ADVANCES IN COMPUTER VISION

MODULE 1:
Geometry

NEURAL RENDERING AND FIELD REPRESENTATIONS



Prof. Vincent Sitzmann
Prof. Frédo Durand

Check piazza for Final project pdf



EECS Graduate Student Survey

- EECS Visiting Committee 2026 PhD Student Review Survey



Instant NGP

Instant Neural Graphics Primitives with a Multiresolution Hash Encoding

THOMAS MÜLLER, NVIDIA, Switzerland
ALEX EVANS, NVIDIA, United Kingdom
CHRISTOPH SCHIED, NVIDIA, USA
ALEXANDER KELLER, NVIDIA, Germany

https://nvlabs.github.io/instant-ngp

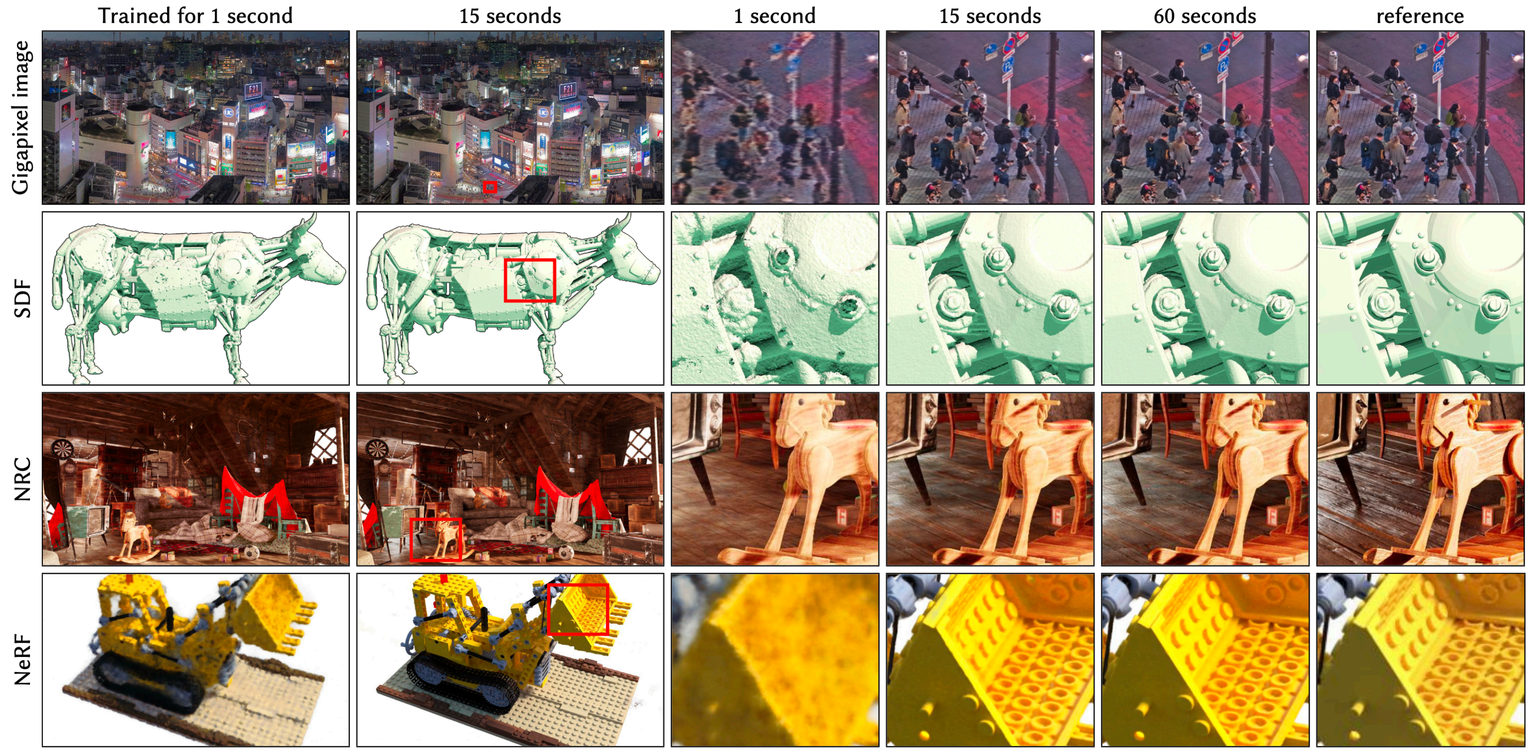


Fig. 1. We demonstrate instant training of neural graphics primitives on a single GPU for multiple tasks. In *Gigapixel image* we represent a gigapixel image by a neural network. *SDF* learns a signed distance function in 3D space whose zero level-set represents a 2D surface. Neural radiance caching (*NRC*) [Müller et al. 2021] employs a neural network that is trained in real-time to cache costly lighting calculations. Lastly, *NeRF* [Mildenhall et al. 2020] uses 2D images and their camera poses to reconstruct a volumetric radiance-and-density field that is visualized using ray marching. In all tasks, our encoding and its efficient implementation provide clear benefits: rapid training, high quality, and simplicity. Our encoding is task-agnostic: we use the same implementation and hyperparameters across all tasks and only vary the hash table size which trades off quality and performance. Photograph ©Trevor Dobson (CC BY-NC-ND 2.0)

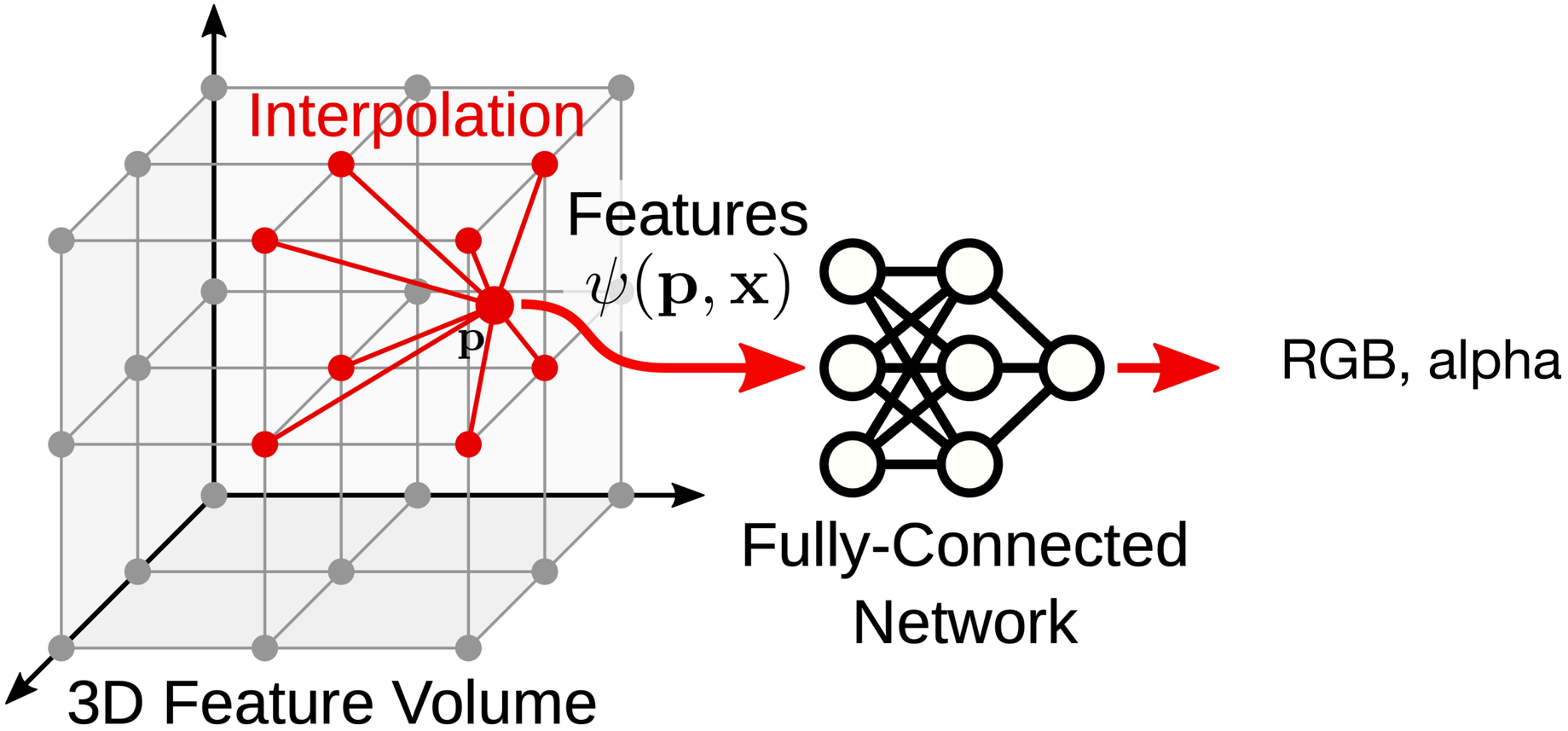
https://nvlabs.github.io/instant-ngp/

Basic idea: single dense grid

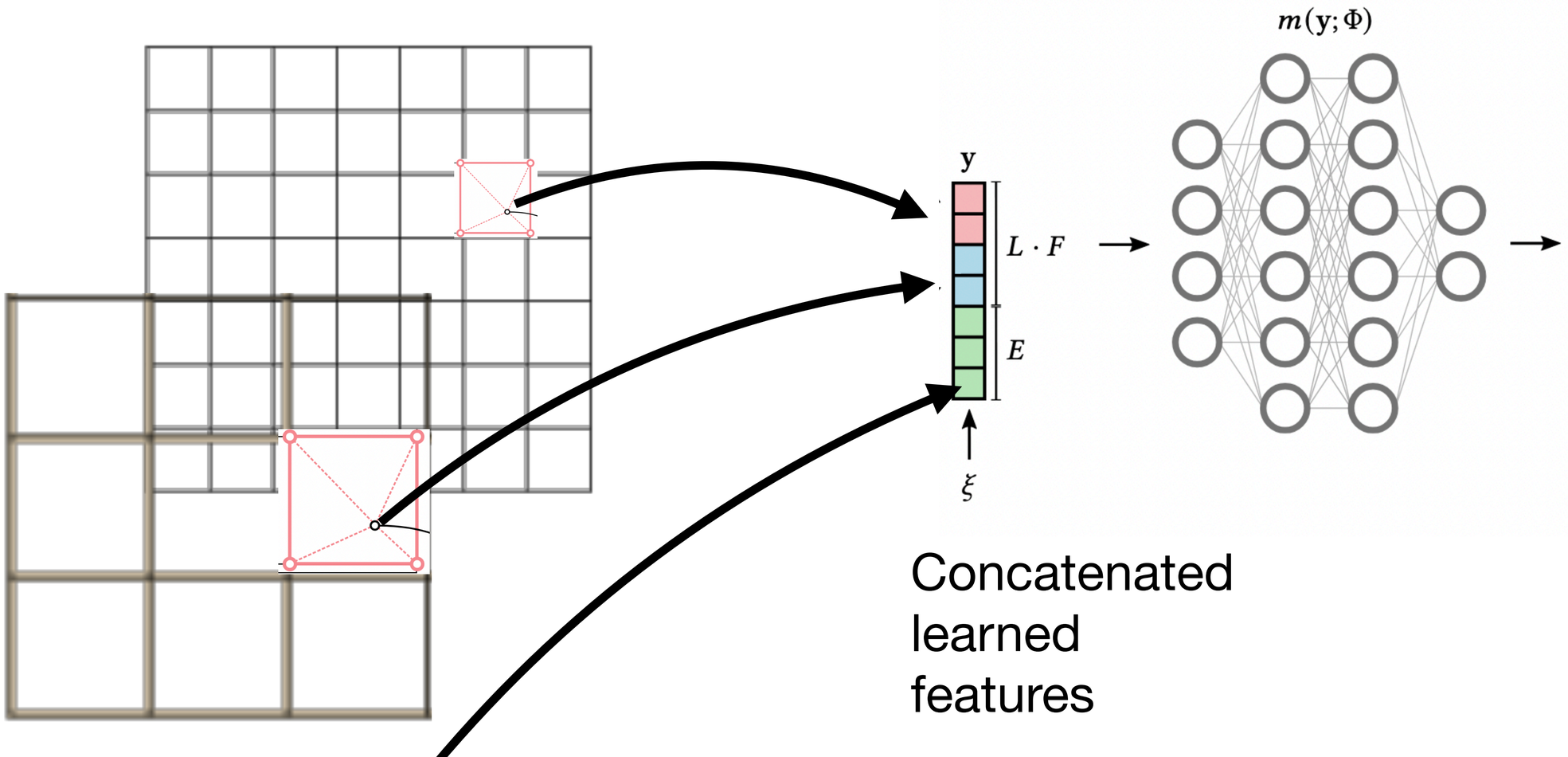
Store grid of **trained** features

Trained MLP takes features, returns RGB and opacity

Joint training of MLP and spatial grid of features
(overfit to, e.g. hundreds of input views)



Extra trick: Multi resolutions dense grids

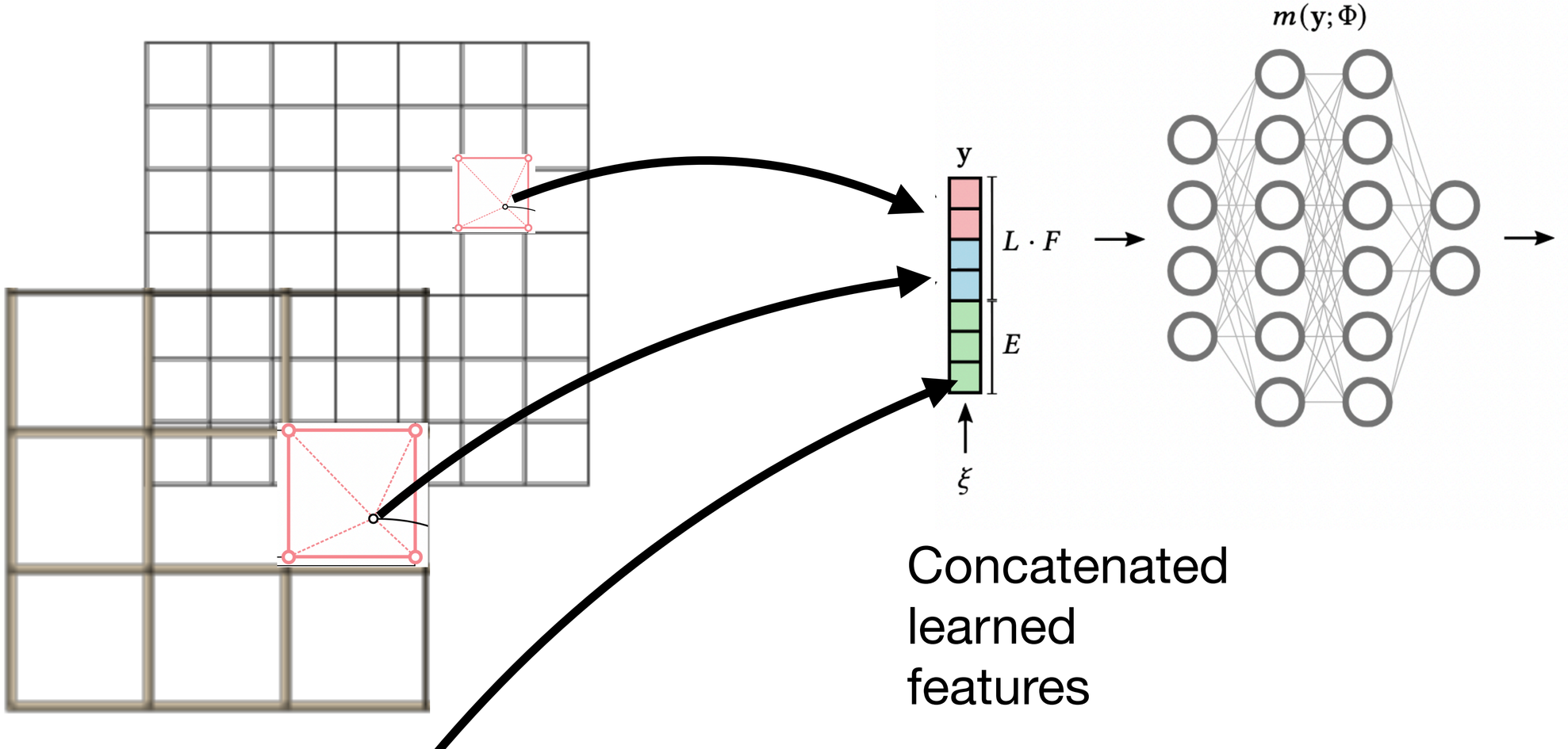


Concatenated
learned
features

RGB, alpha

Instant NGP: scalability through hashing

- Virtually the grids are dense
- But they are stored via hash table

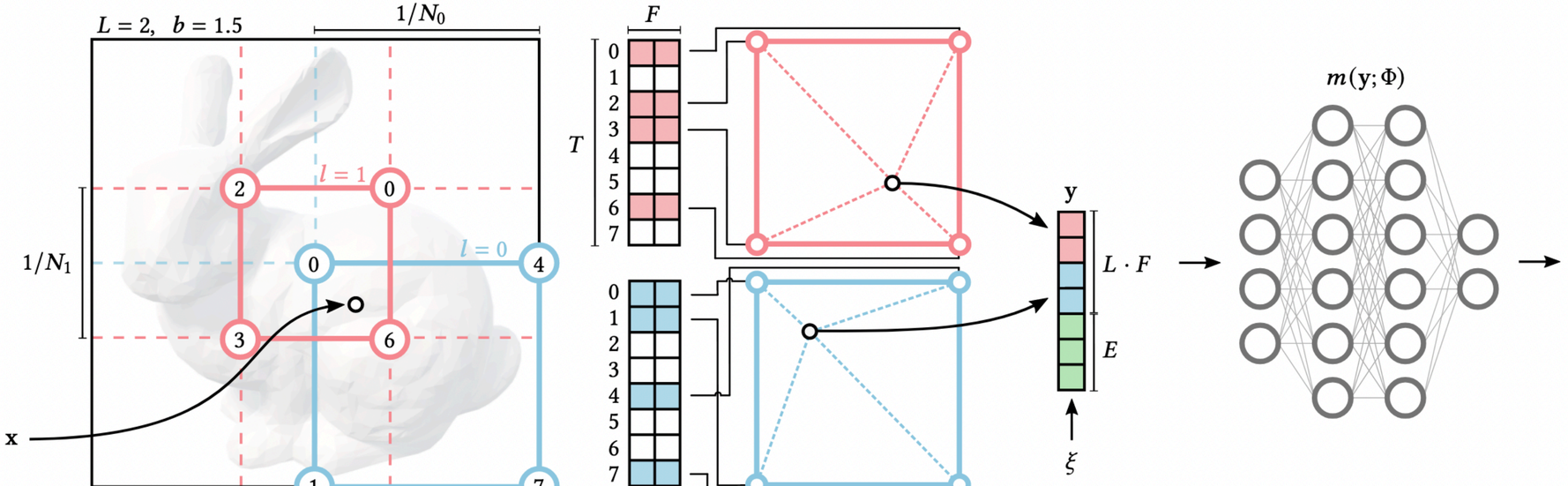


Concatenated learned features

RGB, alpha

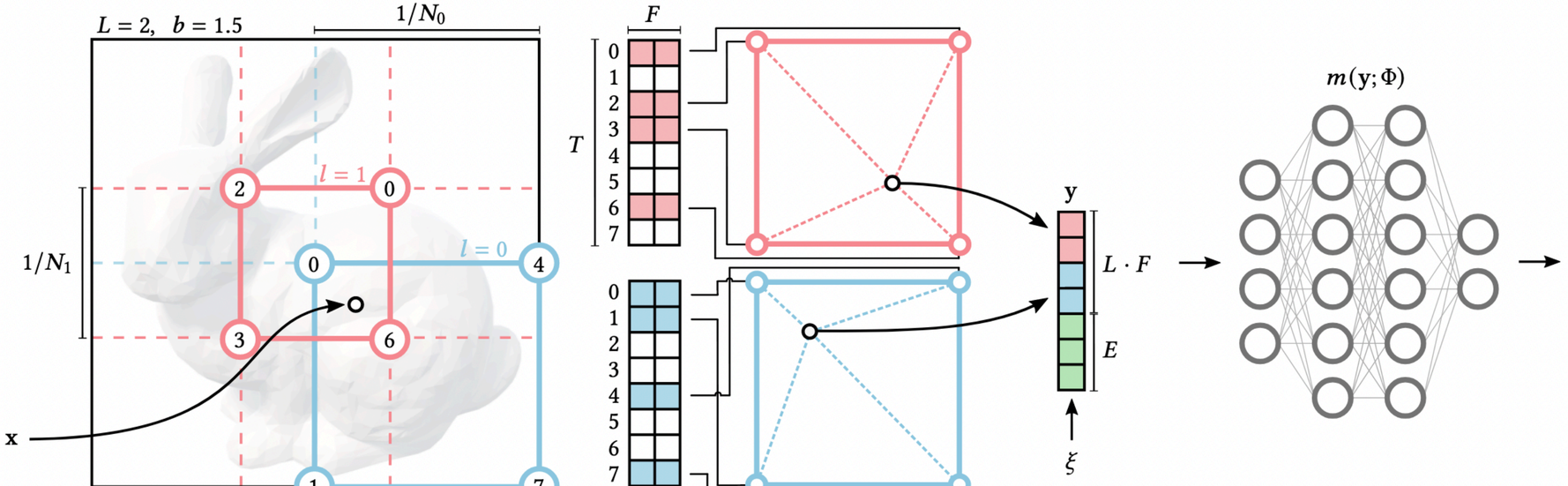
Forward pass

- Full rendering: ray march & query at x,y,z ; accumulate RGB and alpha
  - Core operation: input: x,y,z. Output: RGB, alpha



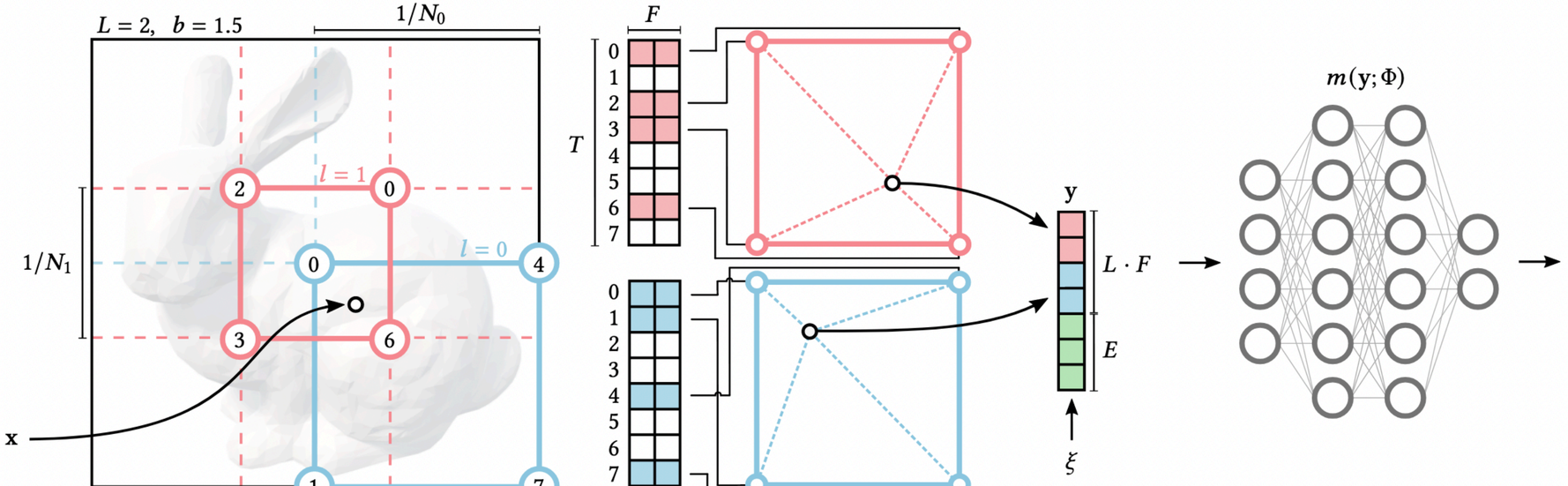
Forward pass

- Full rendering: ray march & query at x,y,z ; accumulate RGB and alpha
  - Core operation: input: x,y,z. Output: RGB, alpha
    - For each grid level
      - Query 8 neighbors in hash table, get neural features
      - Trilinear interpolation -> interpolated feature
    - Concatenate features from all resolutions into one big neural feature
    - Run through MLP, get RGB & alpha



Training

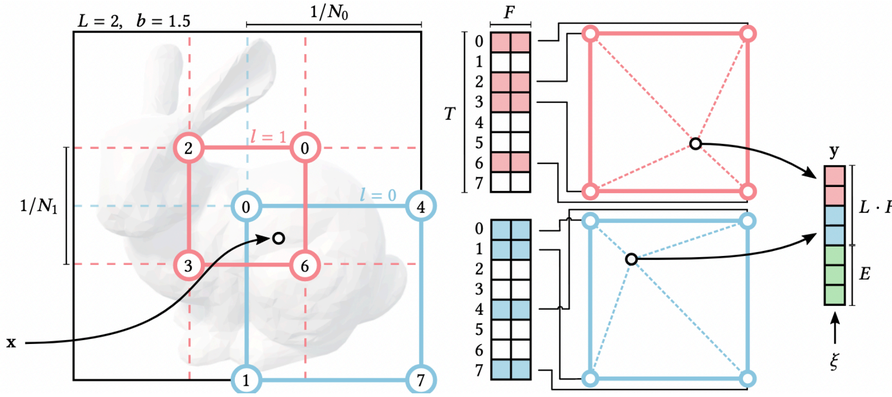
- Jointly train neural features (at all locations for all grid levels) and MLP weights
  - Collisions will contribute gradients for a given feature coming from to different x, y, z
- Supervision: reconstruction input views (or whatever field you are working on).



Instant NGP

- **L levels of sparse/hashed grids at different resolutions stores feature vectors**
  - Each level is independent. Features will be concatenated
  - Each level has fixed size
- **MLP computes field given interpolated features**
- Hashing: to query a grid at location x:
  - Discretize x to 8 corners at grid resolution
  - For each corner, read feature in hash table
  - Big trick for performance : live with collisions, hope the MLP will handle it.
  - One thing that helps: collisions are uncorrelated at different scales

Showing only 2 levels



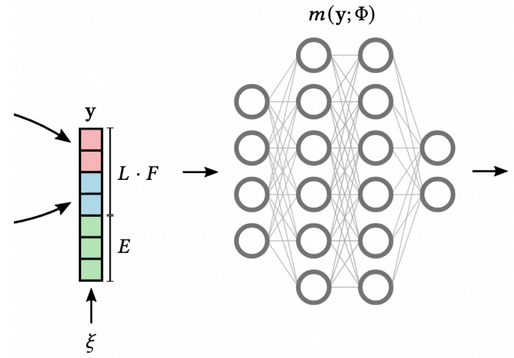


Table 1. Hash encoding parameters and their ranges in our results. Only the hash table size T and max. resolution $N_{\max}$ need to be tuned to the task.

