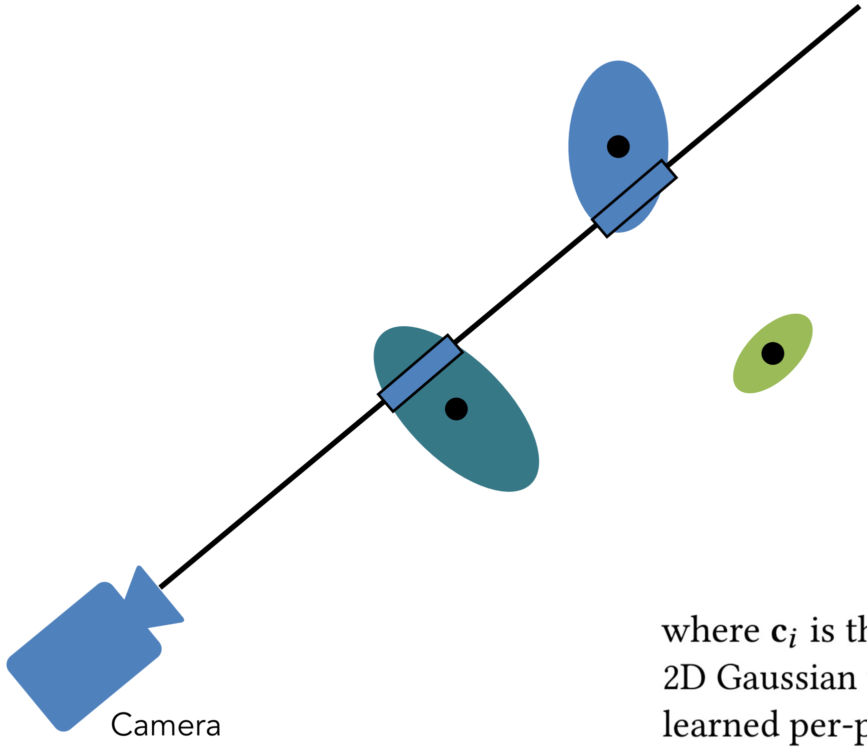
$$\begin{aligned}\mathbf{V}_k &= \mathbf{J}\mathbf{V}'_k\mathbf{J}^T \\ &= \mathbf{J}\mathbf{W}\mathbf{V}''_k\mathbf{W}^T\mathbf{J}^T.\end{aligned}$$

Finally, can integrate along rays:

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

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

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



Volumetric Rendering along a ray

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

with

$$\alpha_i = (1 - \exp(-\sigma_i \delta_i)) \quad \text{and} \quad T_i = \prod_{j=1}^{i-1} (1 - \alpha_i).$$

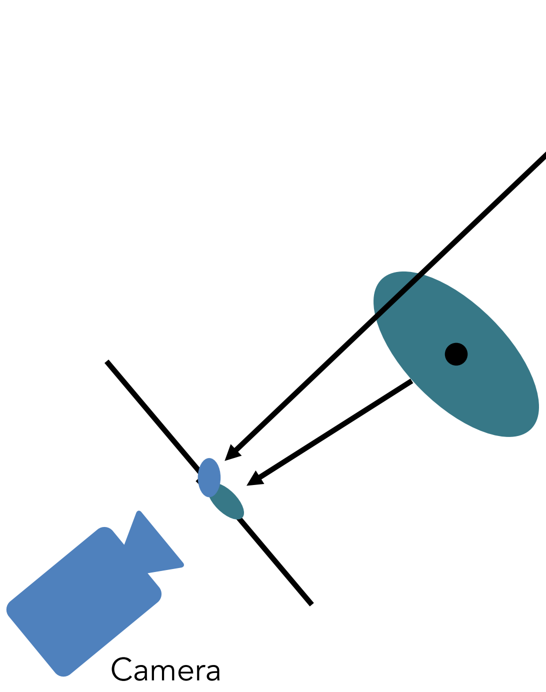
versus

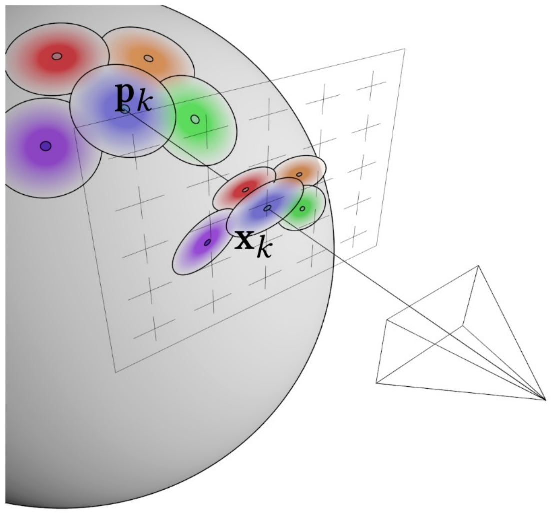
$$C = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \qquad (3)$$

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.

Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

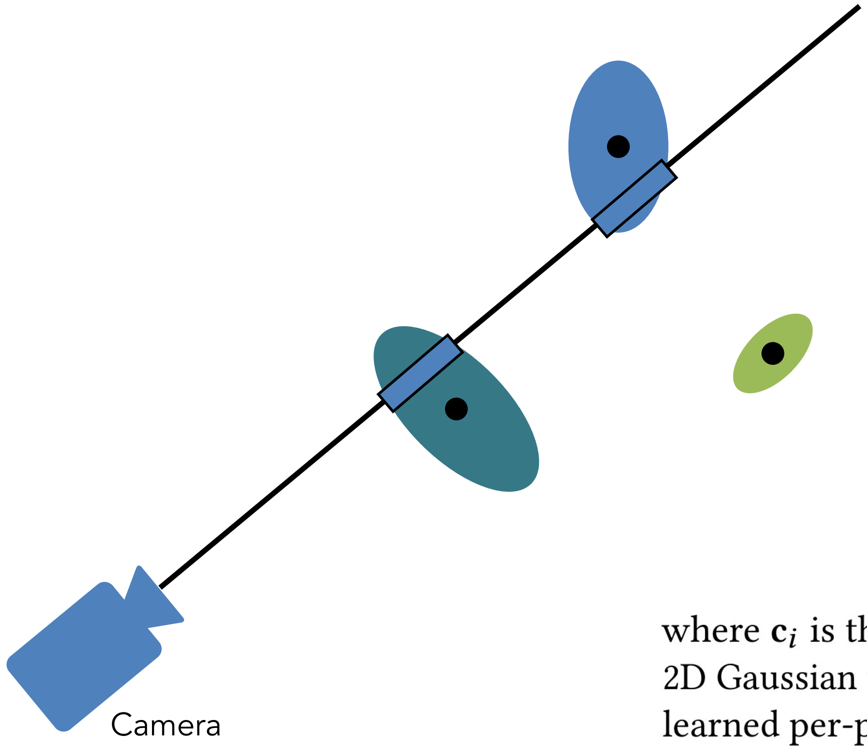
Projected 3D Gaussian makes 2D Gaussian!





Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Any problems for inverse graphics, though…?

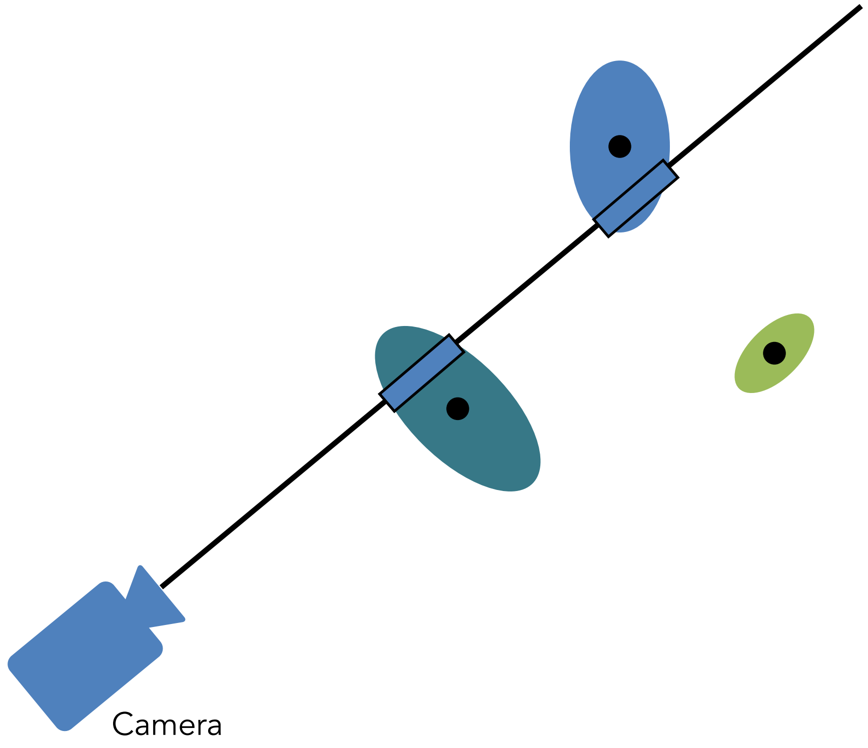


$$C = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \qquad (3)$$

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.