Parameter	Symbol	Value
Number of levels	L	16
Max. entries per level (hash table size)	T	2^{14} to 2^{24}
Number of feature dimensions per entry	F	2
Coarsest resolution	$N_{\min}$	16
Finest resolution	$N_{\max}$	512 to 524288

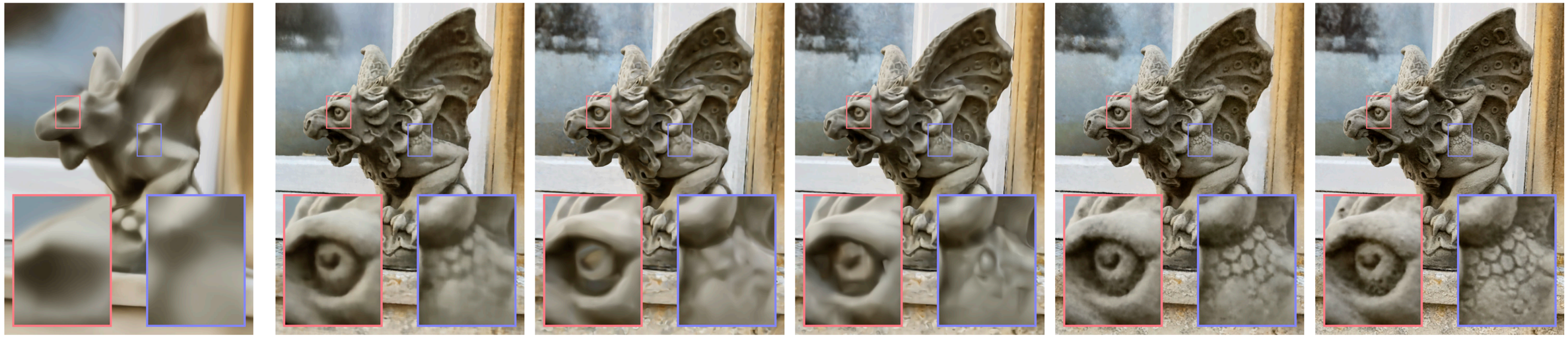
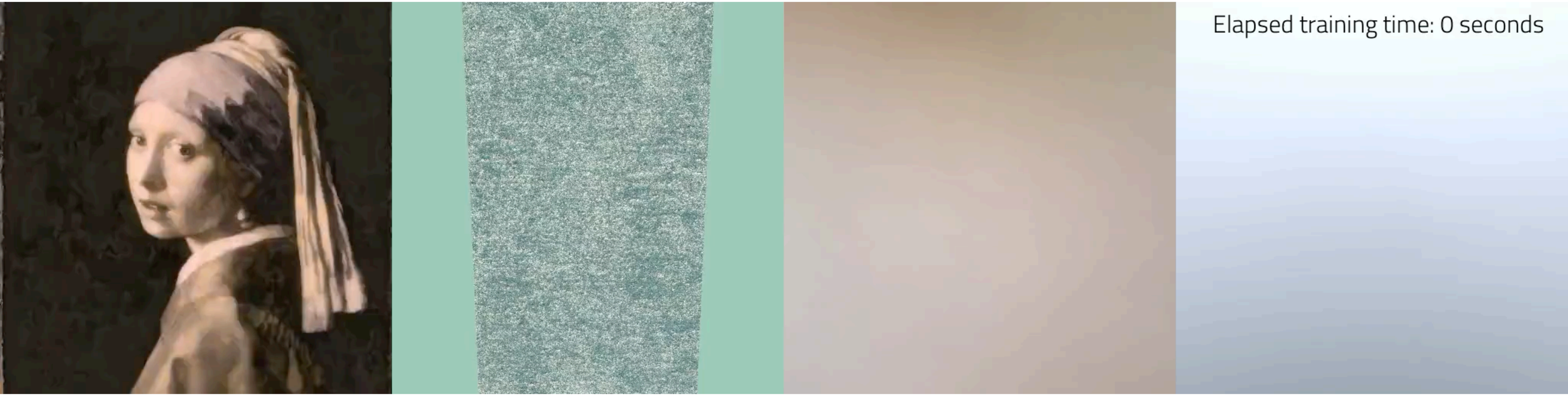


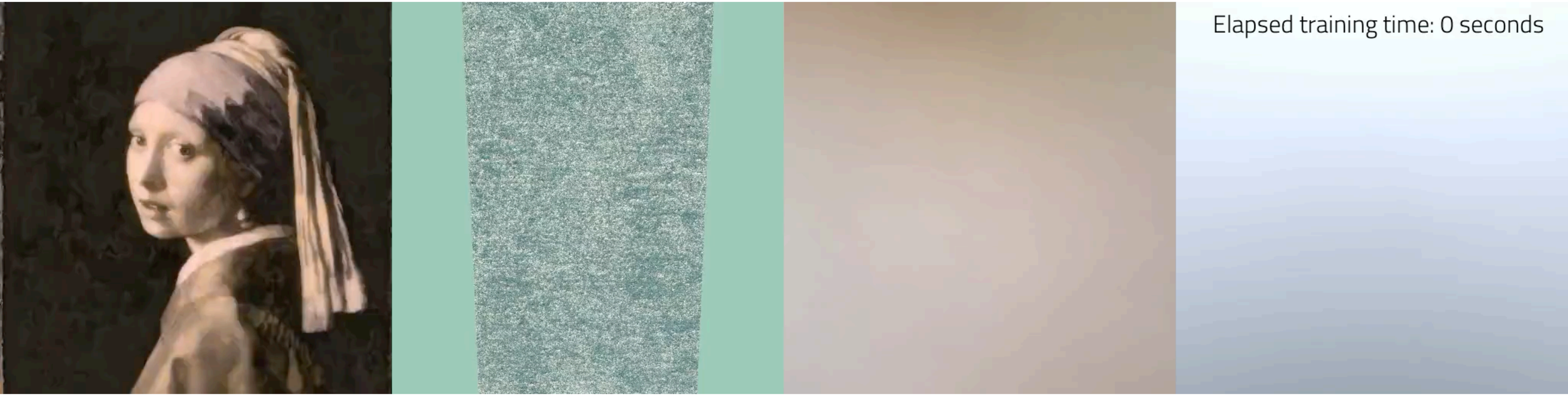
Fig. 2. A demonstration of the reconstruction quality of different encodings and parametric data structures for storing trainable feature embeddings. Each configuration was trained for 11 000 steps using our fast NeRF implementation (Section 5.4), varying only the input encoding and MLP size. The number of trainable parameters (MLP weights + encoding parameters), training time and reconstruction accuracy (PSNR) are shown below each image. Our encoding **(e)** with a similar total number of trainable parameters as the frequency encoding configuration **(b)** trains over 8× faster, due to the sparsity of updates to the parameters and smaller MLP. Increasing the number of parameters **(f)** further improves reconstruction accuracy without significantly increasing training time.

Multi-scale Voxel Grids + Hash Table



Instant Neural Graphics Primitives with a Multiresolution Hash Encoding, Müller et al. 2022

Multi-scale Voxel Grids + Hash Table



Instant Neural Graphics Primitives with a Multiresolution Hash Encoding, Müller et al. 2022

Questions?

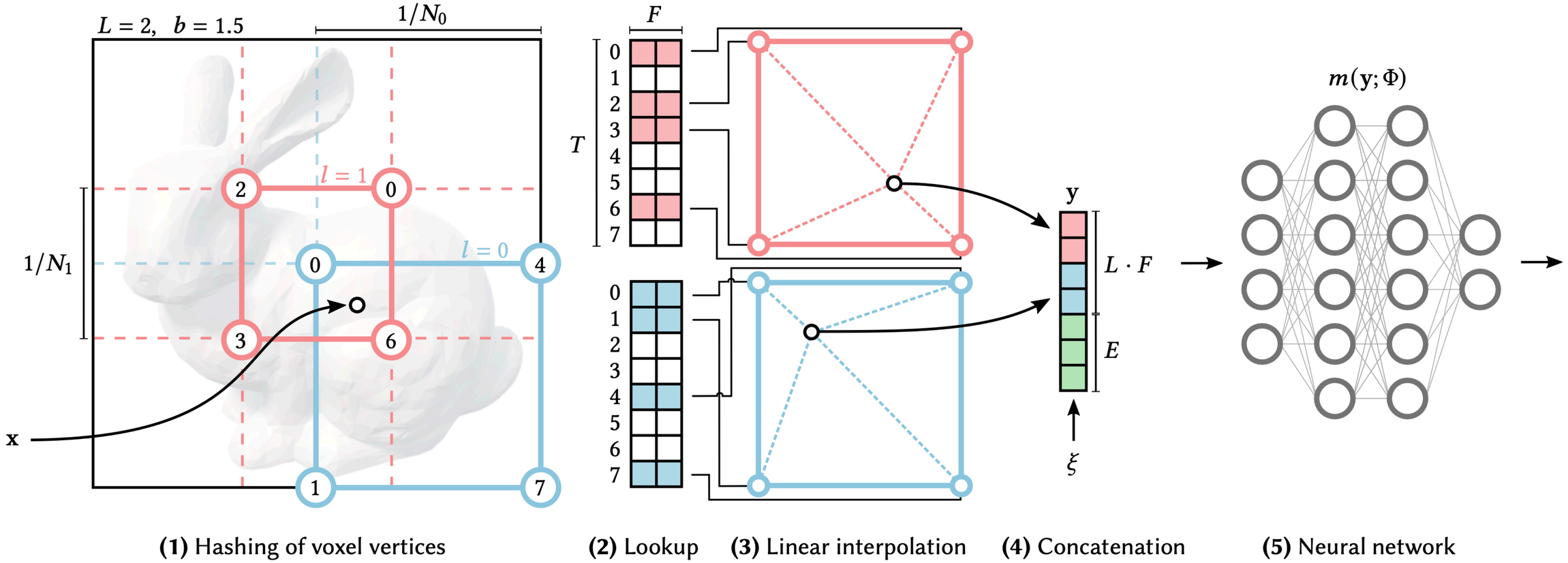
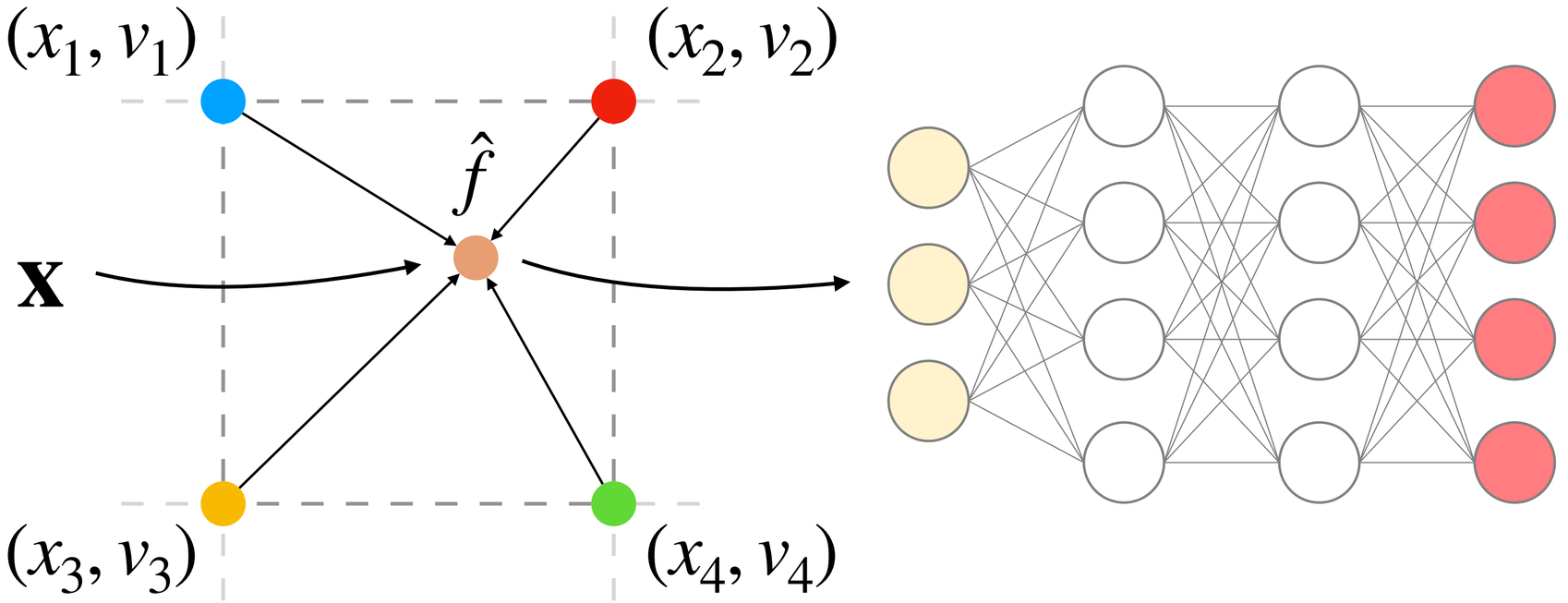


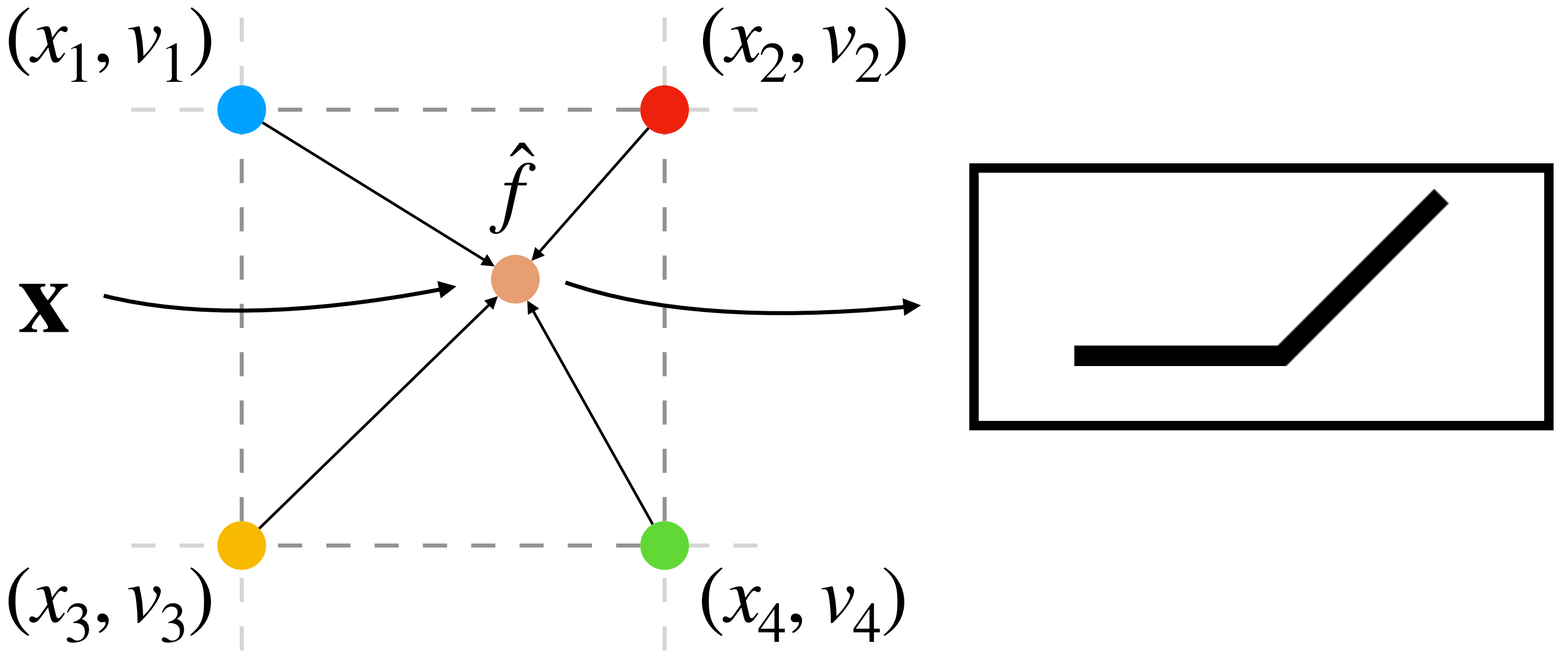
Fig. 3. Illustration of the multiresolution hash encoding in 2D. **(1)** for a given input coordinate $\mathbf{x}$, we find the surrounding voxels at L resolution levels and assign indices to their corners by hashing their integer coordinates. **(2)** for all resulting corner indices, we look up the corresponding F-dimensional feature vectors from the hash tables θ_l and **(3)** linearly interpolate them according to the relative position of $\mathbf{x}$ within the respective l-th voxel. **(4)** we concatenate the result of each level, as well as auxiliary inputs $\xi \in \mathbb{R}^E$, producing the encoded MLP input $y \in \mathbb{R}^{LF+E}$, which **(5)** is evaluated last. To train the encoding, loss gradients are backpropagated through the MLP **(5)**, the concatenation **(4)**, the linear interpolation **(3)**, and then accumulated in the looked-up feature vectors.

ReLU Fields: How much MLP do you need?



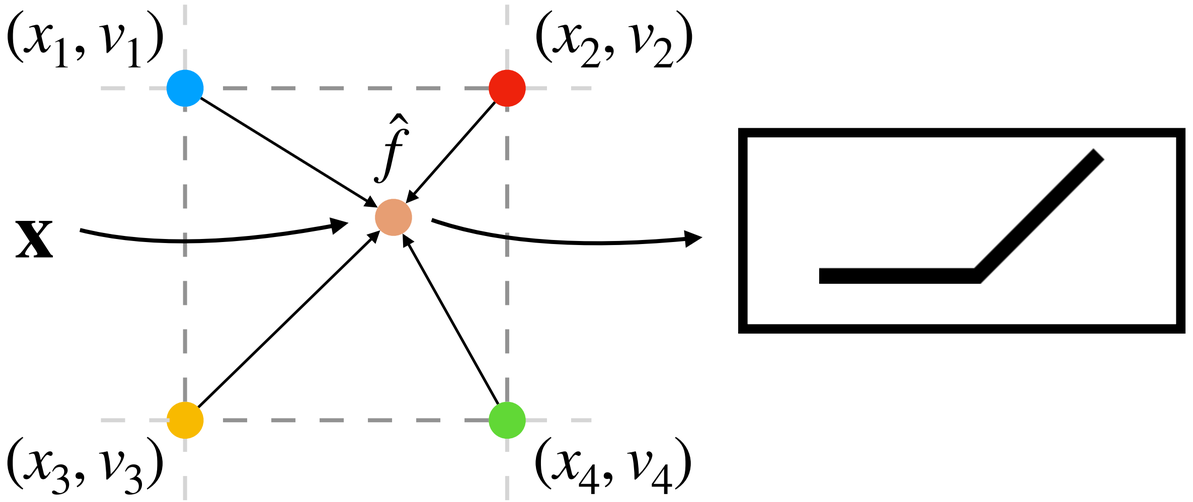
ReLU Fields: The Little Non-linearity That Could (Karnewar et al. 2022)

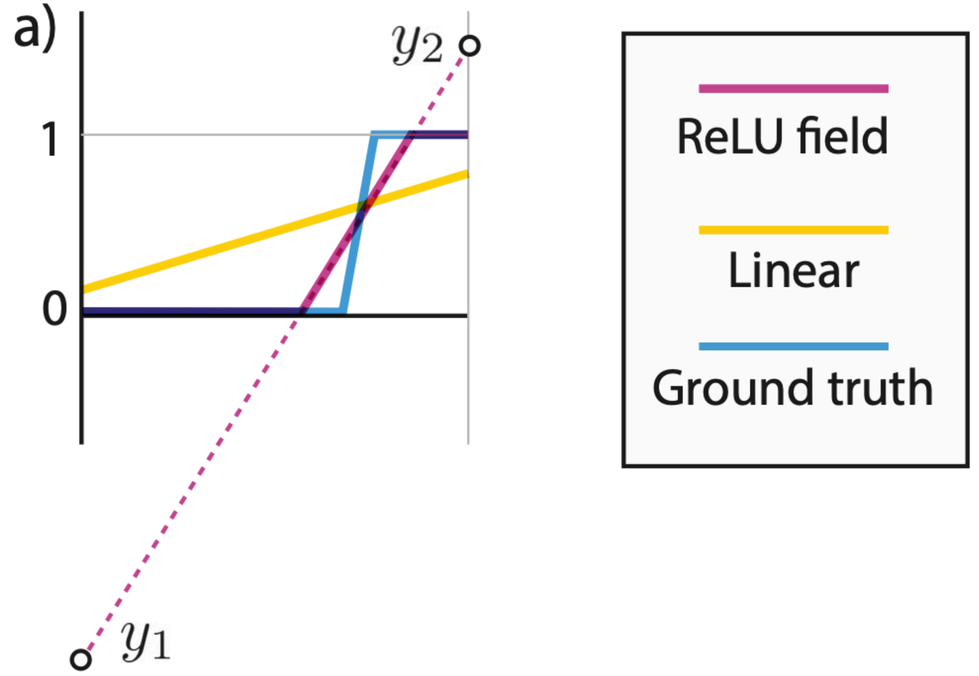
ReLU Fields: How much MLP do you need?



ReLU Fields: The Little Non-linearity That Could (Karnewar et al. 2022)

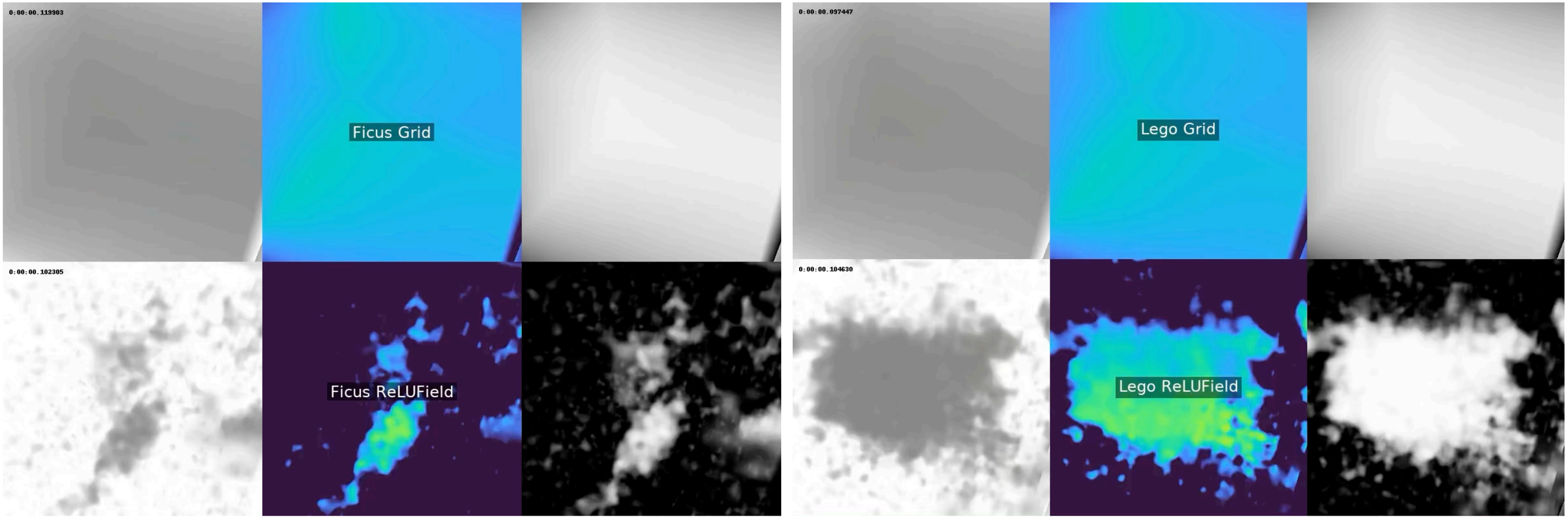
ReLU Fields: How much MLP do you need?





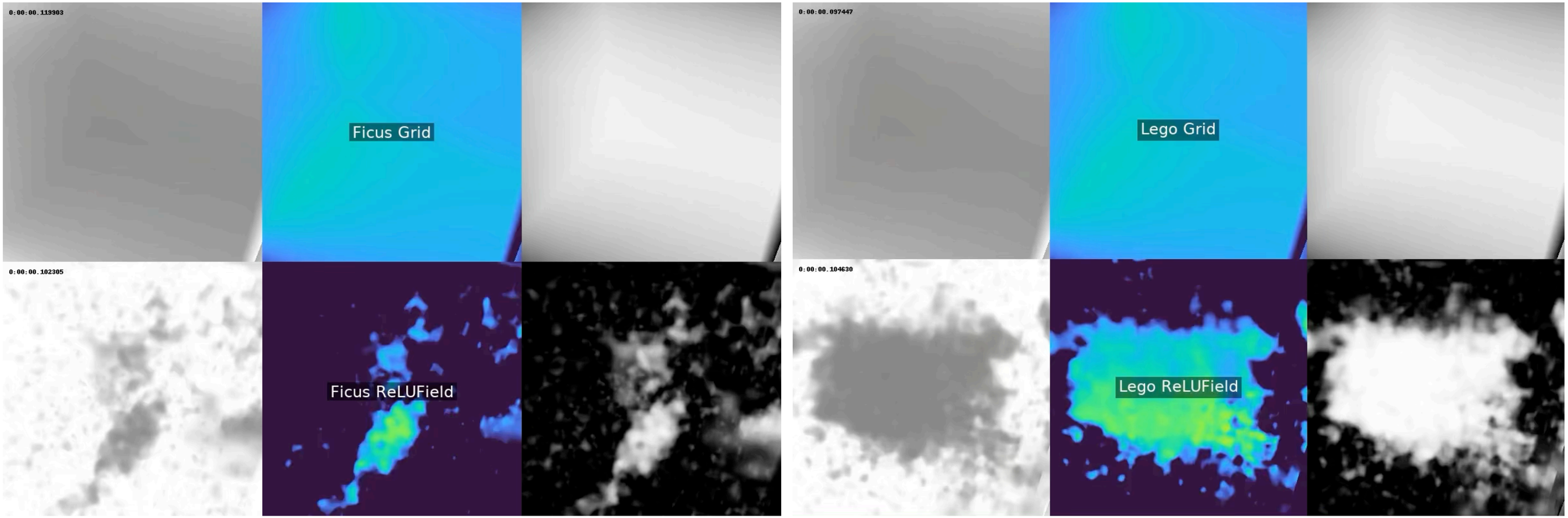
ReLU Fields: The Little Non-linearity That Could (Karnewar et al. 2022)

ReLU Fields: How much MLP do you need?