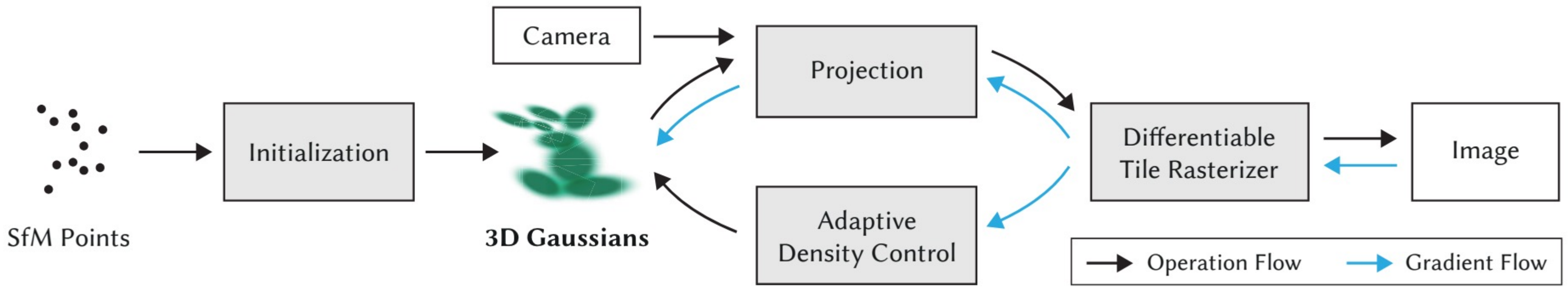
Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Problem: Local minima…



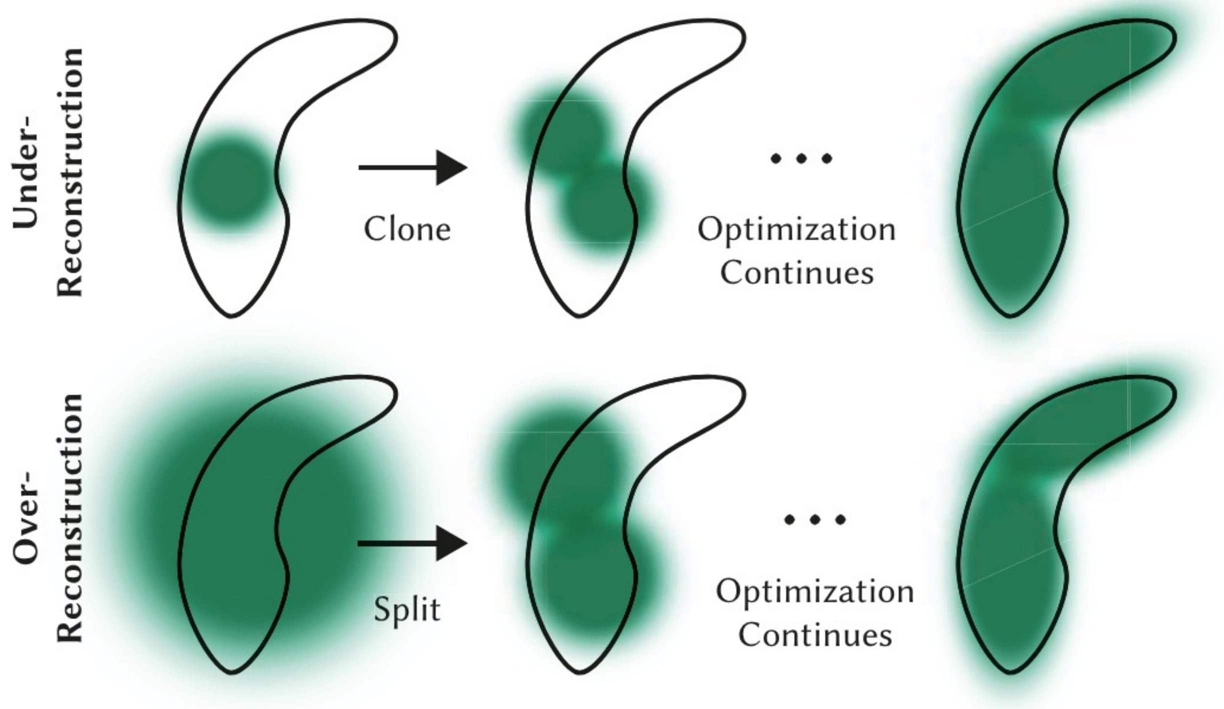
Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Fix 1: Start from SFM point cloud.



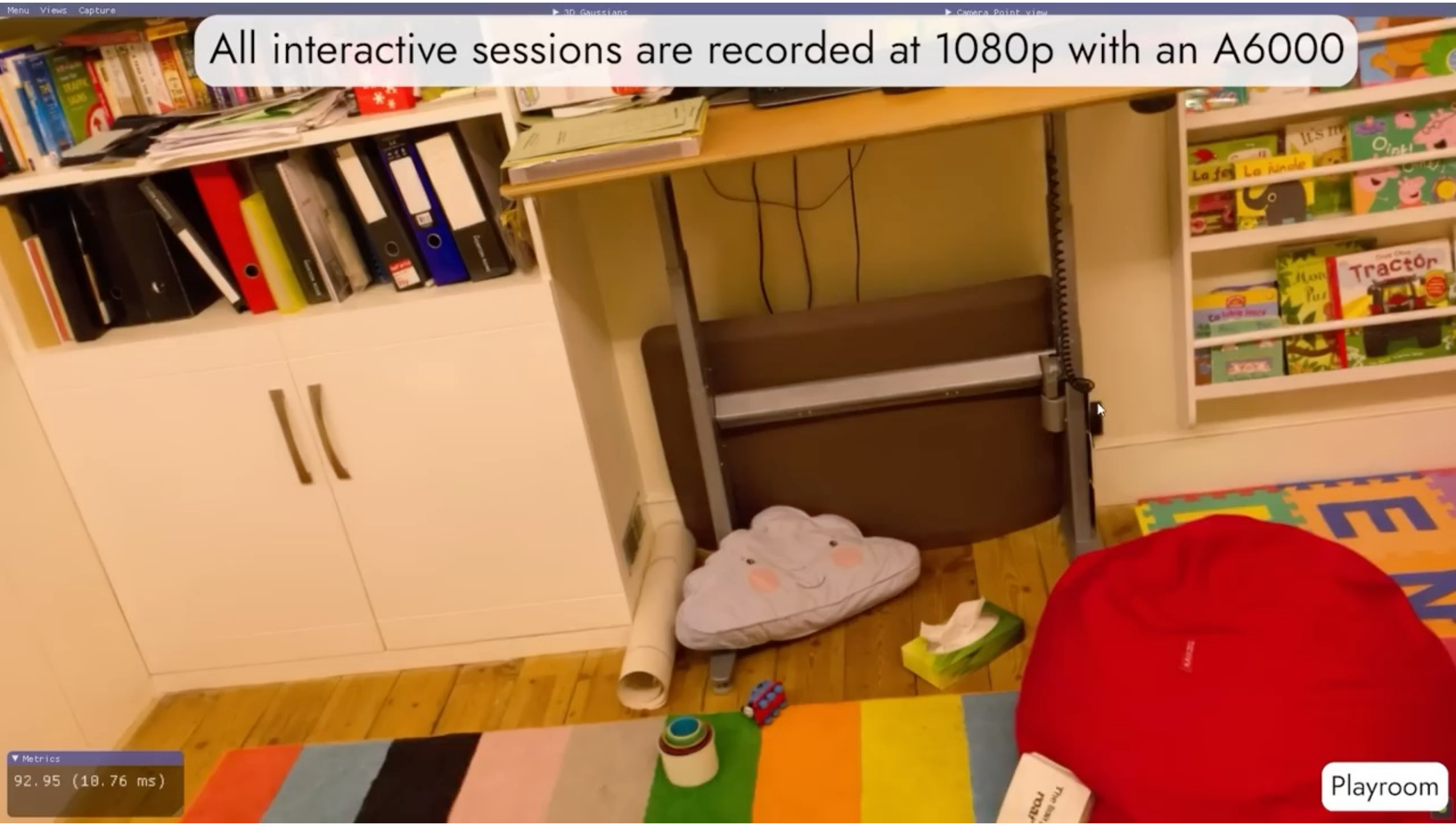
Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

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



Slide adopted from 6.S980 – ML for Inverse Graphics – Vincent Sitzmann

Timelapse of the Optimization
(NeRF-Synthetic Dataset)



All interactive sessions are recorded at 1080p with an A6000

CS231A

Computer Vision:

From 3D Reconstruction to Recognition

Next lecture:

Guest Lecture by Francis Engelmann on Open-World 3D Scene Understanding