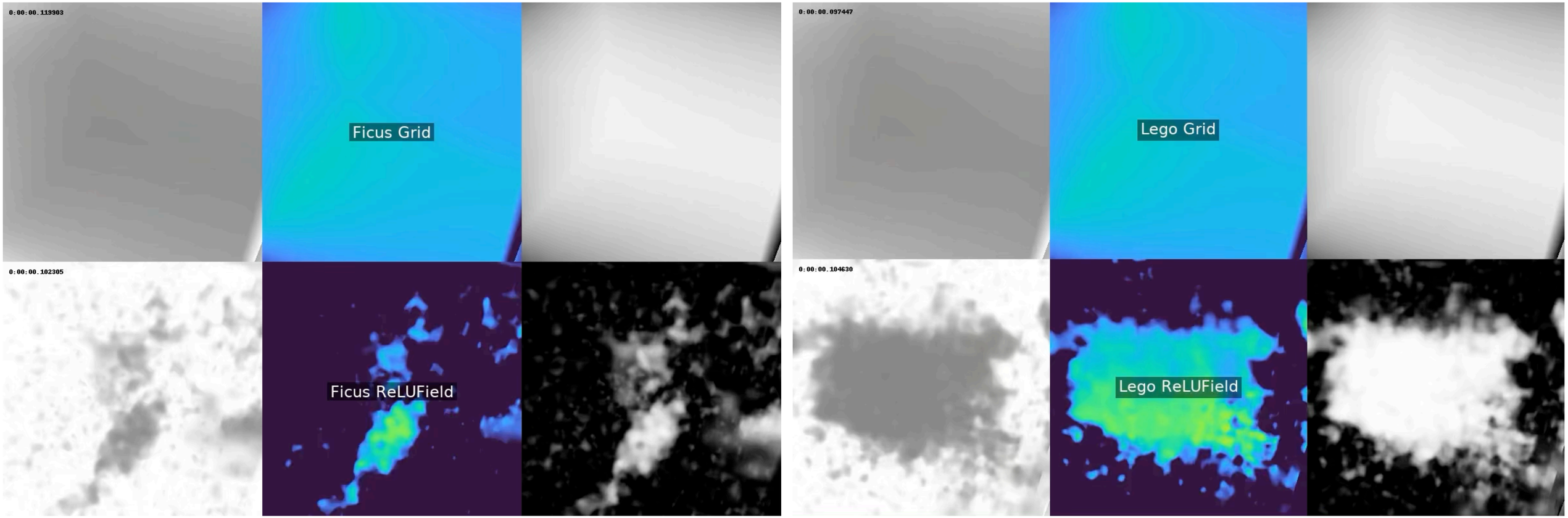
ReLU Fields: The Little Non-linearity That Could (Karnewar et al. 2022)

ReLU Fields: How much MLP do you need?



ReLU Fields: The Little Non-linearity That Could (Karnewar et al. 2022)

ReLU Fields: How much MLP do you need?



ReLU Fields: The Little Non-linearity That Could (Karnewar et al. 2022)

Discussion: ReLu vs sine waves

- When using global positional encoding, sine waves helps a lot
  - Although note that RBFs (bunch of radially-symmetric spatial blobs) help a lot too
  - When used as activation non-linearity, sine waves also help with gradients and higher-order derivatives.
- When subdividing space with a grid, ReLu suffices
  - The grid subdivision gets us close to a primal basis
- My moral of the story: need to provide more structure about the input domain and reduce the range of values, either using positional encoding or grid subdivision

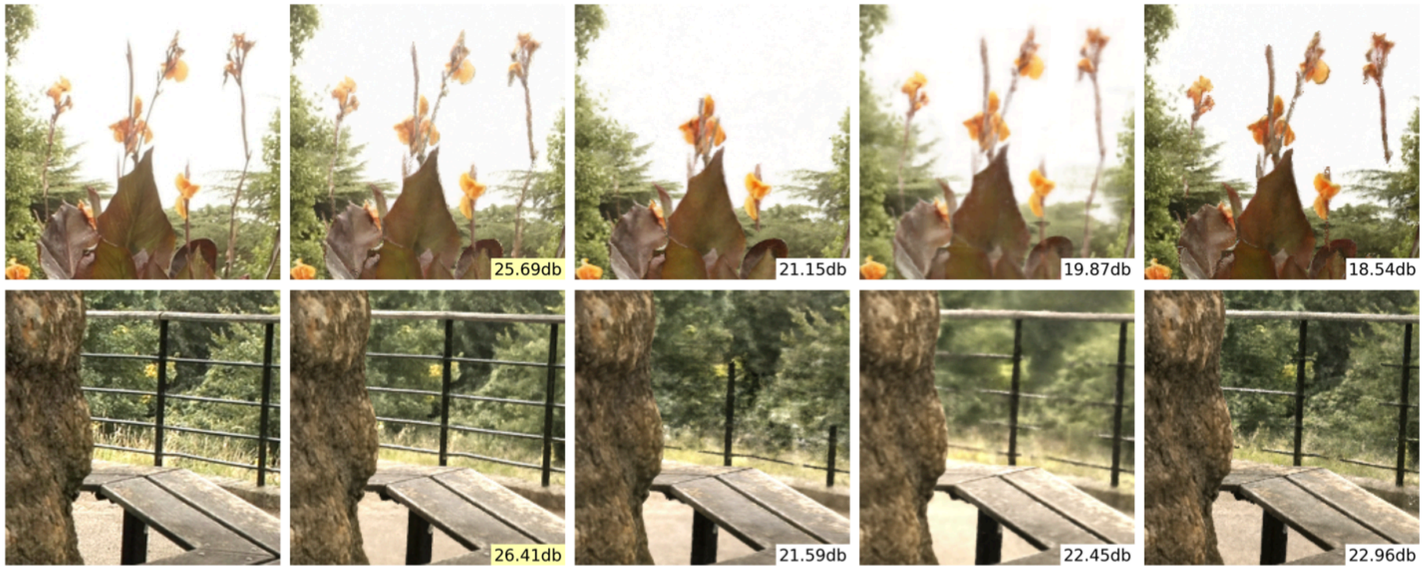
State of the art in NERF

Zip NERF

Instant NGP speed
+ antialiasing

Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields

Jonathan T. Barron Ben Mildenhall Dor Verbin Pratul P. Srinivasan Peter Hedman
Google Research

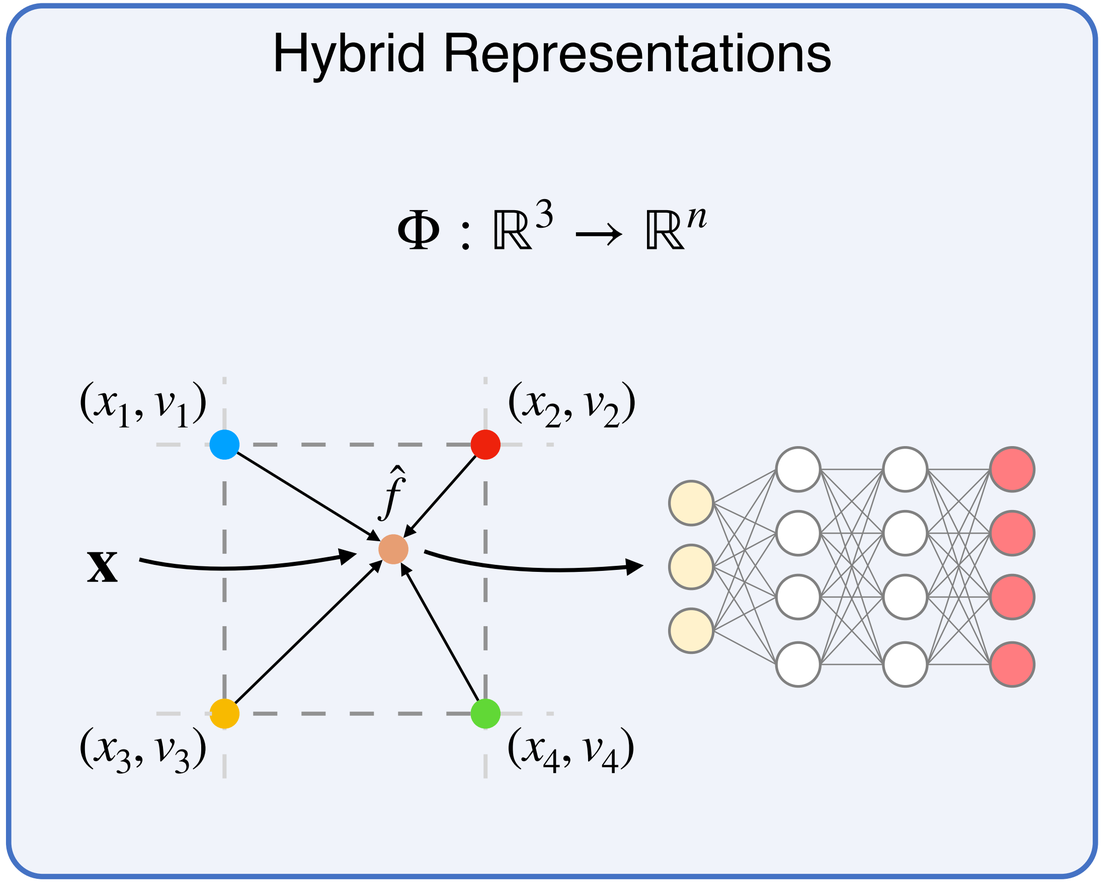


(a) Ground Truth (b) Our Model (c) mip-NeRF 360 [3] (d) iNGP [21] (e) mip-NeRF 360 + iNGP

Figure 1: (a) Test-set images from the mip-NeRF 360 dataset [3] with renderings from (b) our model and (c, d, e) three state-of-the-art baselines. Our model accurately recovers thin structures and finely detailed foliage, while the baselines either oversmooth or exhibit aliasing in the form of jaggies and missing scene content. PSNR values for each patch are inset.

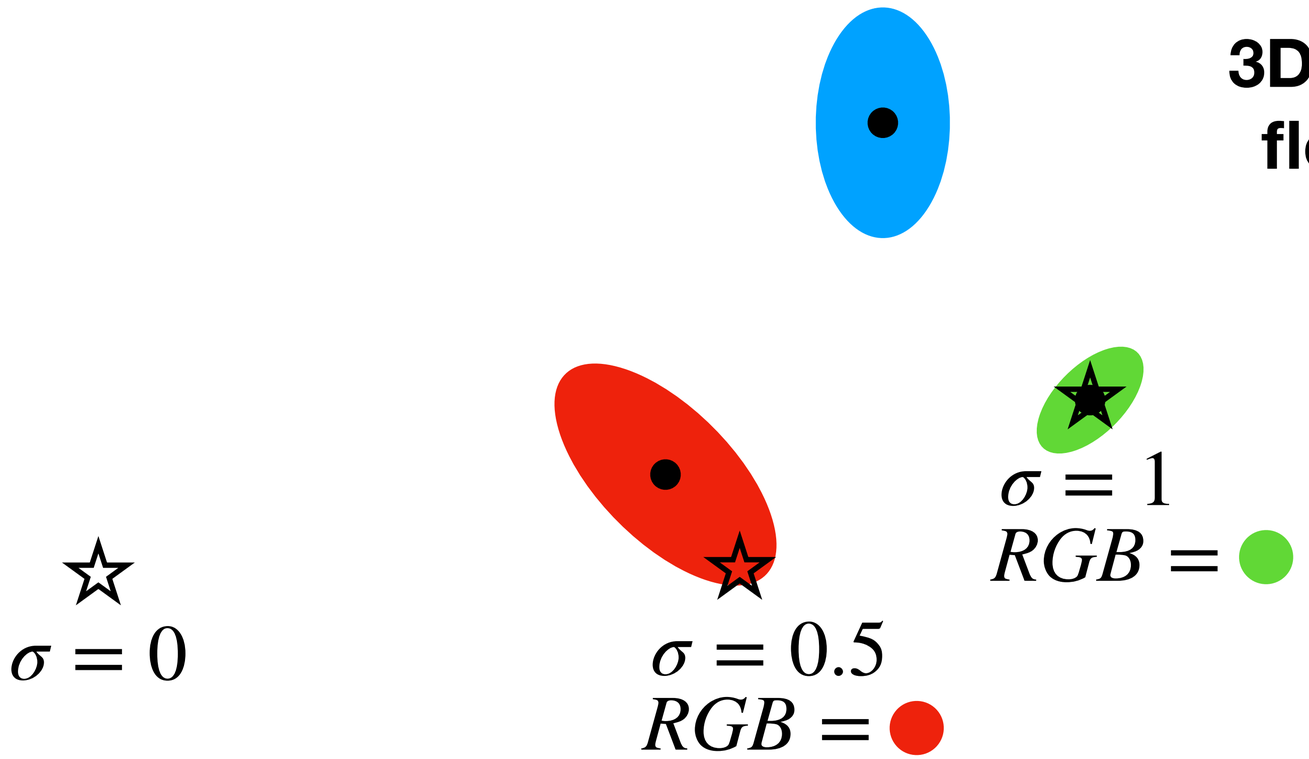
Hybrid Representations

- Very natural to trade off memory with compute.
- Neural network can locally “super resolve” discrete representation: Basically acts as interpolation kernel.
- Can be the best of both worlds: Fast and “continuous”.
- Once we’re local enough, ReLu is good after all.



Primitive-based approaches

Primitive-based representations



Anisotropic Volumetric 3D Gaussians



Final Rendering

3D Gaussian Visualization

Kerblk, Kopanas et al, 3D Gaussian Splatting for Real-Time Radiance Field Rendering

Anisotropic Volumetric 3D Gaussians



Final Rendering

3D Gaussian Visualization

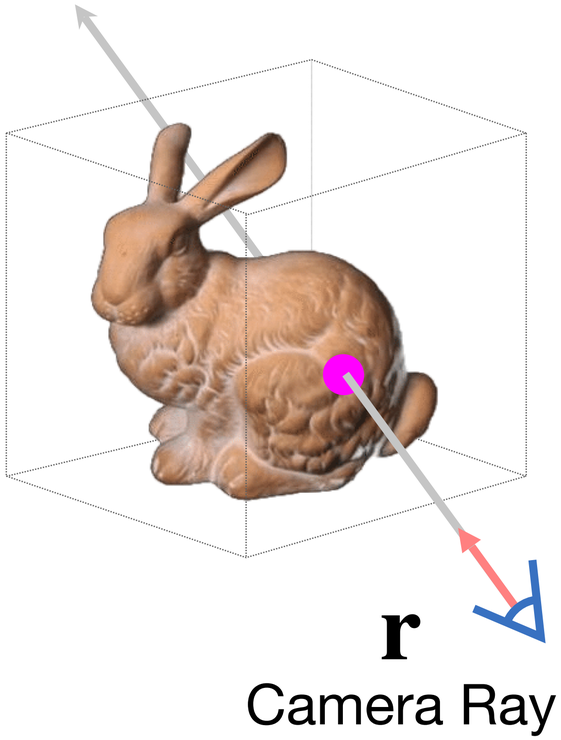
Kerblk, Kopanas et al, 3D Gaussian Splatting for Real-Time Radiance Field Rendering

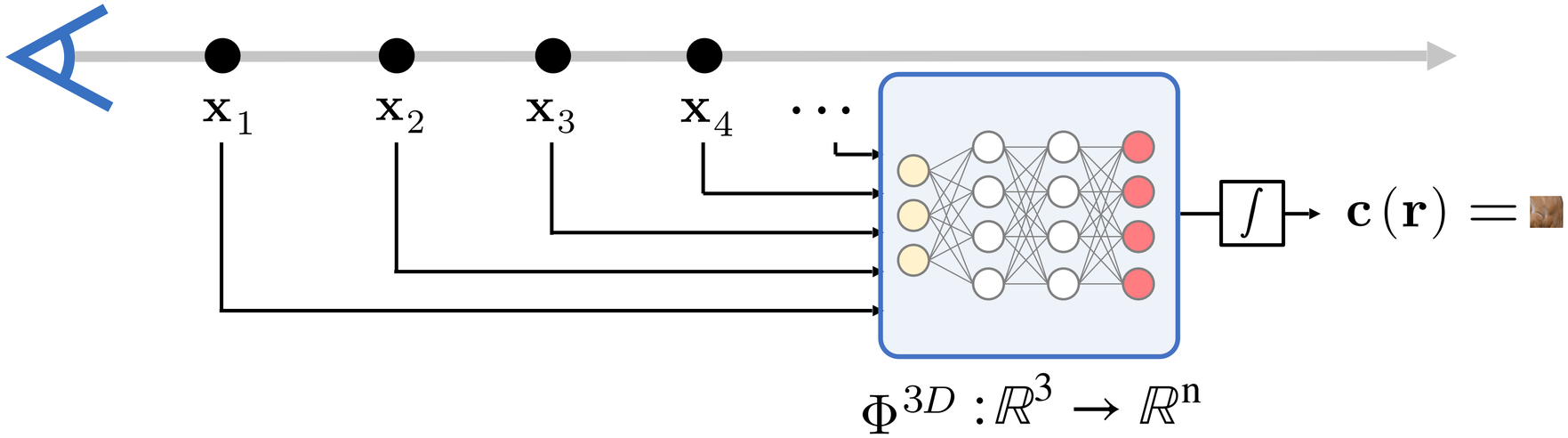
Light field approaches

3D Field issue

3D volumetric approaches such as NERF
require to sample the field many times along a ray (~192)
(more soon)

Also provides/requires strong 3D inductive bias

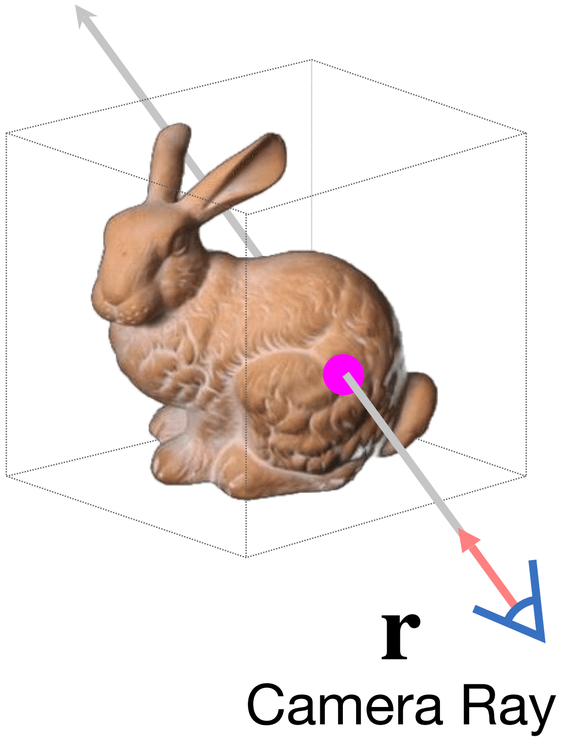




Light Field Networks

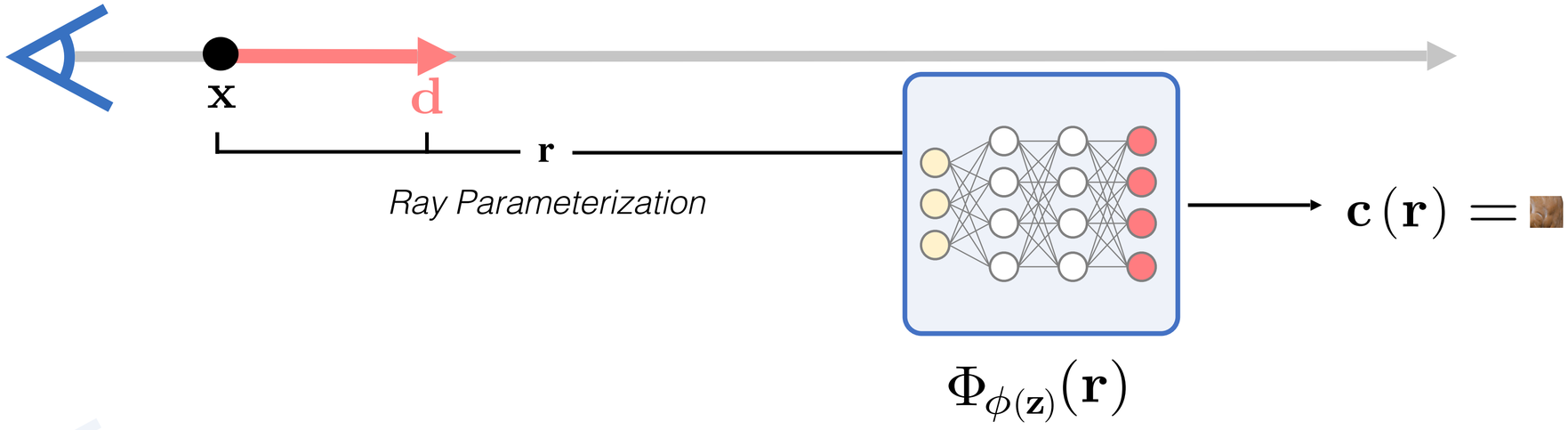
https://www.vincentsitzmann.com/lfns/

An Alternative (more direct) Scene Representation
Input: ray, MLP gives output: RGB



$\mathbf{r}$

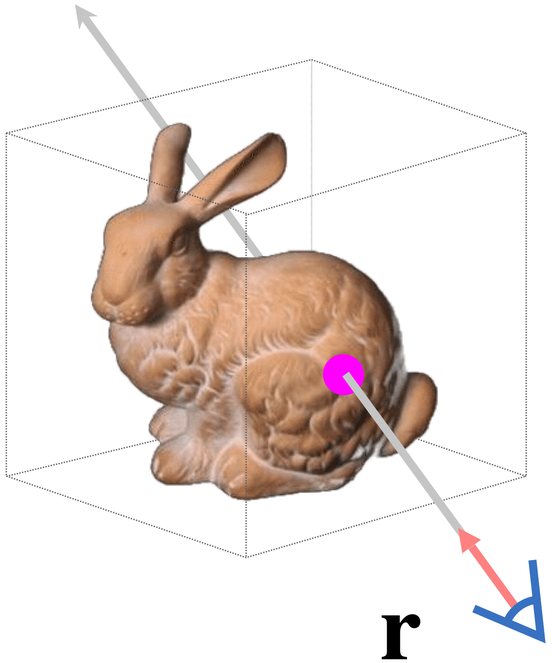
Camera Ray

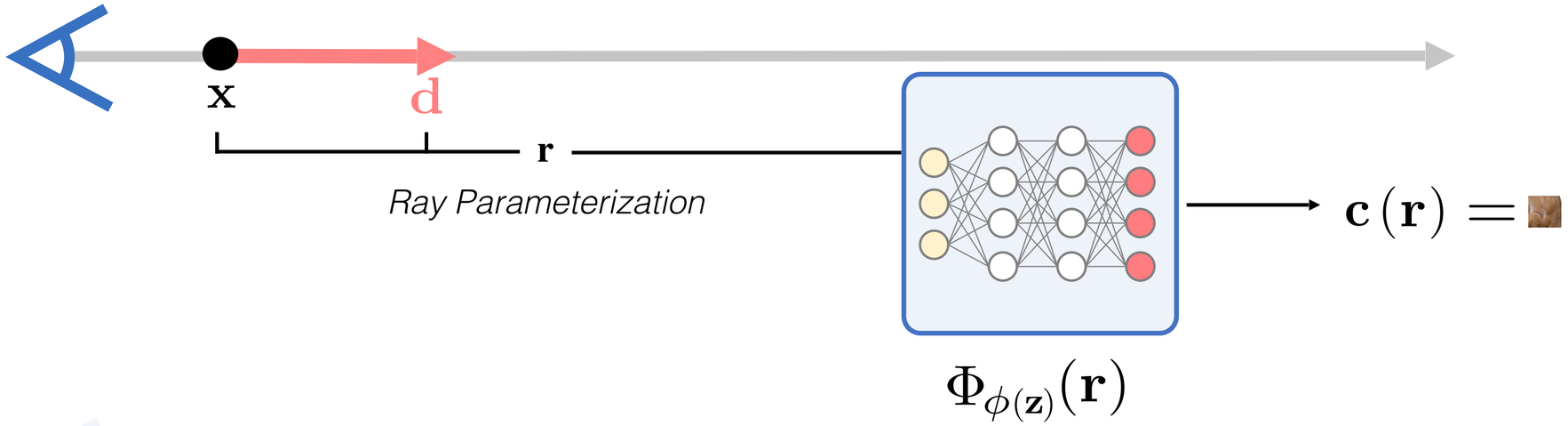


Light Field Networks

https://www.vincentsitzmann.com/lfns/

An Alternative (more direct) Scene Representation
Input: ray, MLP gives output: RGB



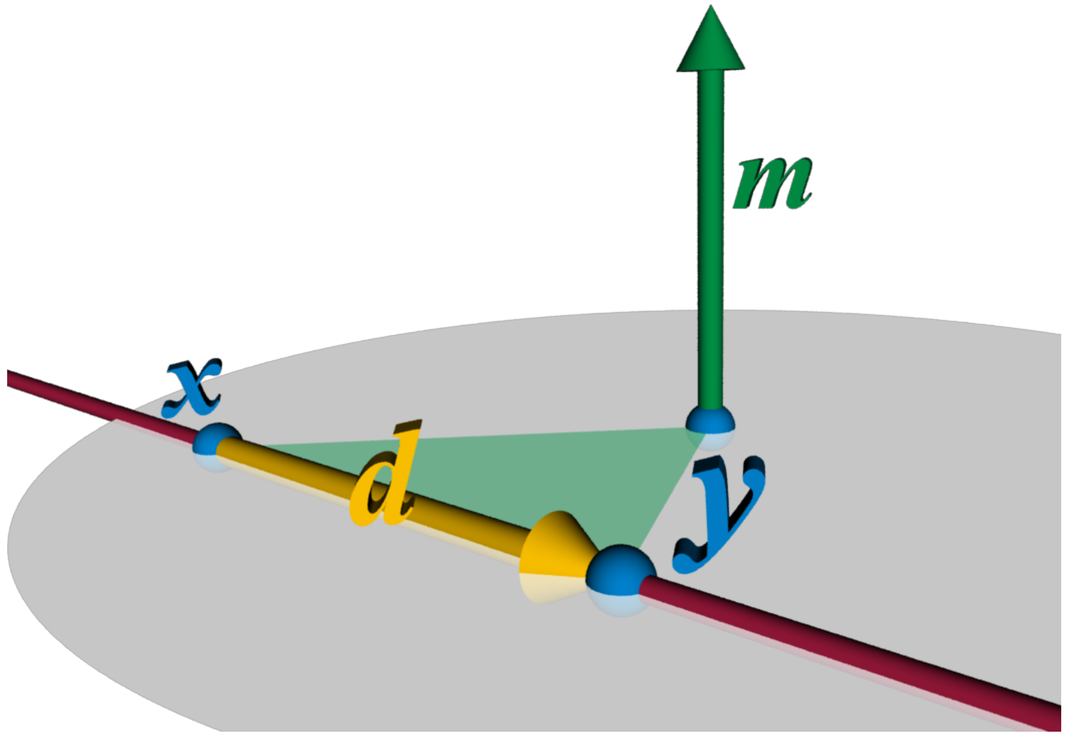


Pros: more directly what you want, more related to 2D images, less inductive bias

Cons: less inductive bias, not as naturally 3D consistent, hybrid approach not as easy

Interesting detail: line/ray representation

- Plucker coordinates
- https://en.wikipedia.org/wiki/Plücker_coordinates
- 6 coordinates even though the space of 3D lines is 4D



More at https://www.vincentsitzmann.com/lfns/

Light Field Networks: Neural Scene Representations with Single-Evaluation Rendering

Vincent Sitzmann[1,*]
sitzmann@mit.edu

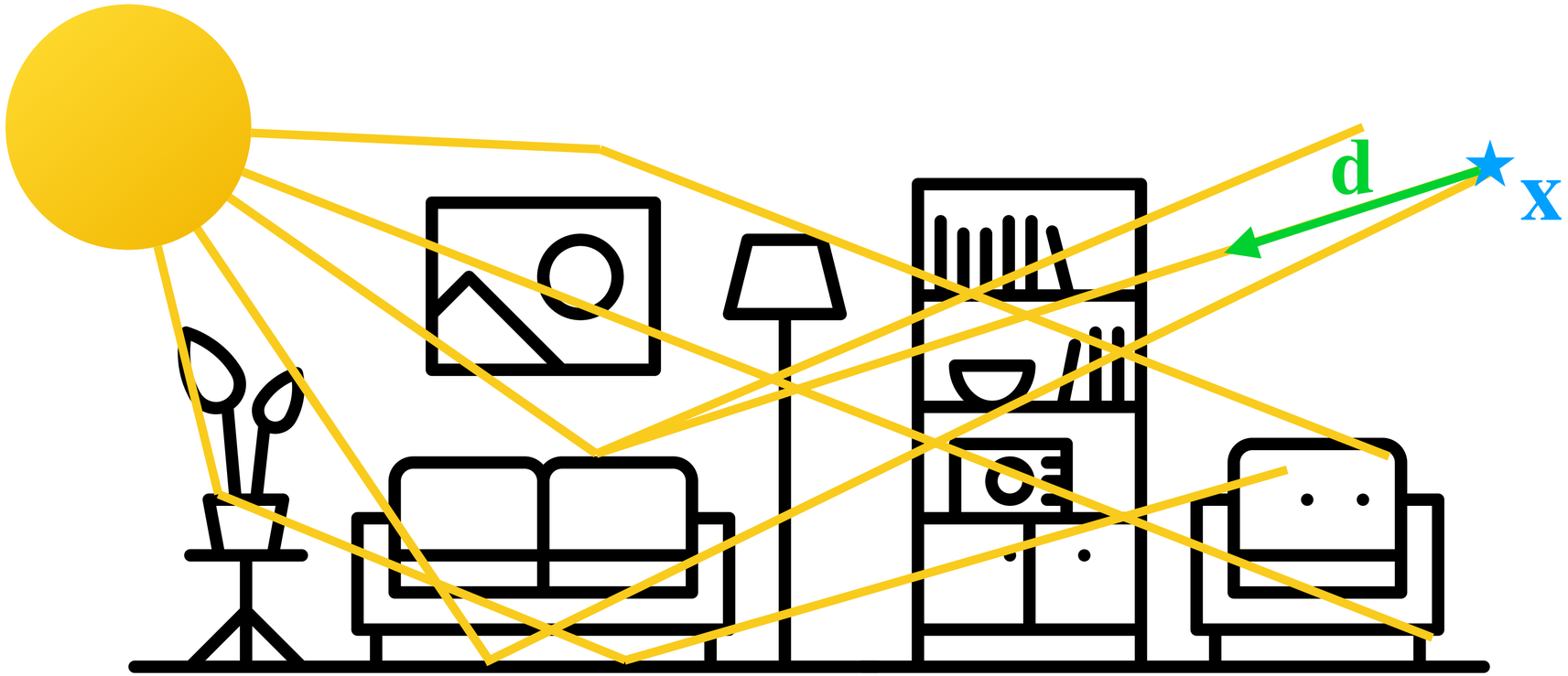
Semon Rezchikov[2,*]
skr@math.columbia.edu

William T. Freeman[1,3]
billf@mit.edu

Joshua B. Tenenbaum[1,4,5]
jbt@mit.edu

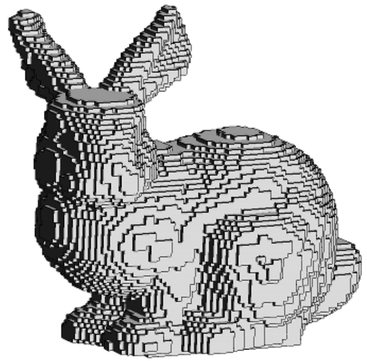
Frédo Durand[1]
fredo@mit.edu

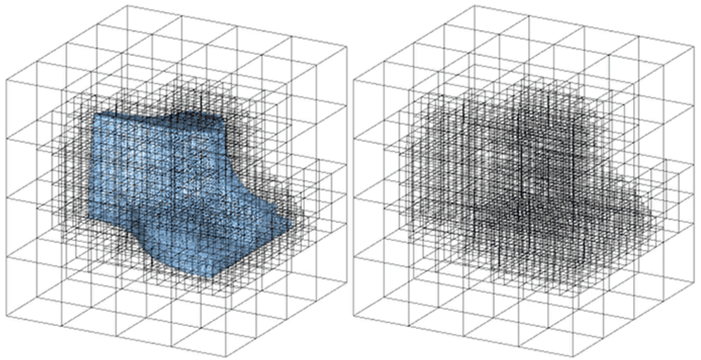
Just because Scene is 3D, don't have to store information in 3D data structure…

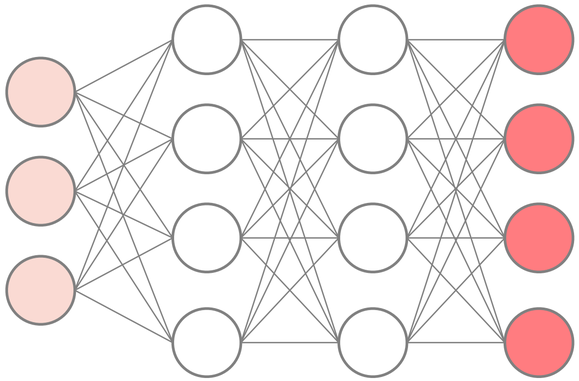


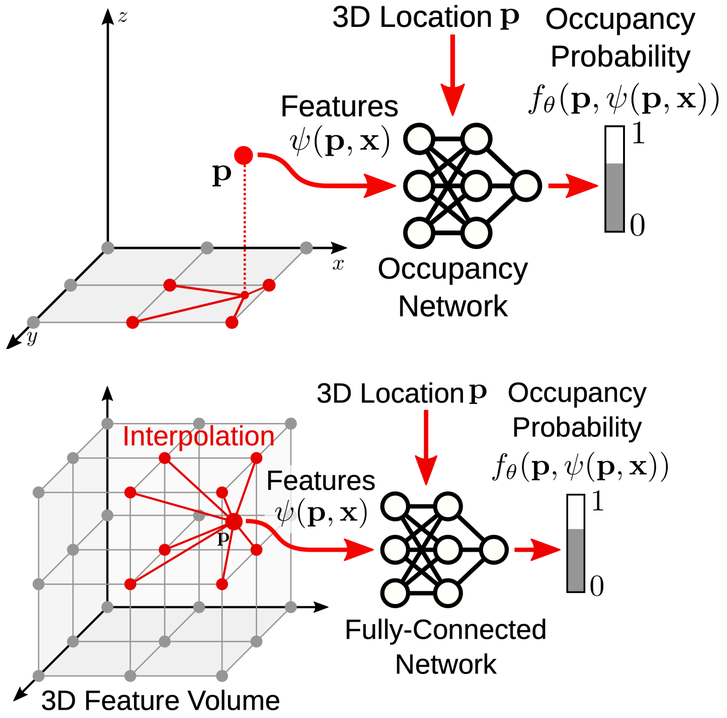
$$LF : \mathbb{R}^3 \times \mathbb{S}^2 \rightarrow \mathbb{R}^3, \quad LF(\mathbf{x}, \mathbf{d}) = \mathbf{c}$$

Summary









- No “Artificial Intelligence”, “understanding” or “Machine Learning” when fitting a single representation to a single 3D scene, even if there are neural networks in them.
- It becomes more interesting when considering *inference,* i.e., we want representation to be output of an encoder! More on that later in “Prior-based reconstruction”.

Inverse rendering

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

Camera calibration + Bundle Adjustment (Multi-View Geometry Lecture)!

Correspondences, 8 point, RANSAC and all that good stuff



Bundle adjustment minimizes the reprojection error of all observations:

$$\Pi^*, \mathcal{X}_w^* = \underset{\Pi, \mathcal{X}_w}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{p=1}^{P} w_{ip} \left\| \mathbf{x}_{ip}^s - \pi_i(\mathbf{x}_p^w) \right\|_2^2$$

Images **3D Scene** **Camera Poses**

Or maybe new neural versions

Can assume we know the camera poses.



Images



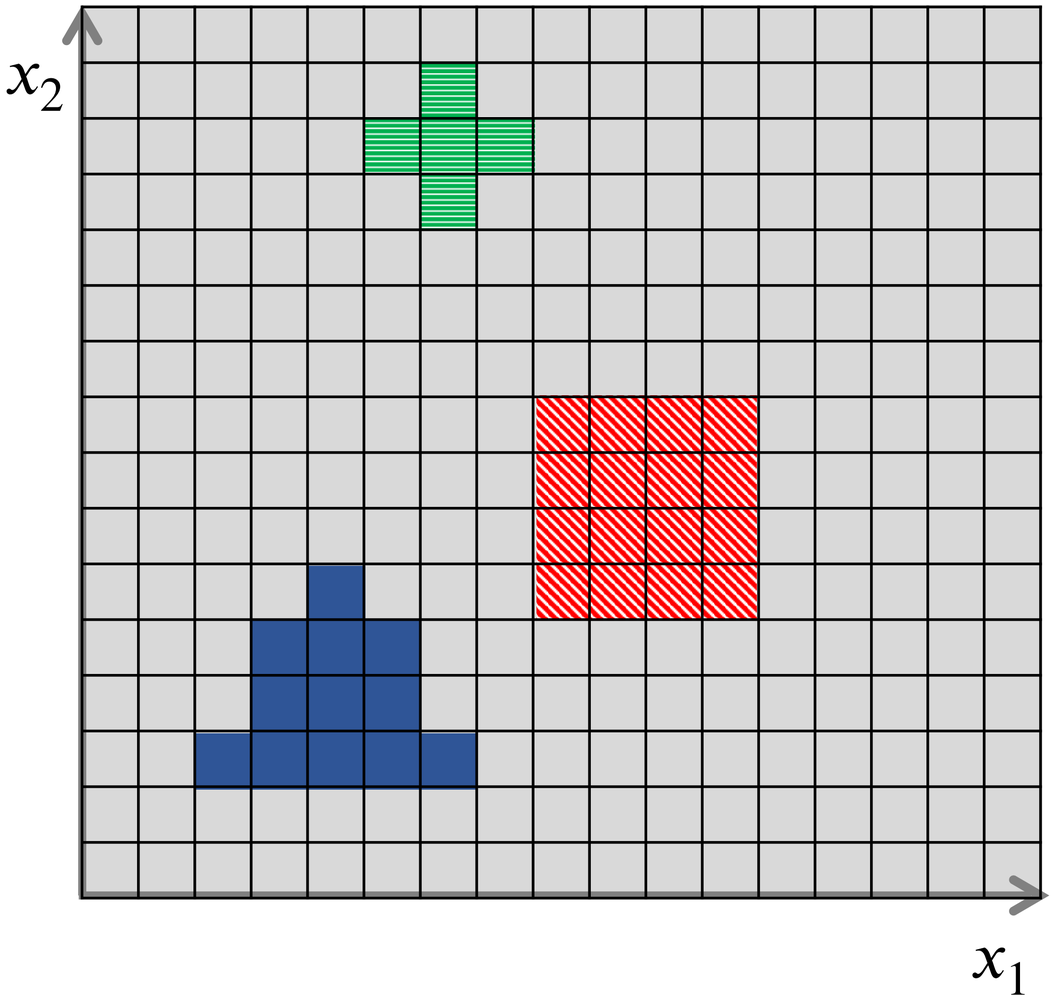
Camera Poses

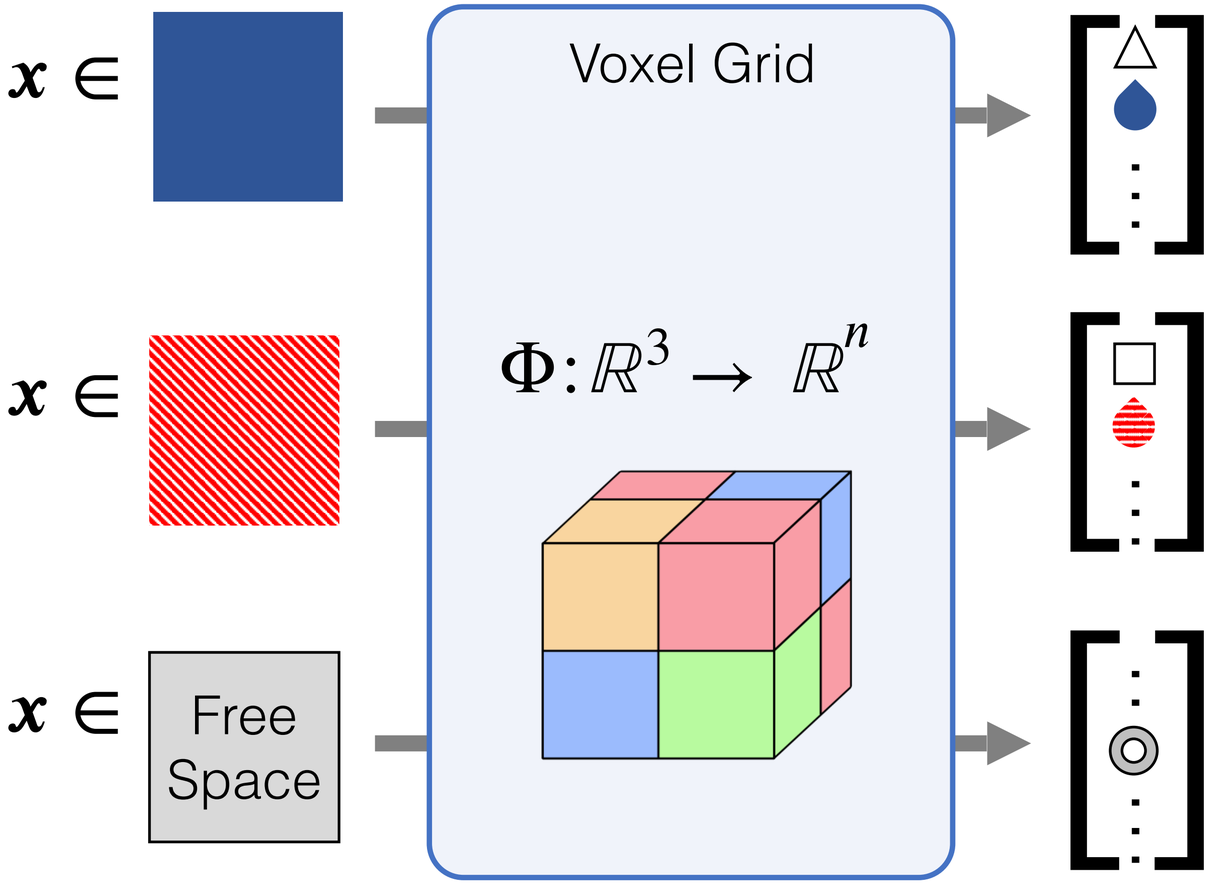
Reconstruct



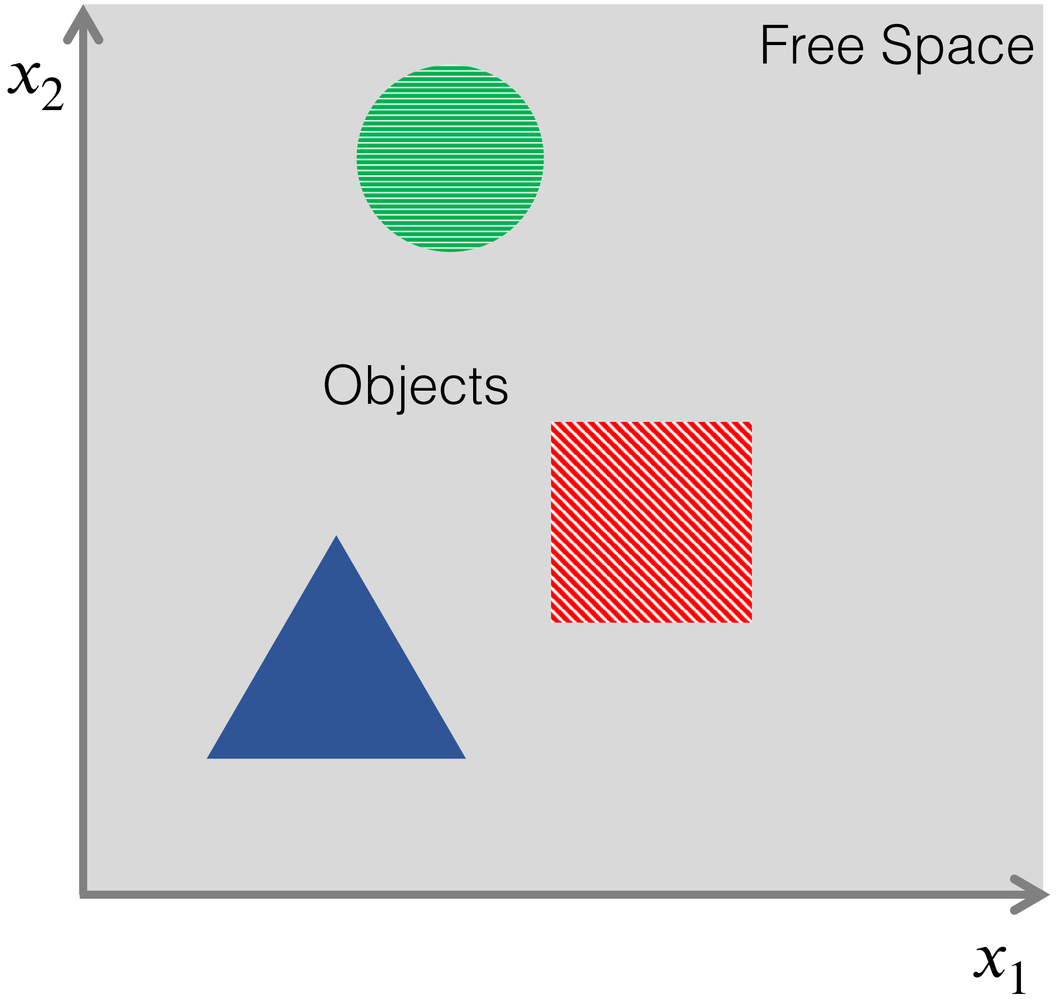
3D Scene

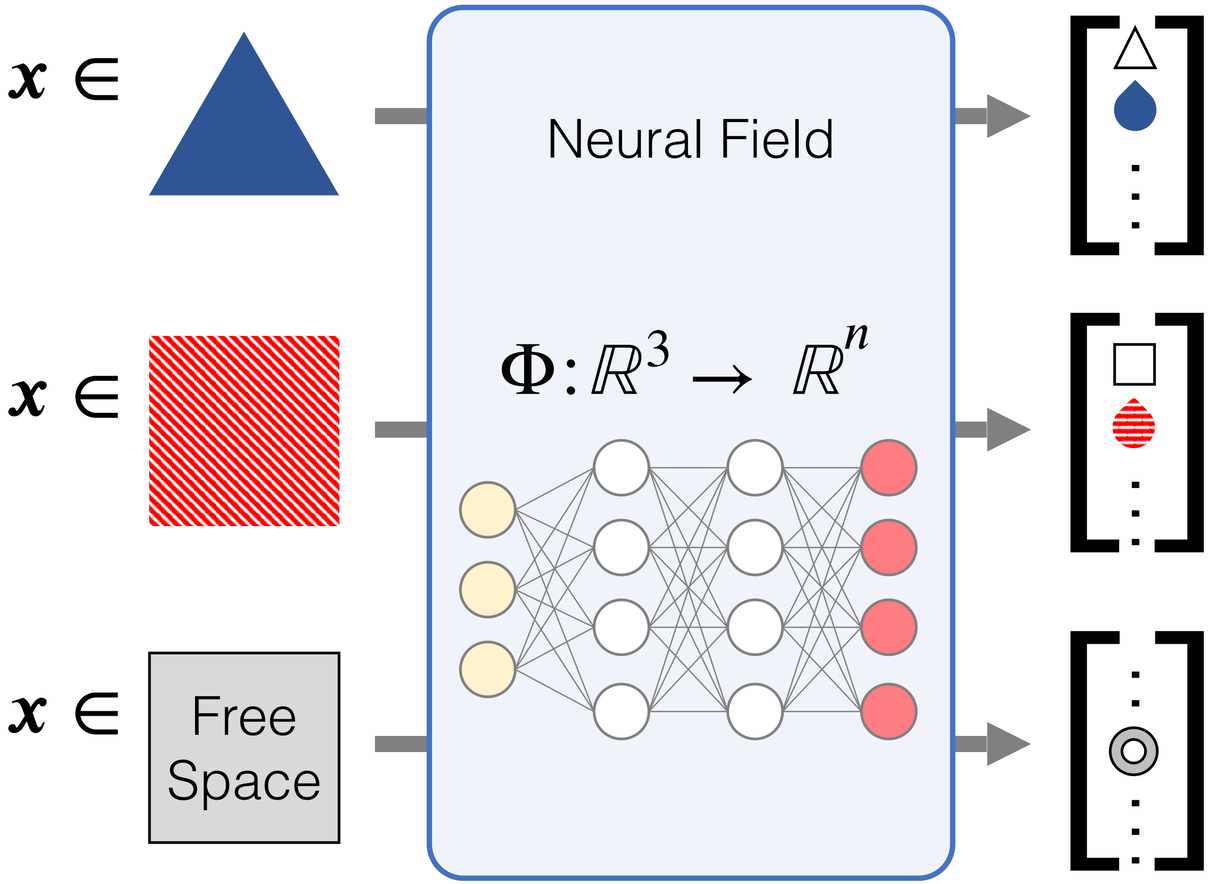
Unknown: 3D Scene Representation. Could be voxels





Unknown: 3D Scene Representation. Could be Neural field





Differentiable Rendering

Calibrated cameras

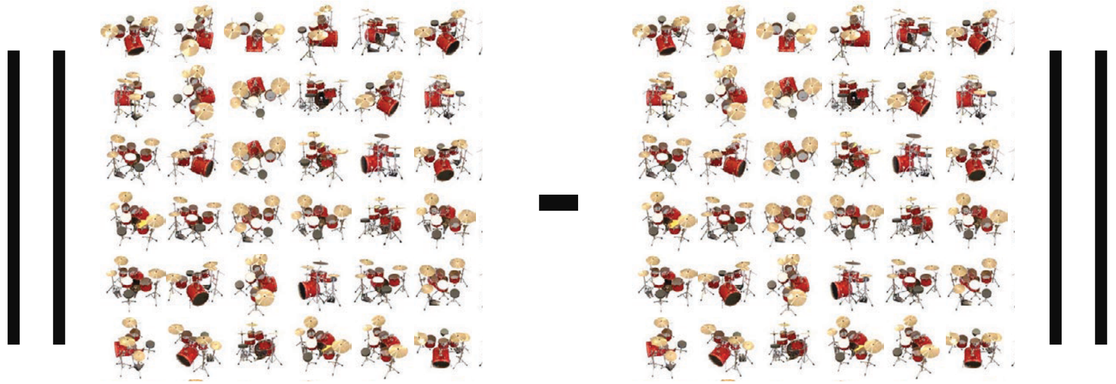


Scene Representation

Unknown

Renderer

Differentiable forward rendering



Rendered Images

Ground Truth (input) Images

Differentiable Rendering

Calibrated cameras

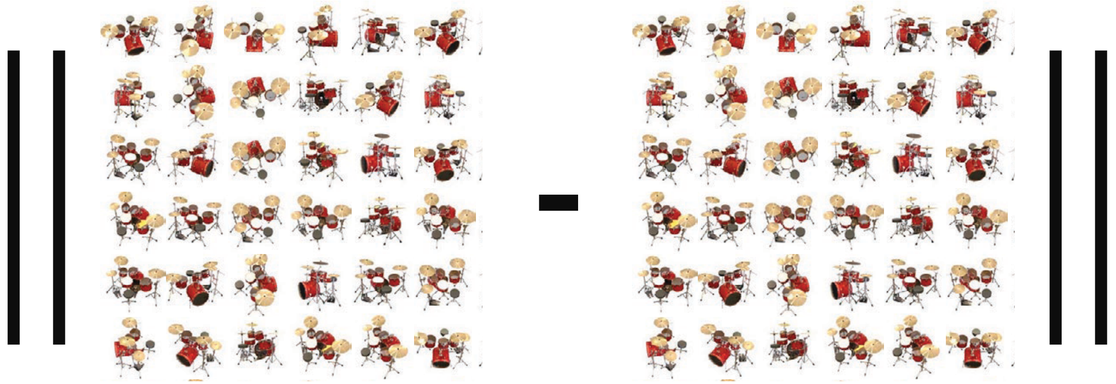


Scene Representation

Renderer

Unknown

Differentiable forward rendering



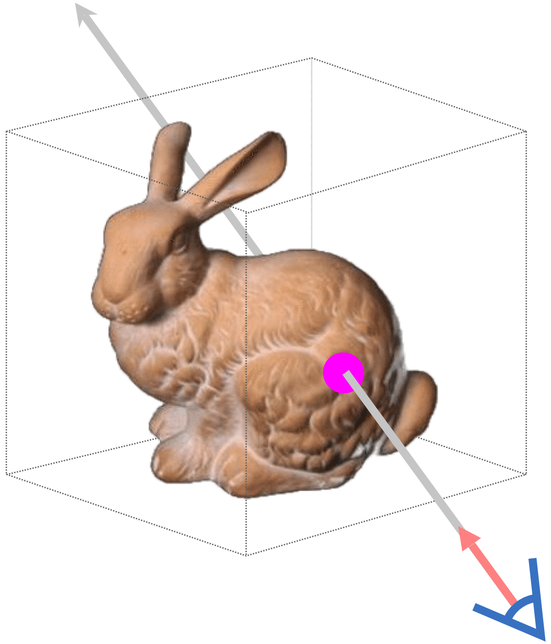
Rendered Images

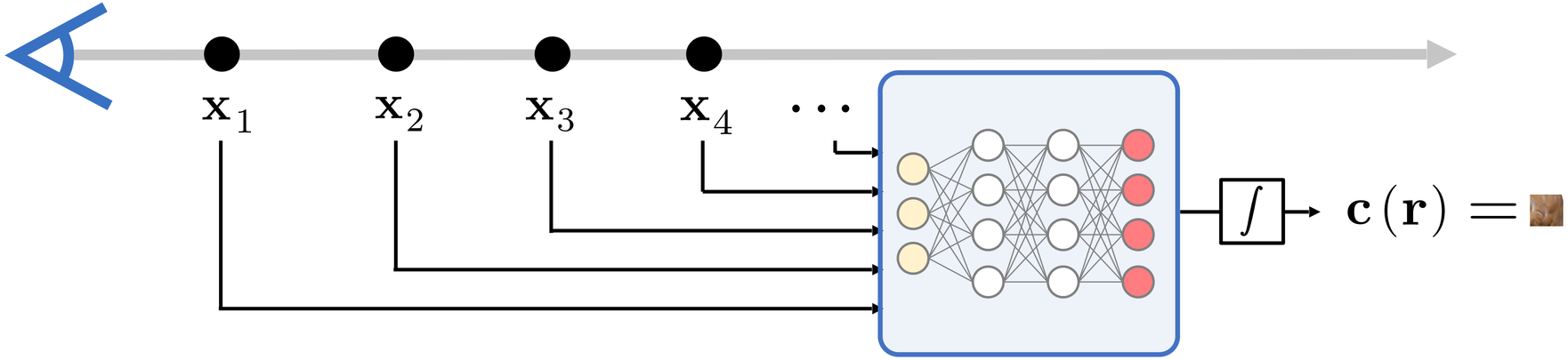
Ground Truth (input) Images

Optimization via Backprop & SGD

Volumetric rendering

General structure of Neural Renderers for 3D Fields





Sphere-Tracing
[JC Hart, 1996]

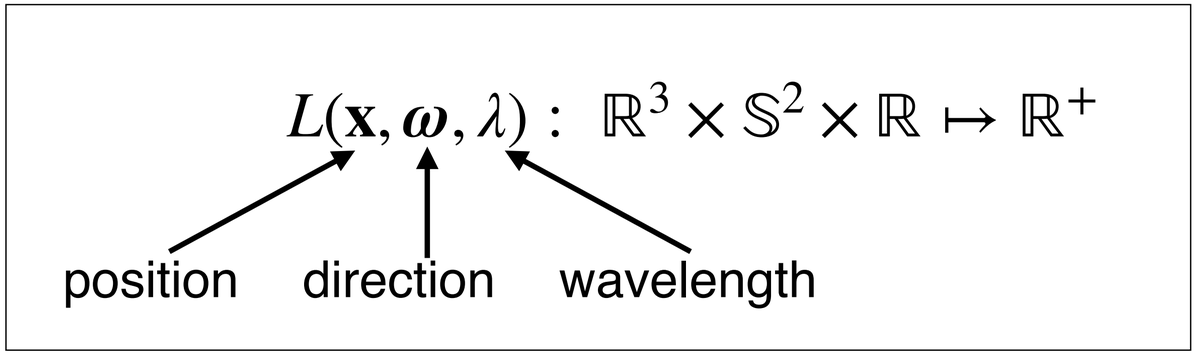
Volumetric

Hybrid implicit-volumetric

Learned aggregation

Radiance

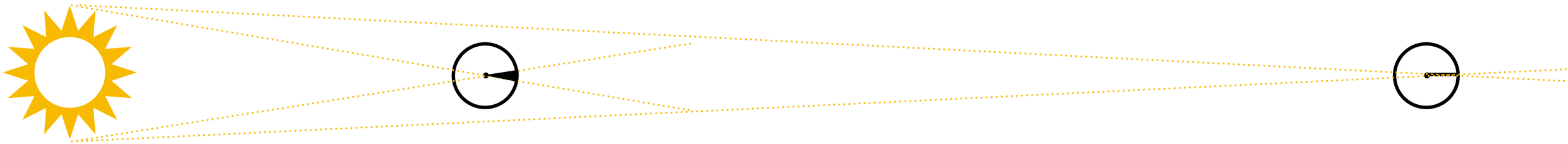
- Radiance is a fundamental quantity of light.
- Radiance is density of photons at a point, traveling in the same direction, at given wavelength.
- Radiance is energy per time, per unit area, per unit solid angle, per wavelength.



The “Light Field”: Radiance for every possible ray

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

Radiance



- Radiance properties
  - Radiance along an unblocked ray is constant.
  - Response of sensors is proportional to radiance of visible surface.
  - Radiance of reflected light is proportional to that of incoming light:

$$L(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) \propto L(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)$$

reflected incoming

Slide Credit: Stanford CS348I, Computer Graphics in the Age of AI, Stanford, C. Karen Liu & Jiajun Wu

The Rendering equation

Recursive formulation of radiance for global illumination

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_\Omega f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda)\, L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda)\, (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

But most neural rendering ignores all this and only cares about outgoing radiance with illumination “baked in”

Volume Rendering



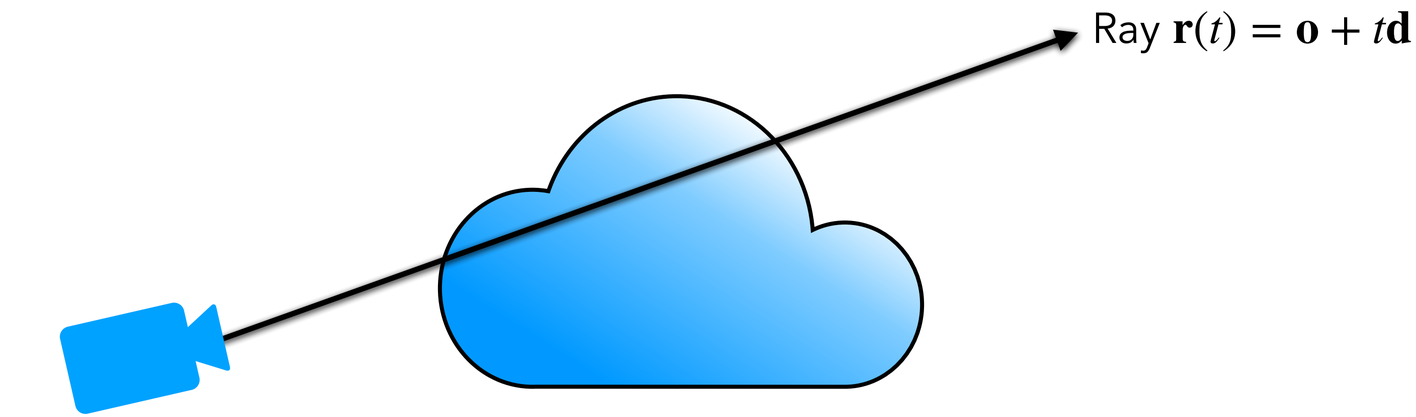
Scene is a cloud of tiny colored particles

Max and Chen 2010, *Local and Global Illumination in the Volume Rendering Integral*

Neural Radiance Fields, Mildenhall et al. 2021

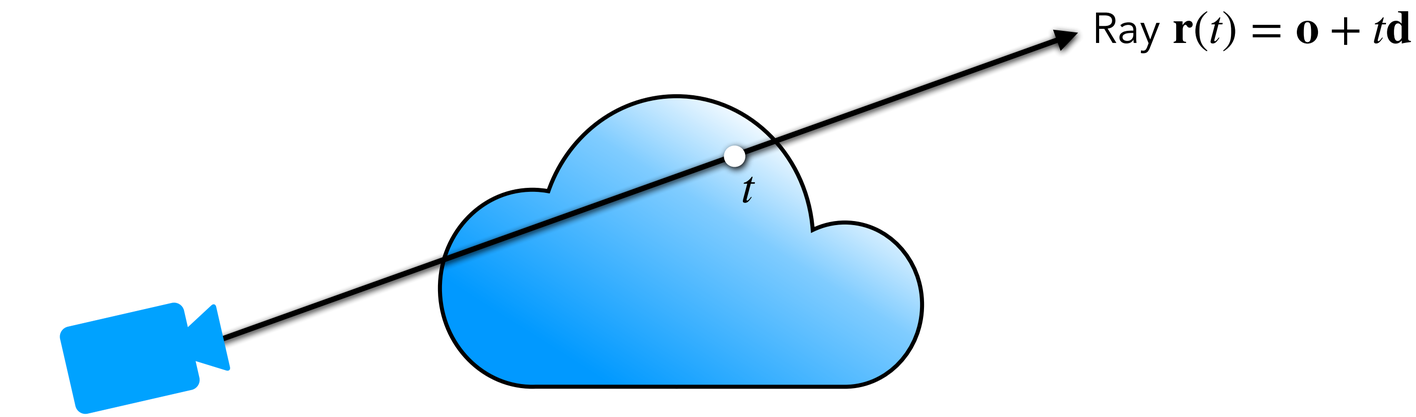
Slide credit: Ben Mildenhall

Volume Rendering



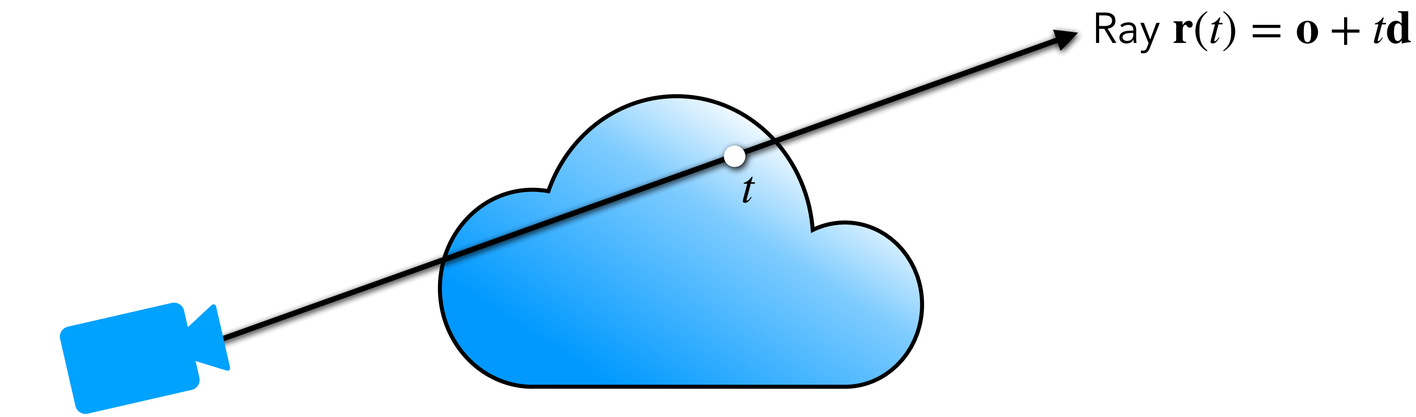
If a ray traveling through the scene hits a particle at t, we return its color $\mathbf{c}(t)$

Volume Rendering



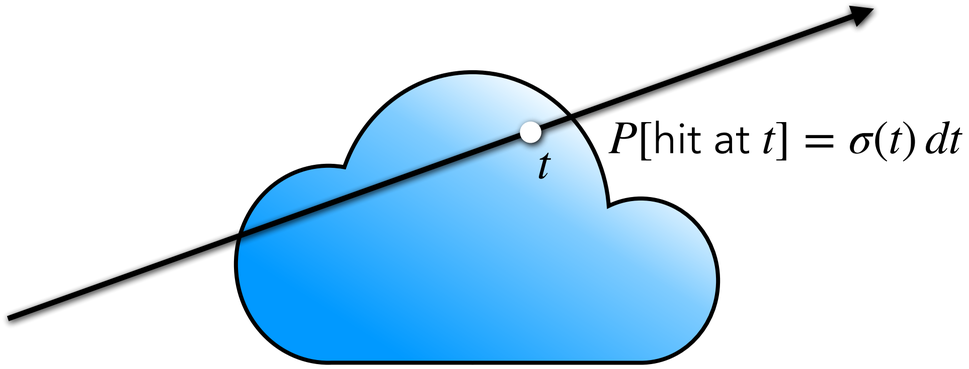
If a ray traveling through the scene hits a particle at t, we return its color $\mathbf{c}(t)$

Volume Rendering



If a ray traveling through the scene hits a particle at t, we return its color $\mathbf{c}(t)$

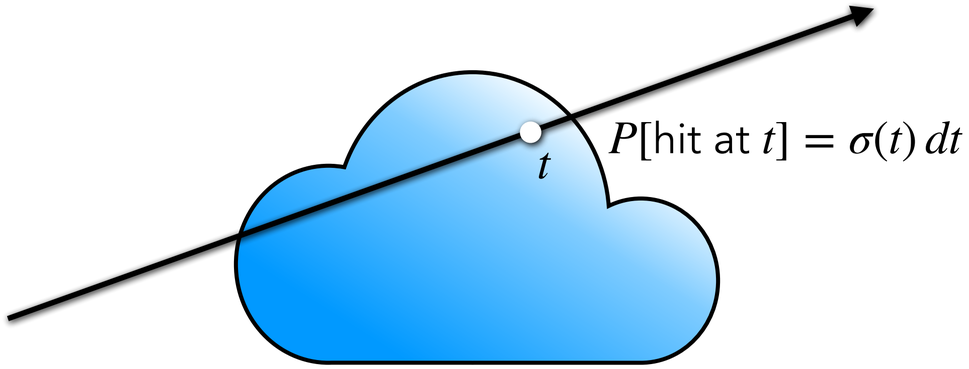
Volume Rendering



Probability that ray stops in a small interval around t is $\sigma(t)\,dt$.

σ is known as the **Volume Density.**

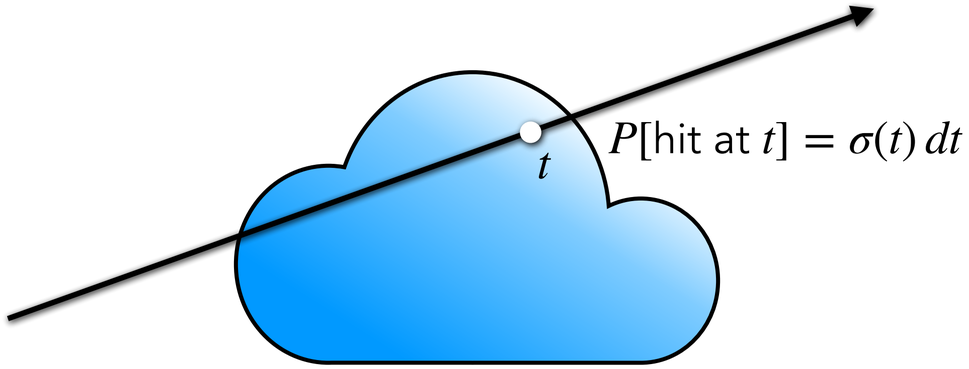
Volume Rendering



Probability that ray stops in a small interval around t is $\sigma(t)\,dt$.

σ is known as the **Volume Density.**

Volume Rendering



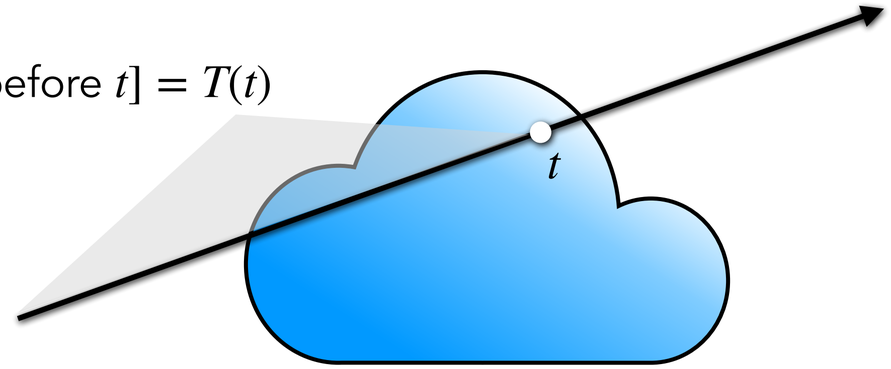
Probability that ray stops in a small interval around t is $\sigma(t)\,dt$.
σ is known as the **Volume Density.**

Our field thus has the function signature:

$$\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \times \mathbb{R}^+; \quad \Phi(\mathbf{x}) = (\sigma, \mathbf{c})$$

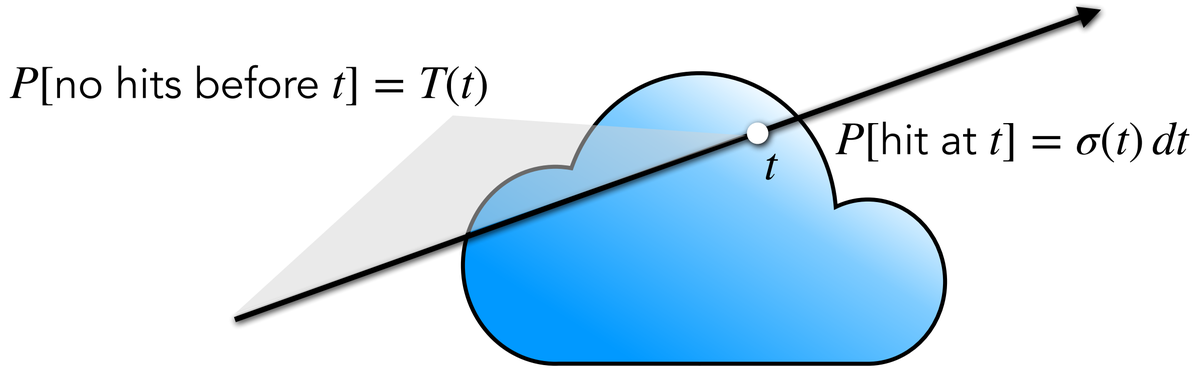
Volume Rendering

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



To determine if t is the *first* hit, need to know $T(t)$:
probability that the ray didn't hit any particles earlier.
$T(t)$ is called **Transmittance.**

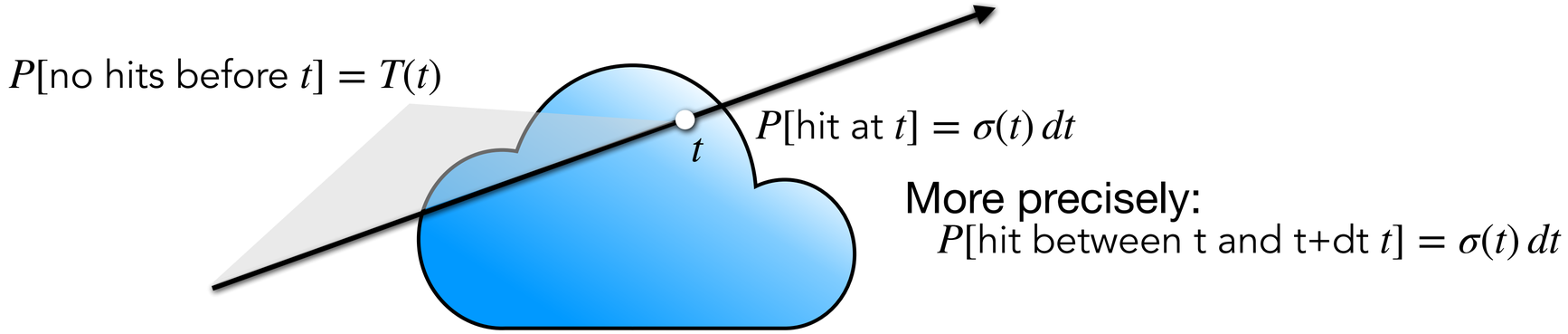
Volume Rendering



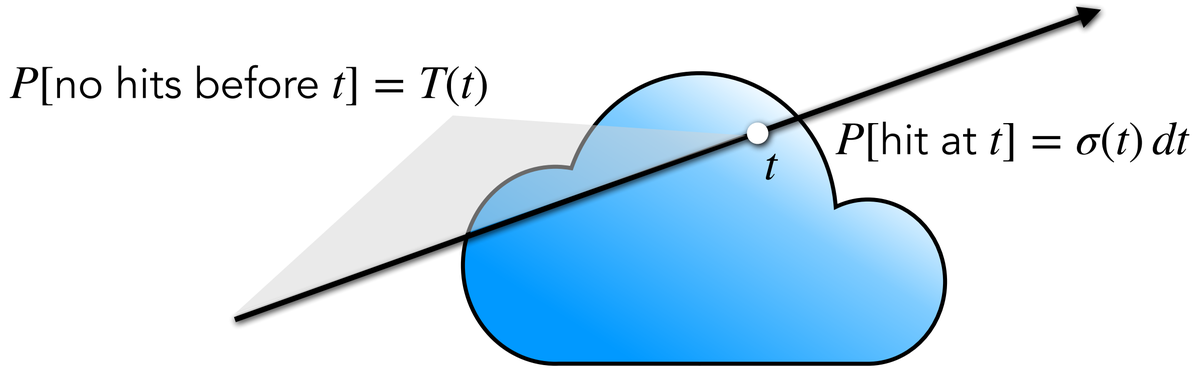
σ and T are related via

$$P[\text{no hit before } t + dt] = P[\text{no hit before } t] \times P[\text{no hit at } t]$$

Volume Rendering



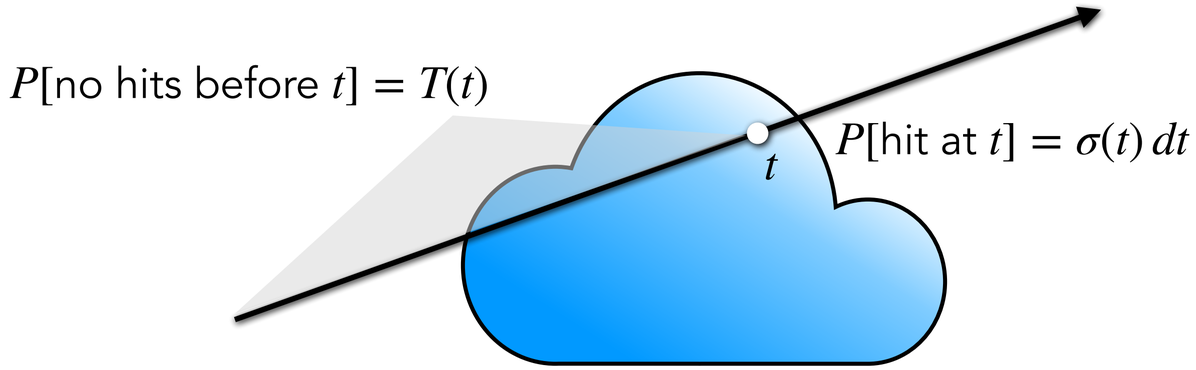
Volume Rendering



σ and T are related via

$$P[\text{no hit before } t + dt] = P[\text{no hit before } t] \times P[\text{no hit at } t]$$

Volume Rendering



σ and T are related via

$$T(t + dt) = T(t) \quad \times \quad [\quad 1 - \sigma(t)dt \quad]$$

Volume Rendering

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

Volume Rendering

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

$$\frac{T(t+dt)-T(t)}{dt} = -T(t)\sigma(t)$$

$$T'(t) = -T(t)\sigma(t)$$

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

$$\int_a^b \frac{T'(t)}{T(t)}dt = -\int_a^b \sigma(t)dt$$

$$\log T(t)\,|_a^b = -\int_a^b \sigma(t)dt$$

Neural Radiance Fields, Mildenhall et al. 2021

Volume Rendering

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

$$\frac{T(t+dt)-T(t)}{dt} = -T(t)\sigma(t)$$

$$T'(t) = -T(t)\sigma(t)$$

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

$$\int_a^b \frac{T'(t)}{T(t)}dt = -\int_a^b \sigma(t)dt$$

$$\log T(t)\,|_a^b = -\int_a^b \sigma(t)dt$$

Solving simple differential equation (for great explanation, see "Volume Rendering Digest", Tagliasacchi et al.)

Volume Rendering

No hits before t is equal to integral over density up until t.

$$T(t) = \exp\left(-\int_0^t \sigma(a)da\right)$$

Neural Radiance Fields, Mildenhall et al. 2021

Volume Rendering

No hits before t is equal to integral over density up until t.

$$T(t) = \exp\left(-\int_0^t \sigma(a)da\right)$$

$1 - T(t)$ can be seen as cumulative distribution function of probability that ray hits something before reaching t.

Then $T'(t) = T(t)\sigma(t)$ is probability that ray stops *exactly* at t .

Neural Radiance Fields, Mildenhall et al. 2021

Questions?

$$T(t) = \exp\left(-\int_0^t \sigma(a)da\right)$$

Volume Rendering

Then $T'(t) = T(t)\sigma(t)$ is probability that ray stops *exactly* at t
and returns the color c(t) at that location

Volume Rendering

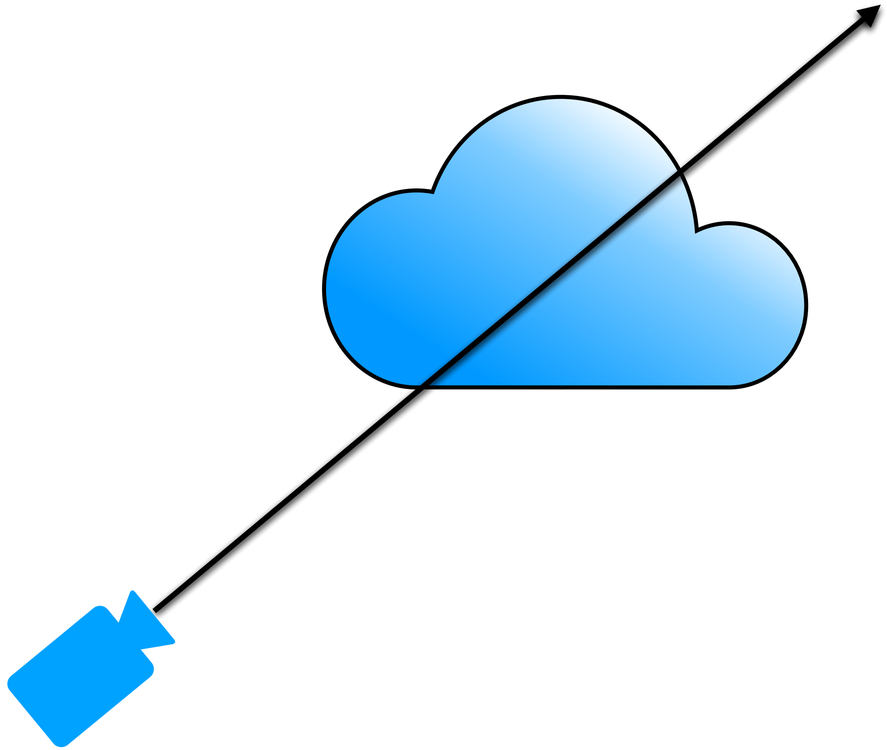
Then $T'(t) = T(t)\sigma(t)$ is probability that ray stops *exactly* at t
and returns the color c(t) at that location

So the expected color returned by the ray will be

$$\int_{t_0}^{t_1} T(t)\sigma(t)\mathbf{c}(t)\, dt$$

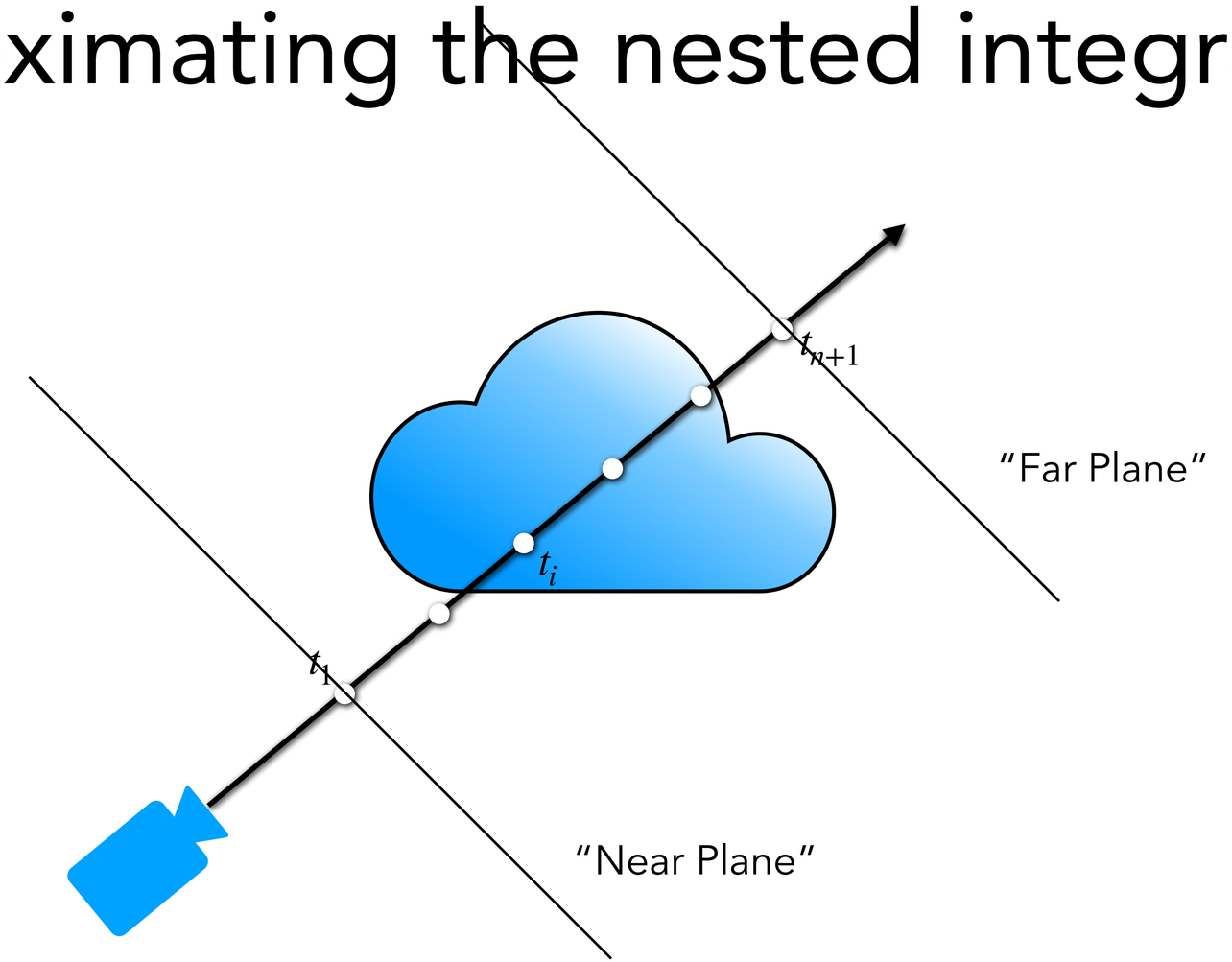
Note the nested integral! (Since T involves an integral)

Approximating the nested integral



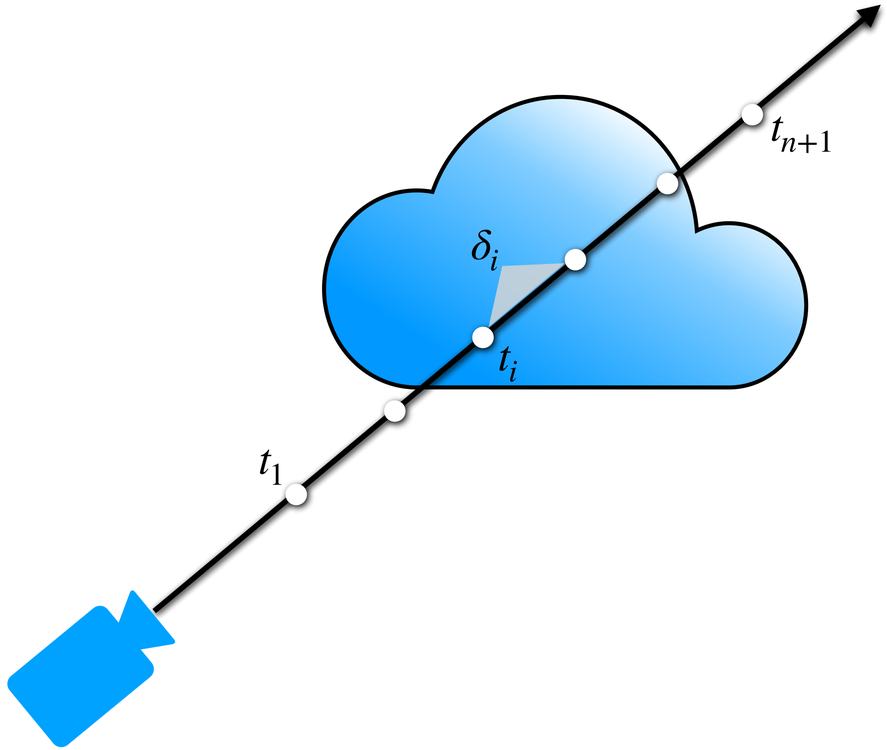
We use quadrature to approximate the nested integral,

Approximating the nested integral



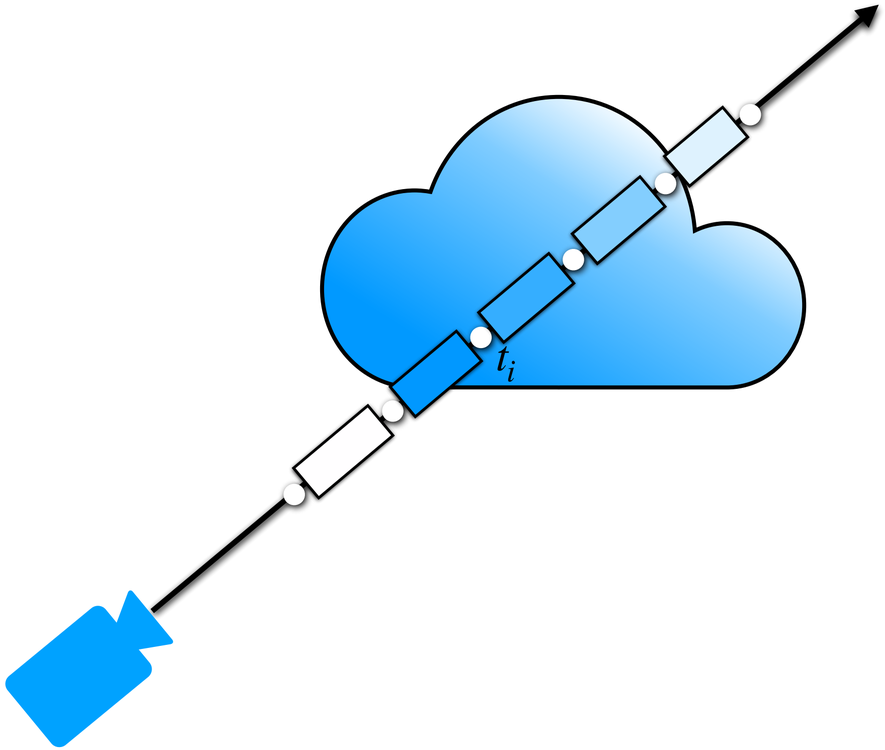
We use quadrature to approximate the nested integral,
splitting the ray up into n segments with endpoints $\{t_1, t_2, \ldots, t_{n+1}\}$

Approximating the nested integral



We use quadrature to approximate the nested integral,
splitting the ray up into n segments with endpoints $\{t_1, t_2, \ldots, t_{n+1}\}$
with lengths $\delta_i = t_{i+1} - t_i$

Approximating the nested integral



We assume volume density and color are roughly constant within each interval

Approximating the nested integral

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

This allows us to break the outer integral

Approximating the nested integral

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

This allows us to break the outer integral into a sum

Approximating the nested integral

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

Caveat: piecewise constant density and color
do not imply constant transmittance!

Linear approximations of exponentials are not great!

$$T(t) = \exp\left(-\int_0^t \sigma(a)da\right)$$

Important to account for how early part of a segment blocks later part when σ_i is high

Approximating the nested integral

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

Approximating the nested integral

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

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

Approximating the nested integral

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

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

Integrate

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

Approximating the nested integral

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

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

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

Cancel σ_i

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

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

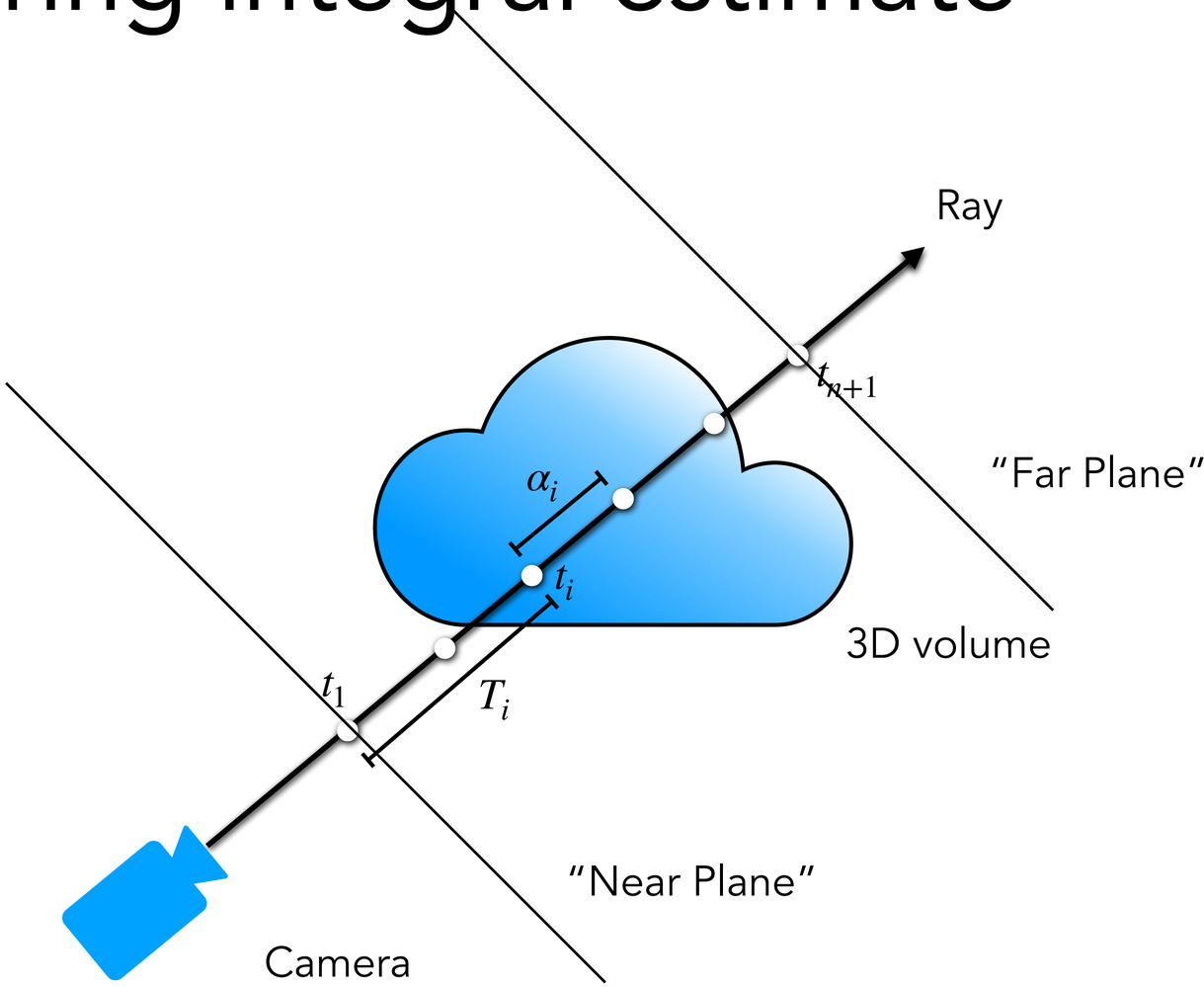
weights

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*:

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



NERF extra inductive bias

MLP that outputs sigma
only takes x, y, z

For color, add direction

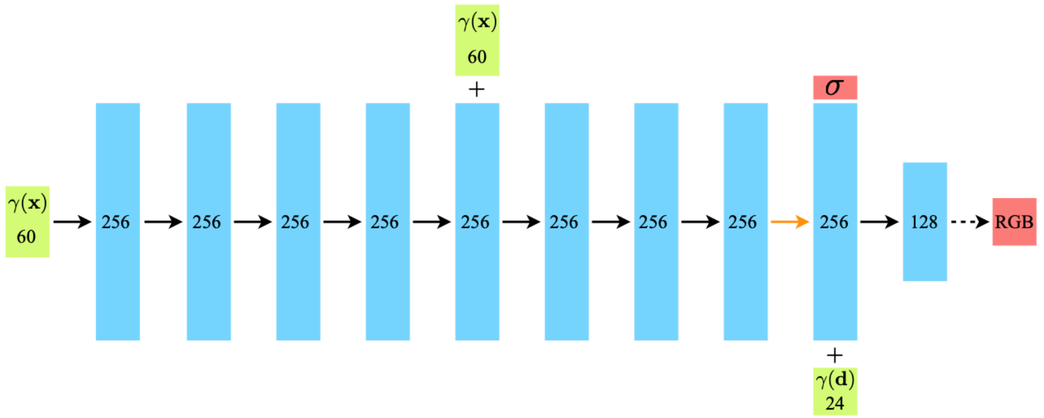
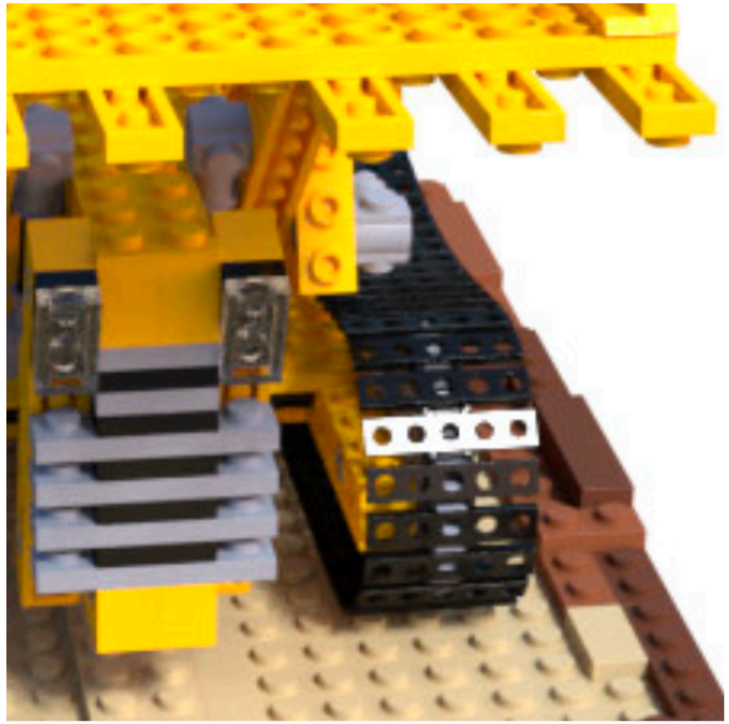


Fig. 7: A visualization of our fully-connected network architecture. Input vectors are shown in green, intermediate hidden layers are shown in blue, output vectors are shown in red, and the number inside each block signifies the vector's dimension. All layers are standard fully-connected layers, black arrows indicate layers with ReLU activations, orange arrows indicate layers with no activation, dashed black arrows indicate layers with sigmoid activation, and "+" denotes vector concatenation. The positional encoding of the input location ($\gamma(\mathbf{x})$) is passed through 8 fully-connected ReLU layers, each with 256 channels. We follow the DeepSDF [32] architecture and include a skip connection that concatenates this input to the fifth layer's activation. An additional layer outputs the volume density σ (which is rectified using a ReLU to ensure that the output volume density is nonnegative) and a 256-dimensional feature vector. This feature vector is concatenated with the positional encoding of the input viewing direction ($\gamma(\mathbf{d})$), and is processed by an additional fully-connected ReLU layer with 128 channels. A final layer (with a sigmoid activation) outputs the emitted RGB radiance at position $\mathbf{x}$, as viewed by a ray with direction $\mathbf{d}$.

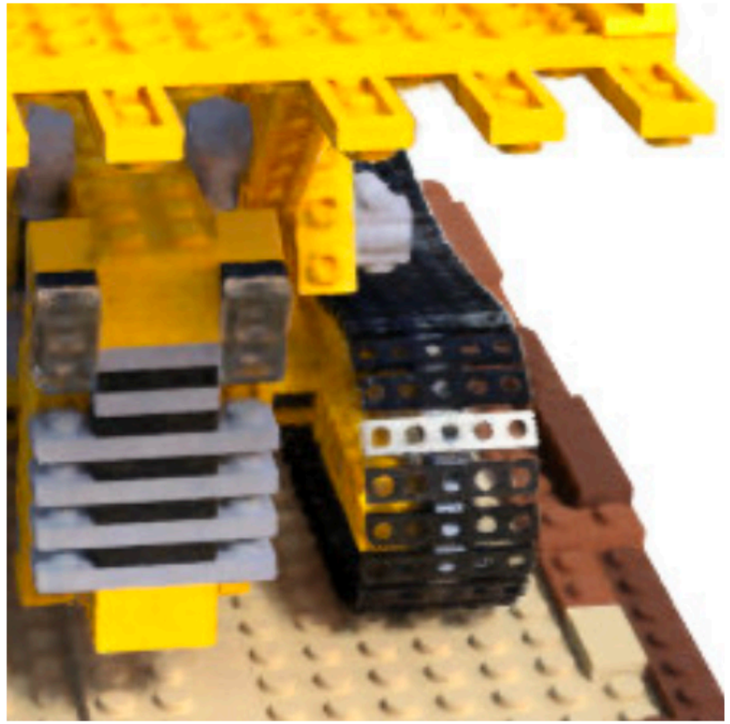
NERF extra inductive bias

MLP that outputs sigma
only takes x, y, z

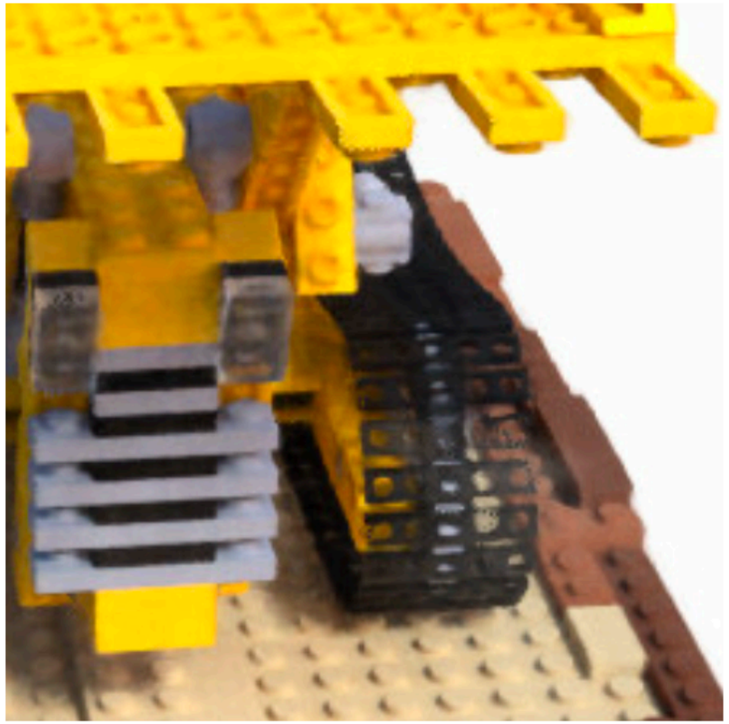
For color, add direction



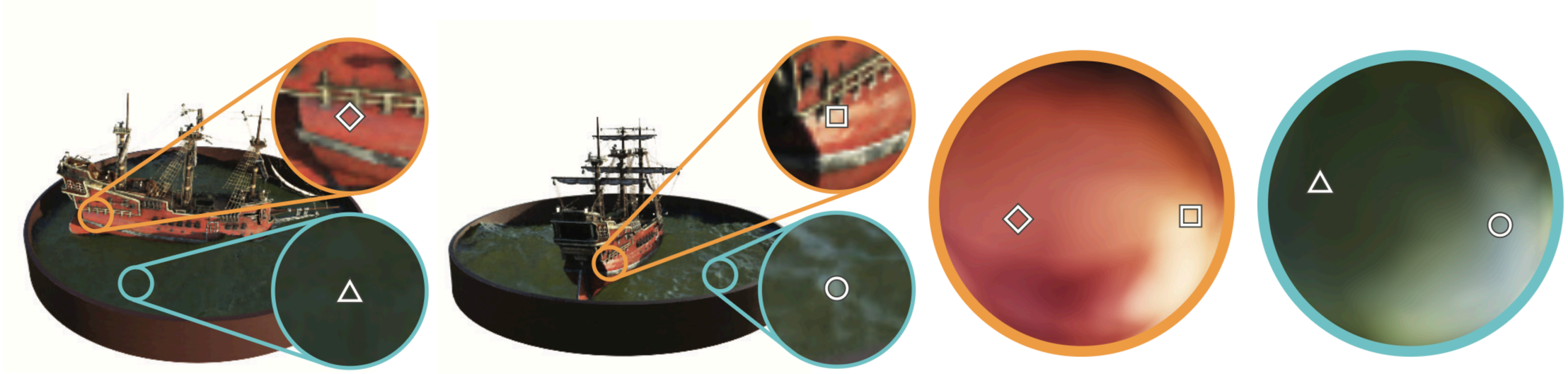
Ground Truth



Complete Model



No View Dependence



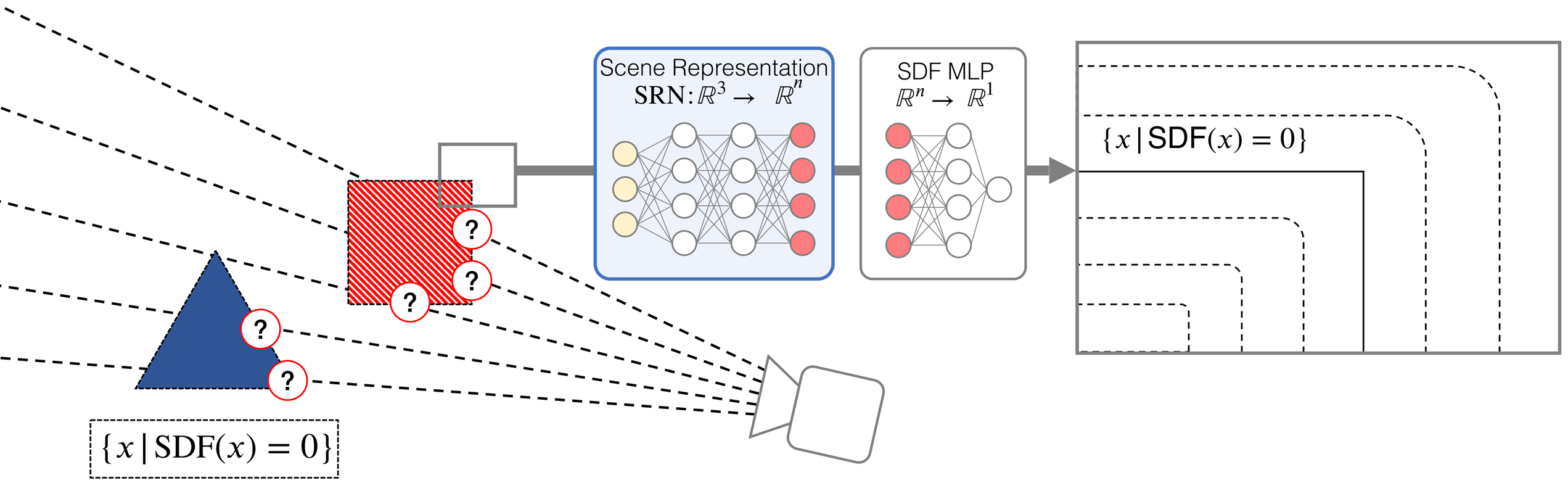
(a) View 1 (b) View 2 (c) Radiance Distributions

Fig. 3: A visualization of view-dependent emitted radiance. Our neural radiance field representation outputs RGB color as a 5D function of both spatial position $\mathbf{x}$ and viewing direction $\mathbf{d}$. Here, we visualize example directional color distributions for two spatial locations in our neural representation of the *Ship* scene. In (a) and (b), we show the appearance of two fixed 3D points from two different camera positions: one on the side of the ship (orange insets) and one on the surface of the water (blue insets). Our method predicts the changing specular appearance of these two 3D points, and in (c) we show how this behavior generalizes continuously across the whole hemisphere of viewing directions.

Ray marching/ sphere tracing for SDFs

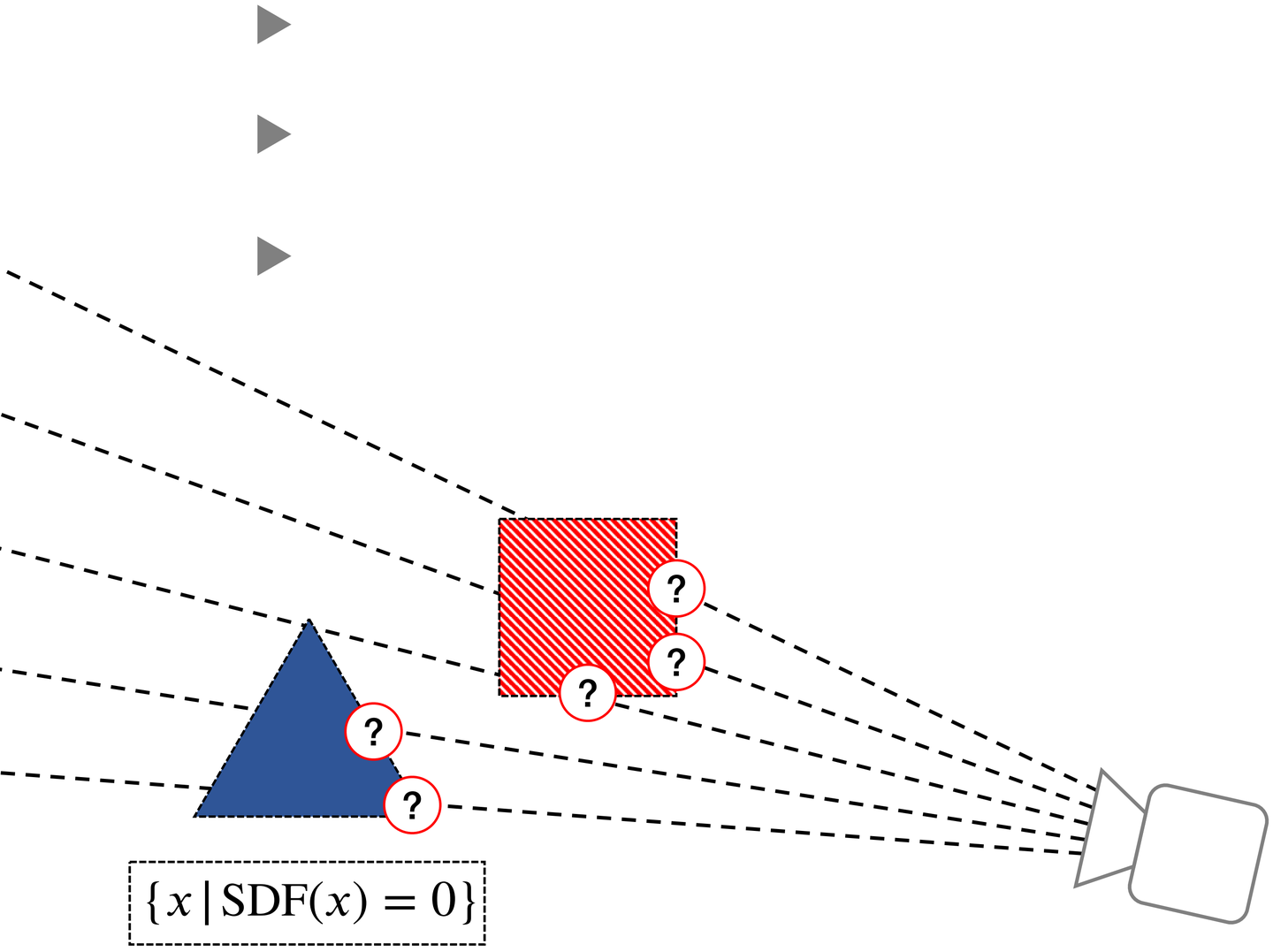
Sphere Tracing Step 1: Special case of SDF (signed distance field)

- Signed Distance Function maps point to distance to closest surface.



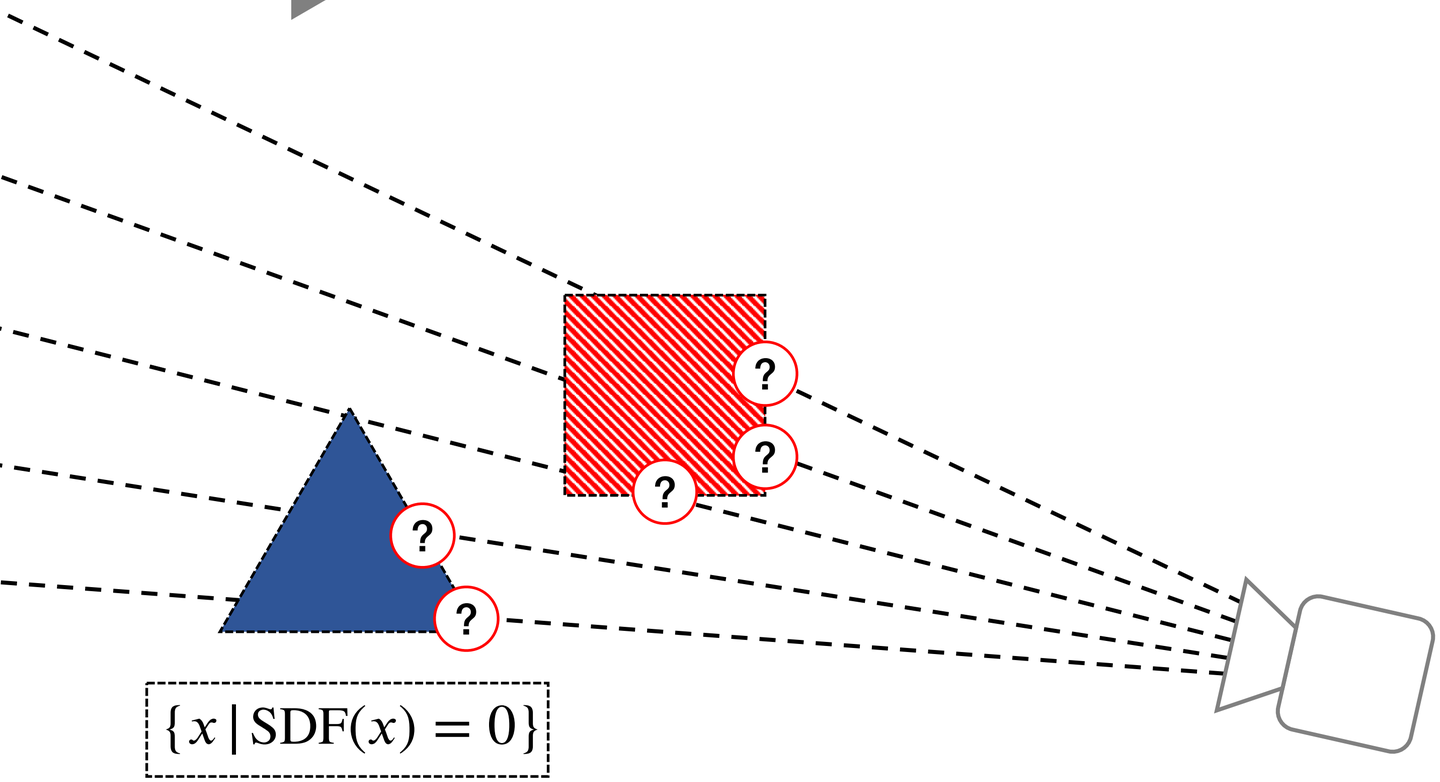
DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation, Mescheder et al.

Sphere Tracing Step 1: Intersection Testing.



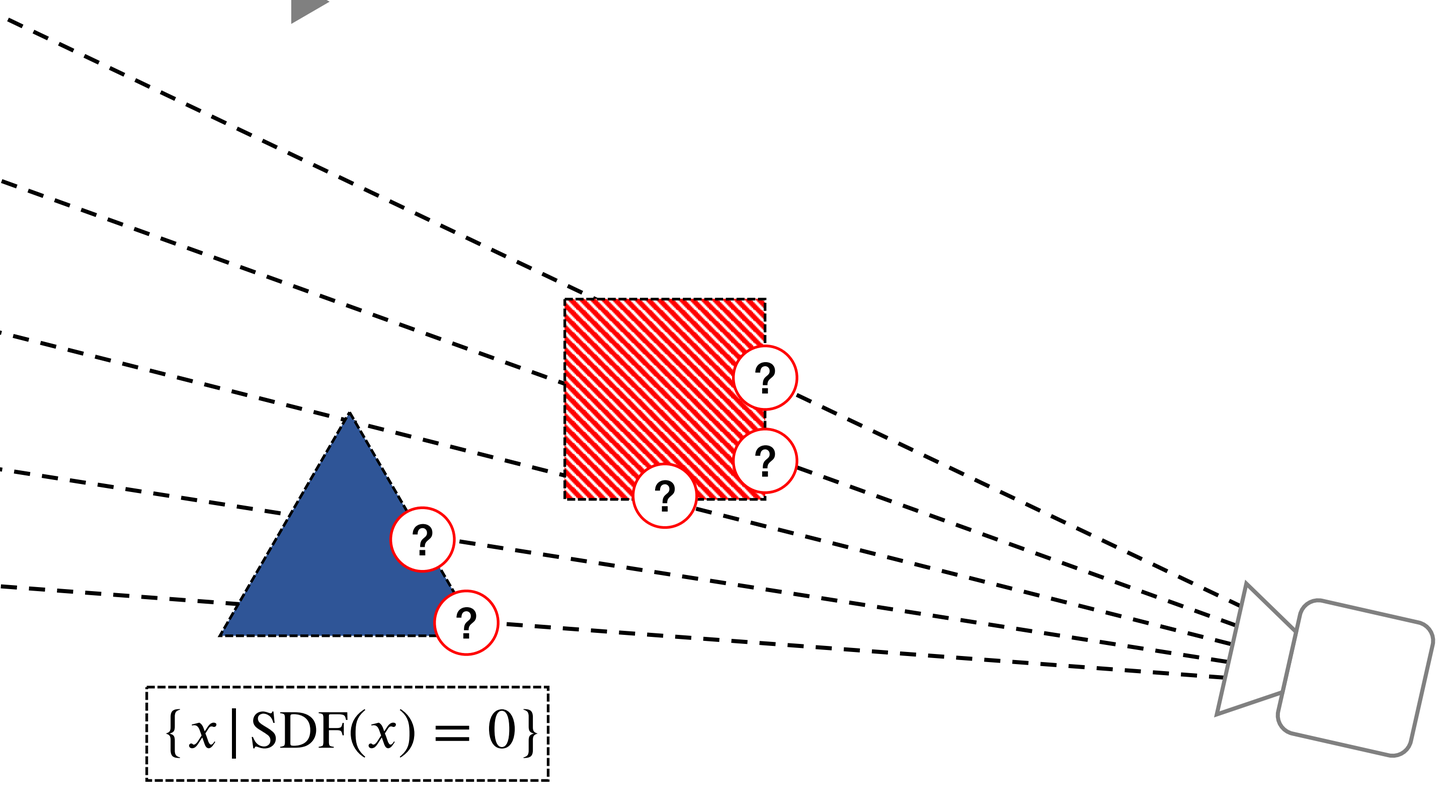
Sphere Tracing Step 1: Intersection Testing.

- How to find intersections?
-
-



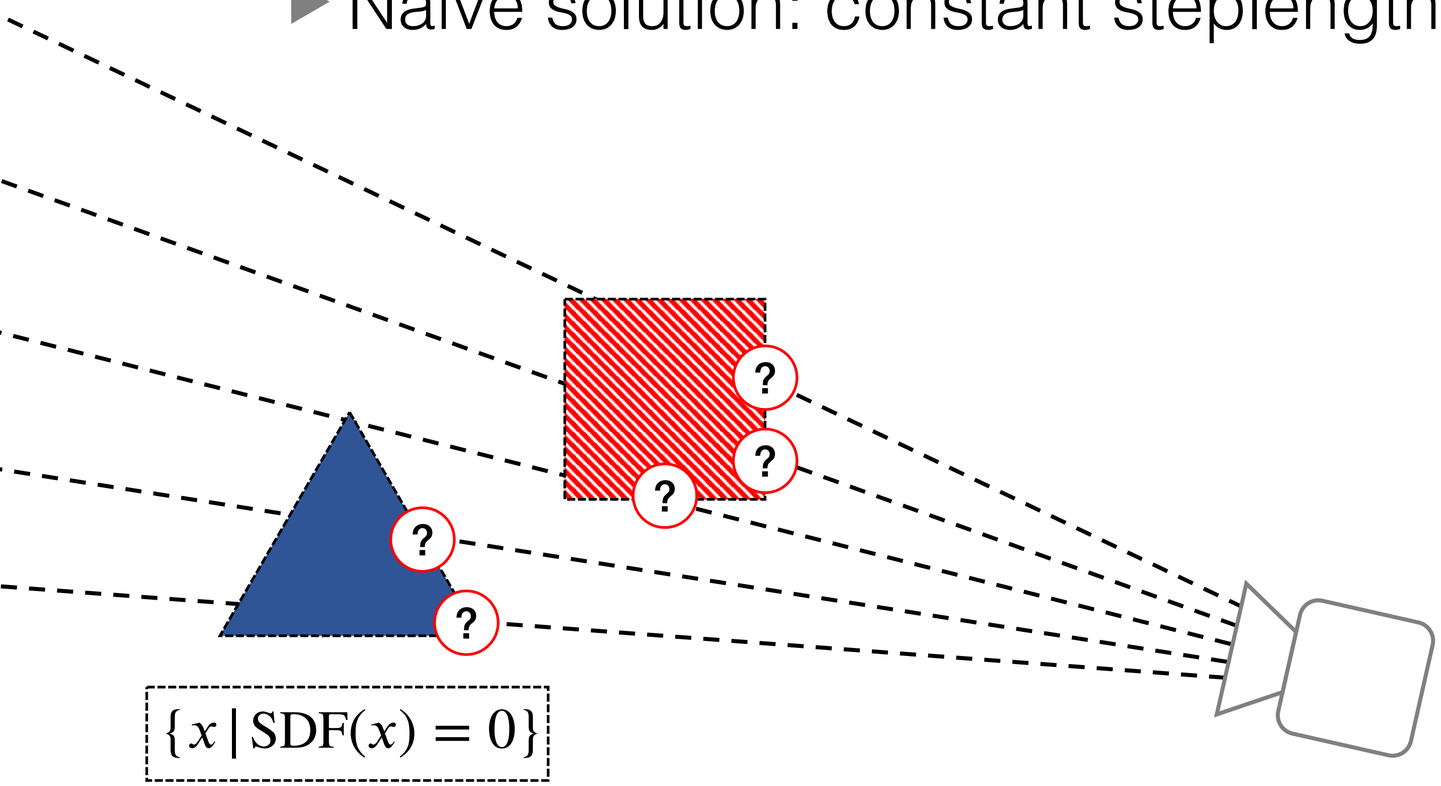
Sphere Tracing Step 1: Intersection Testing.

- How to find intersections?
- March along ray until we find zero-levelset.
-

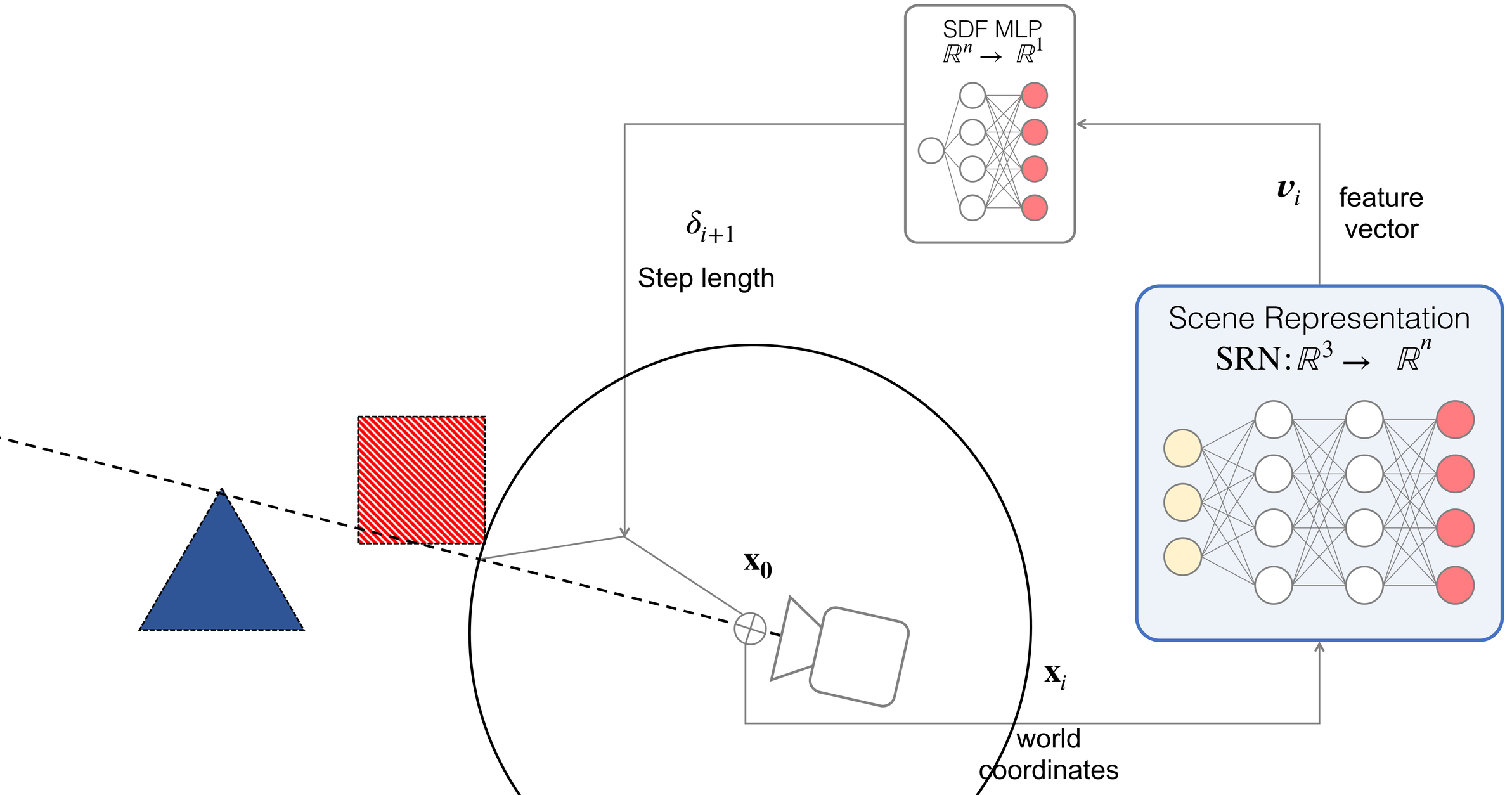


Sphere Tracing Step 1: Intersection Testing.

- How to find intersections?
- March along ray until we find zero-levelset.
- Naïve solution: constant steplength. Too expensive!

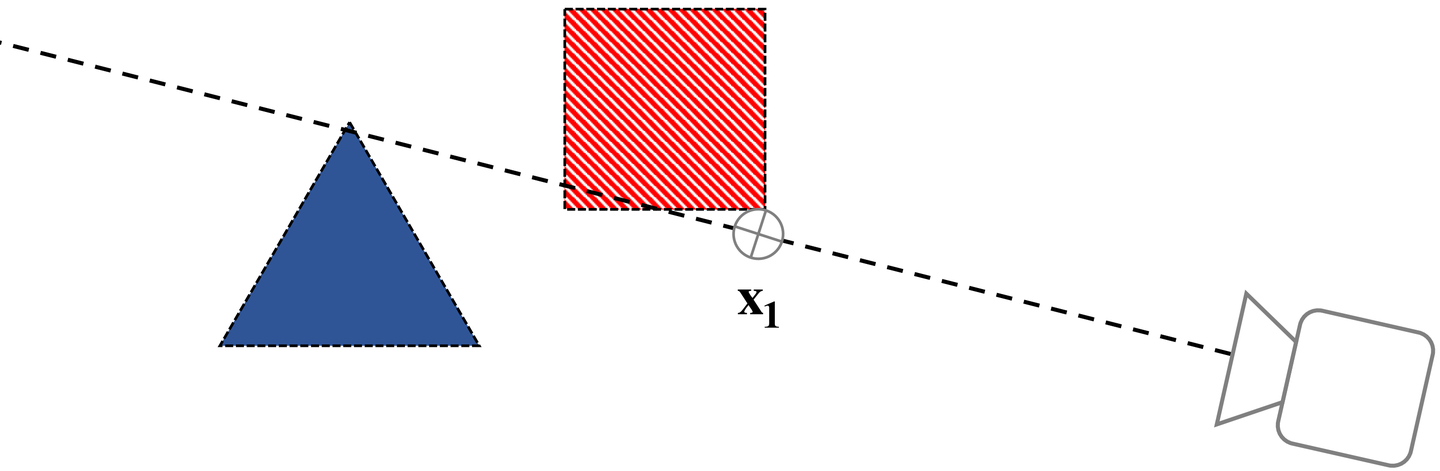


Sphere Tracing Step 1: Intersection Testing.



Sphere Tracing Step 1: Intersection Testing.

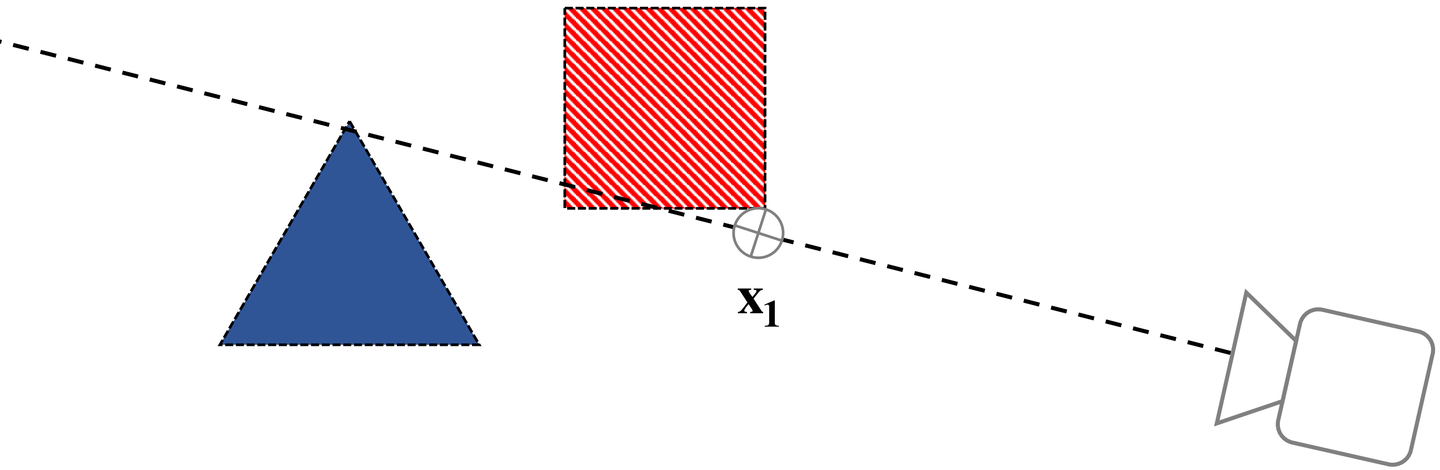
Well, if you have a true distance field (which is often not guaranteed, but that's usually swept under the rug)



Sphere Tracing Step 1: Intersection Testing.

- Guaranteed to converge to surface.

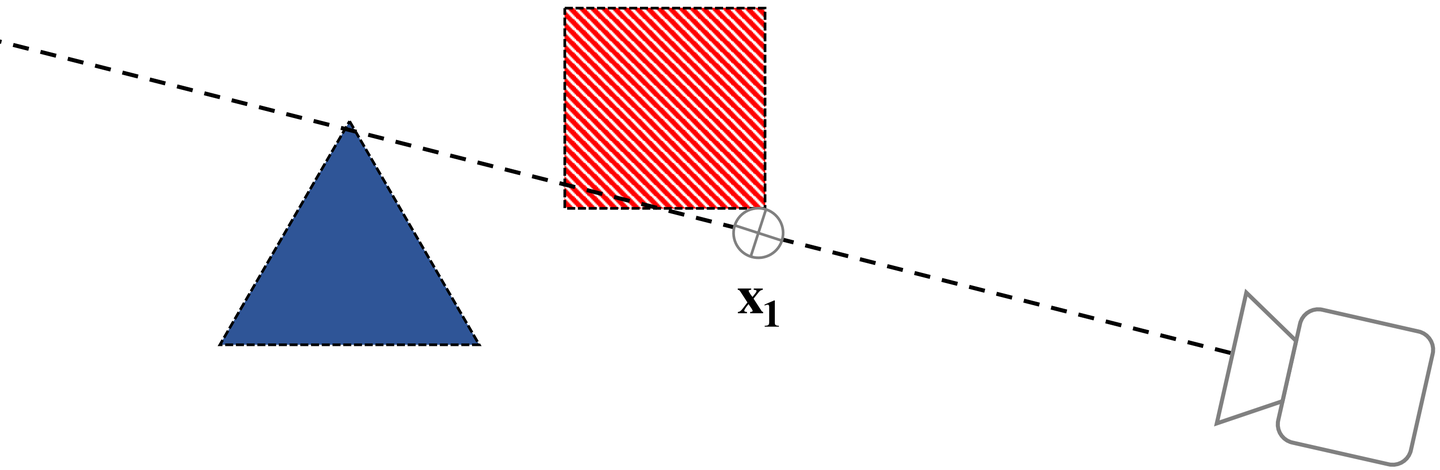
Well, if you have a true distance field (which is often not guaranteed, but that's usually swept under the rug)



Sphere Tracing Step 1: Intersection Testing.

- Guaranteed to converge to surface.
- Guaranteed to not step *through* a surface.

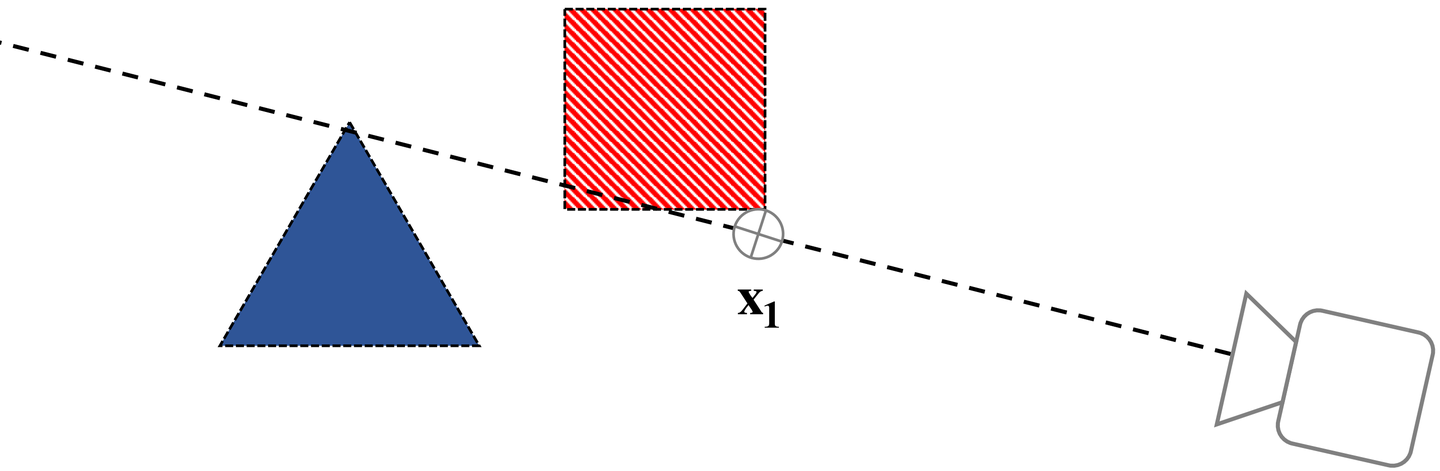
Well, if you have a true distance field (which is often not guaranteed, but that's usually swept under the rug)

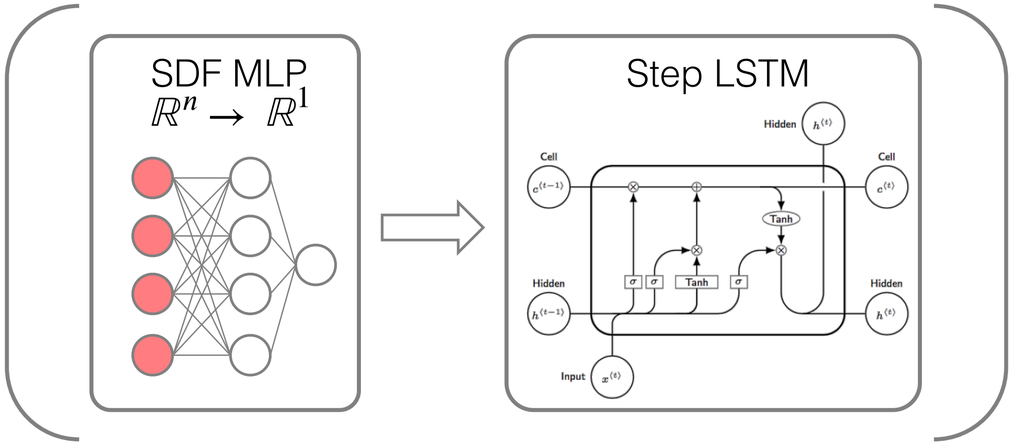


Sphere Tracing Step 1: Intersection Testing.

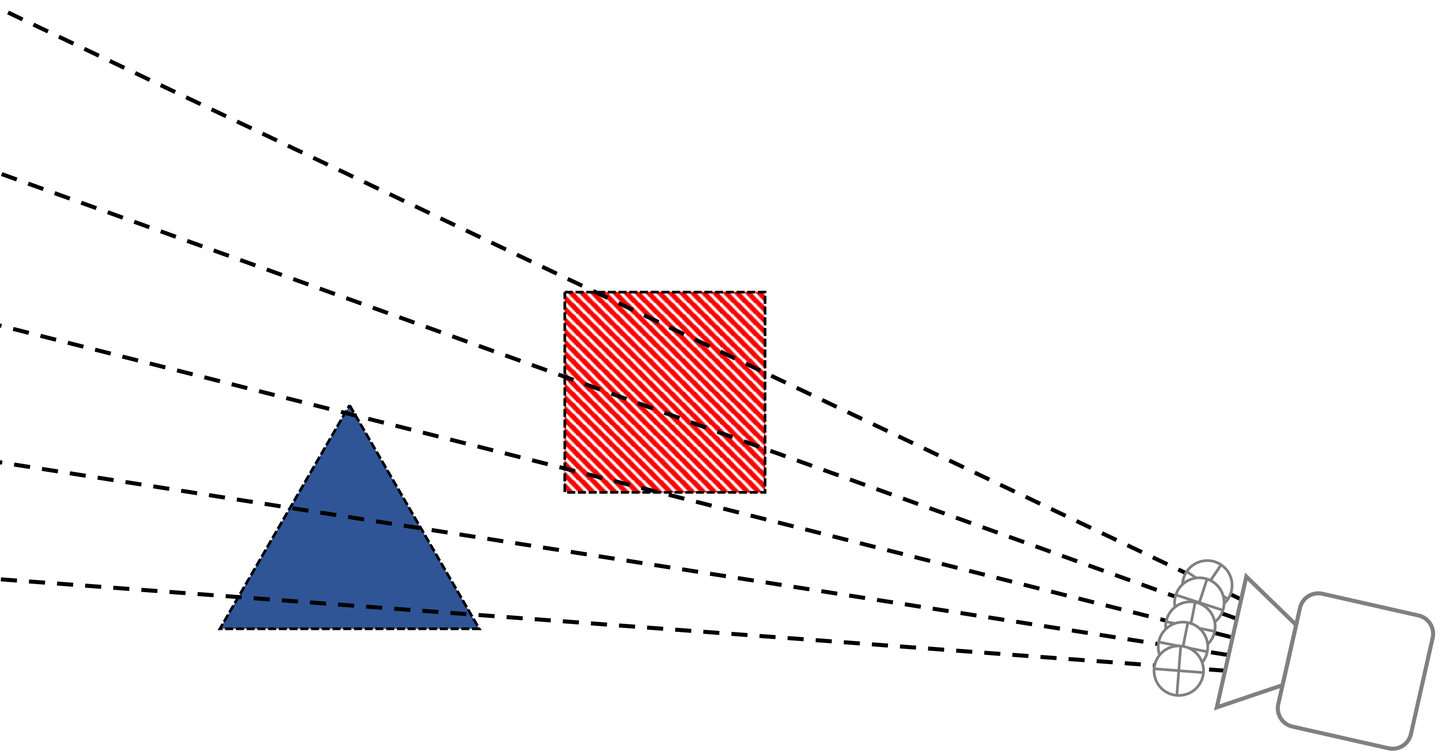
- Guaranteed to converge to surface.
- Guaranteed to not step *through* a surface.

Well, if you have a true distance field (which is often not guaranteed, but that's usually swept under the rug)

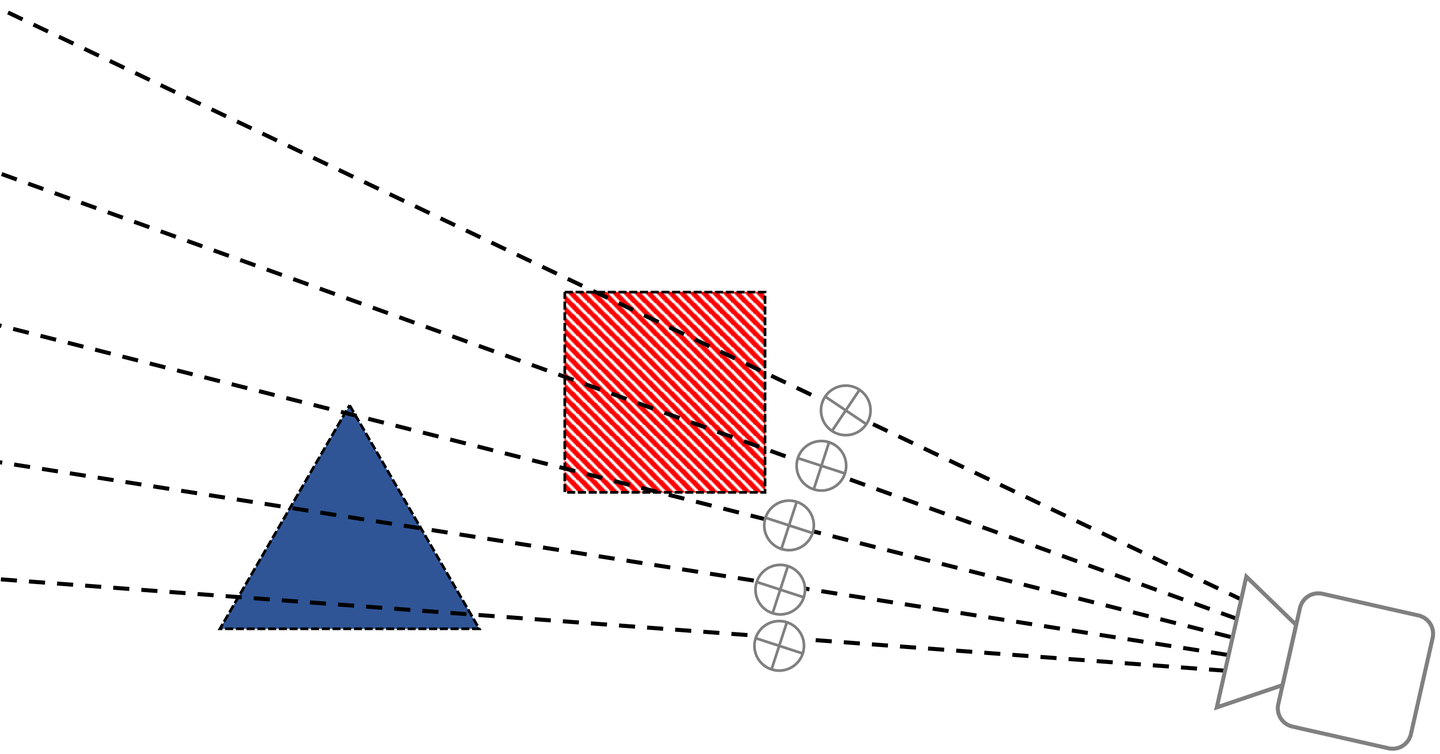




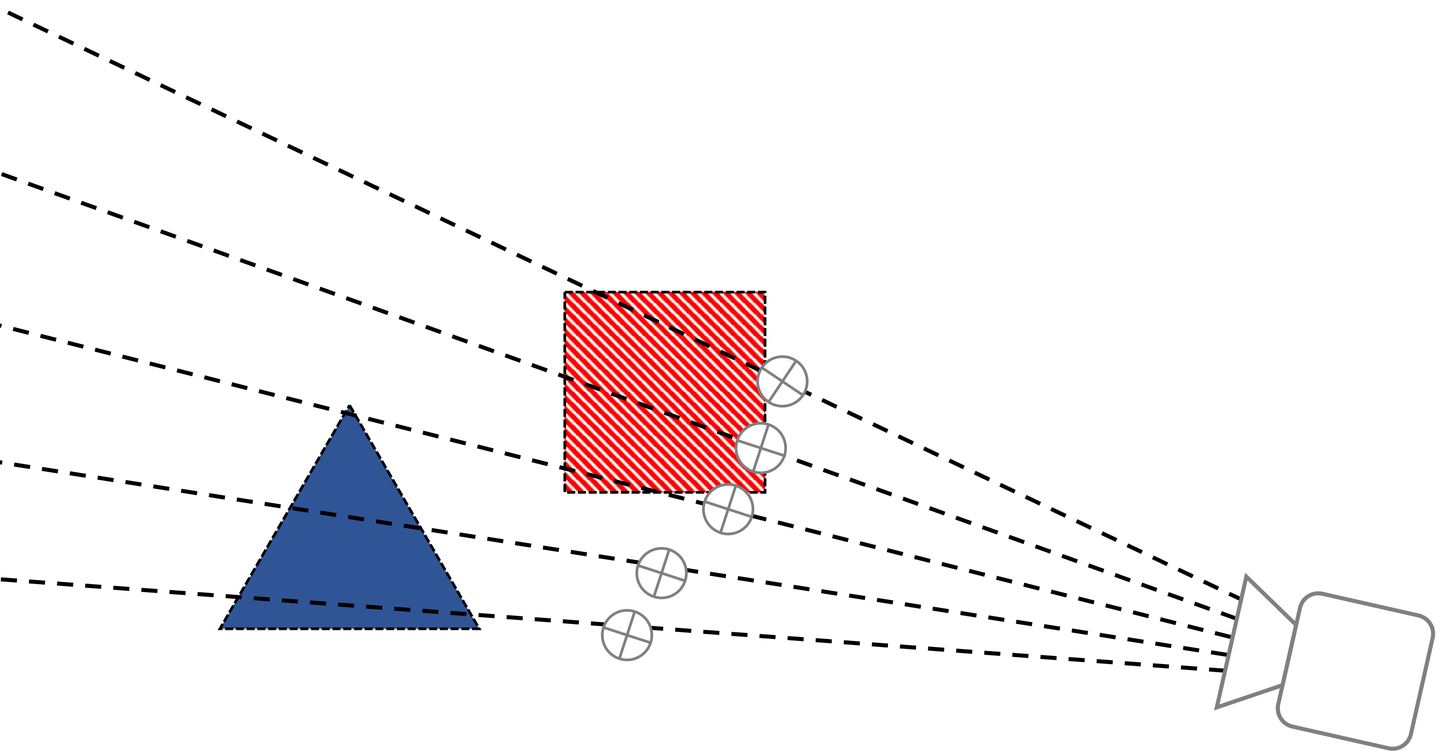
Sphere Tracing Step 1: Intersection Testing.



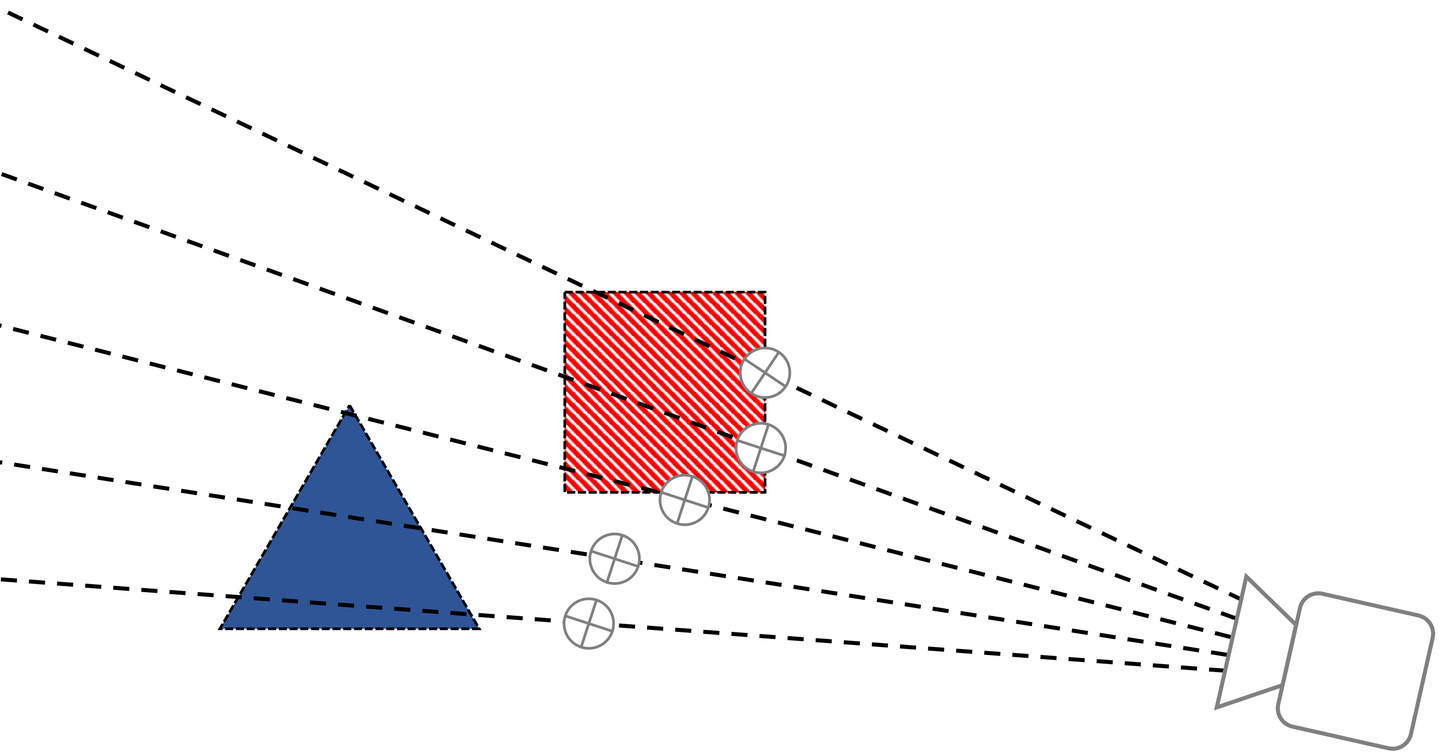
Sphere Tracing Step 1: Intersection Testing.



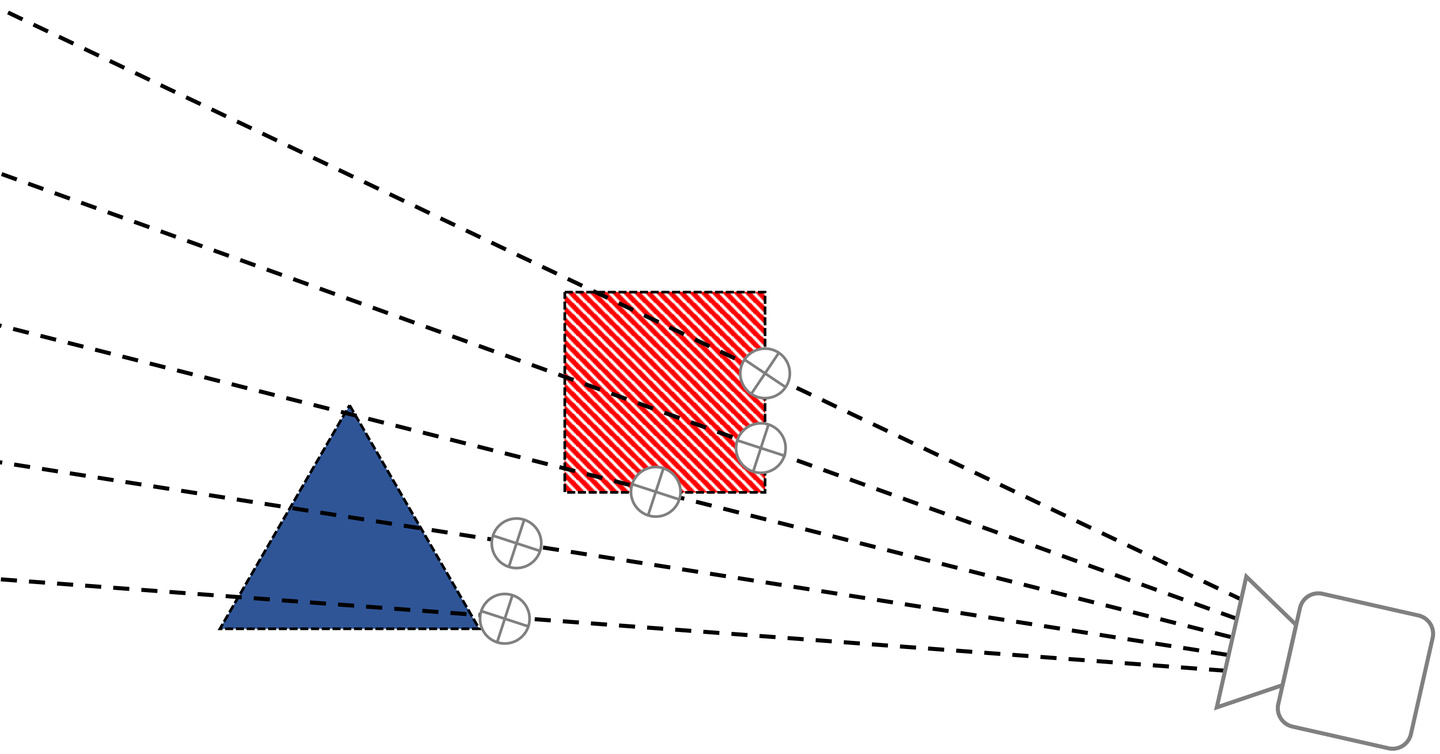
Sphere Tracing Step 1: Intersection Testing.



Sphere Tracing Step 1: Intersection Testing.

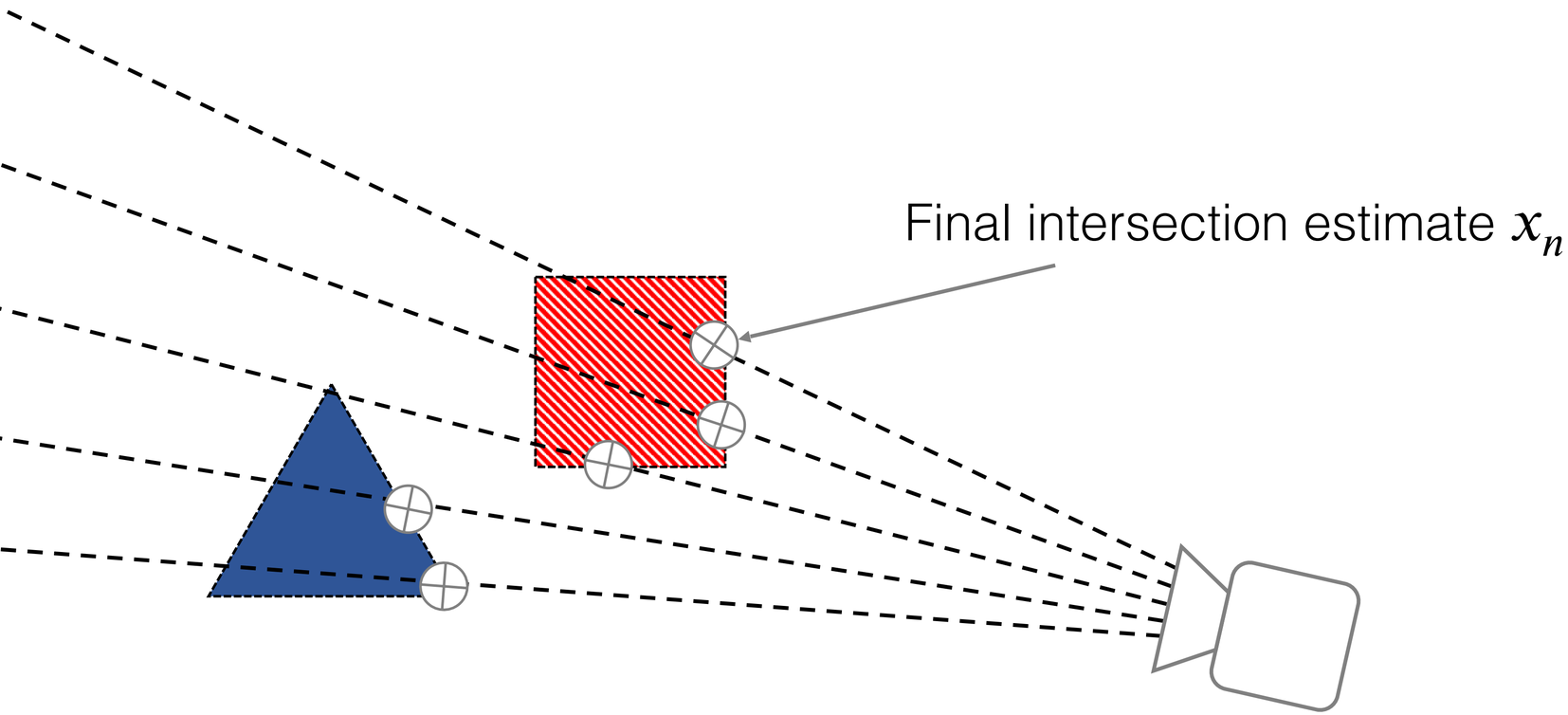


Sphere Tracing Step 1: Intersection Testing.

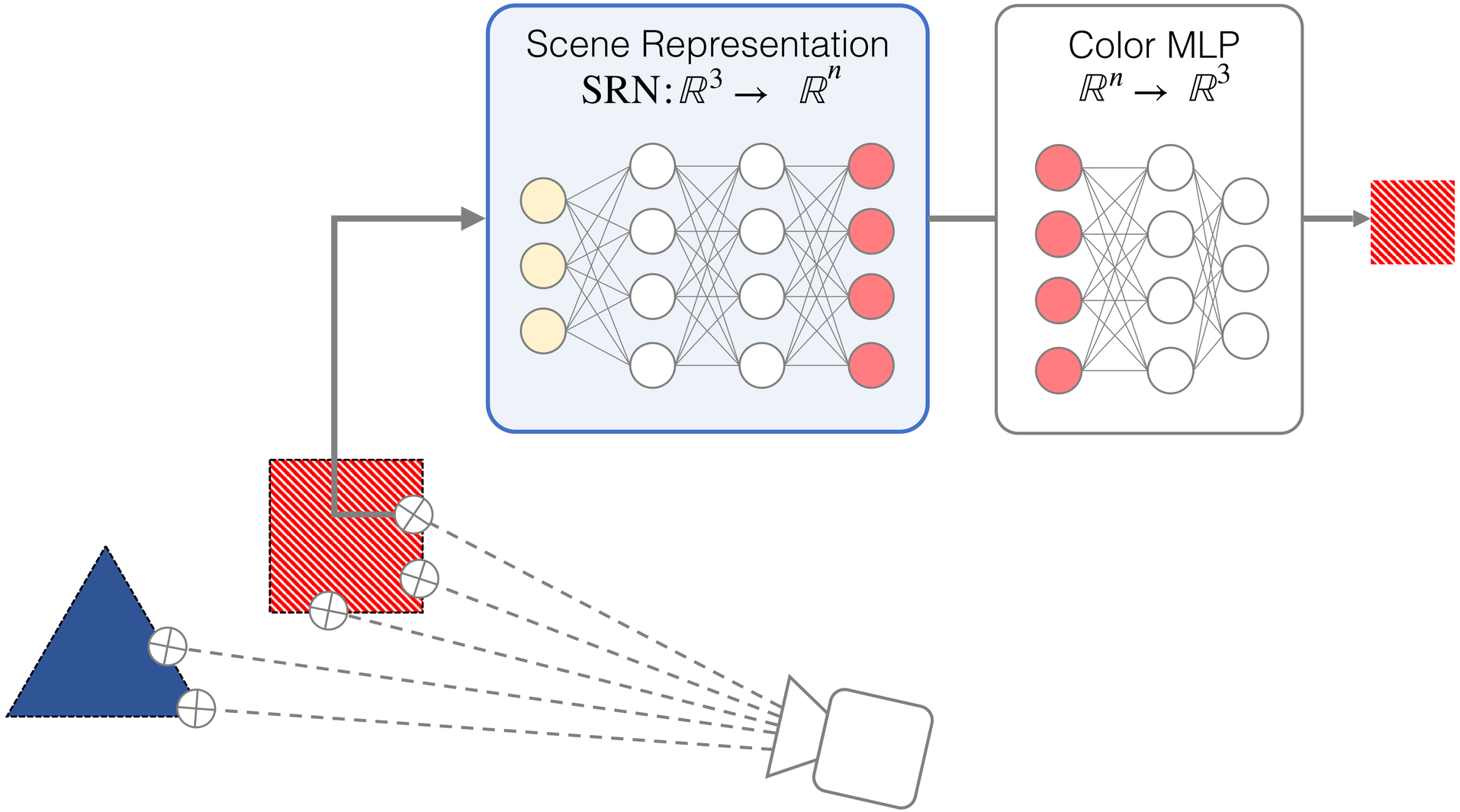


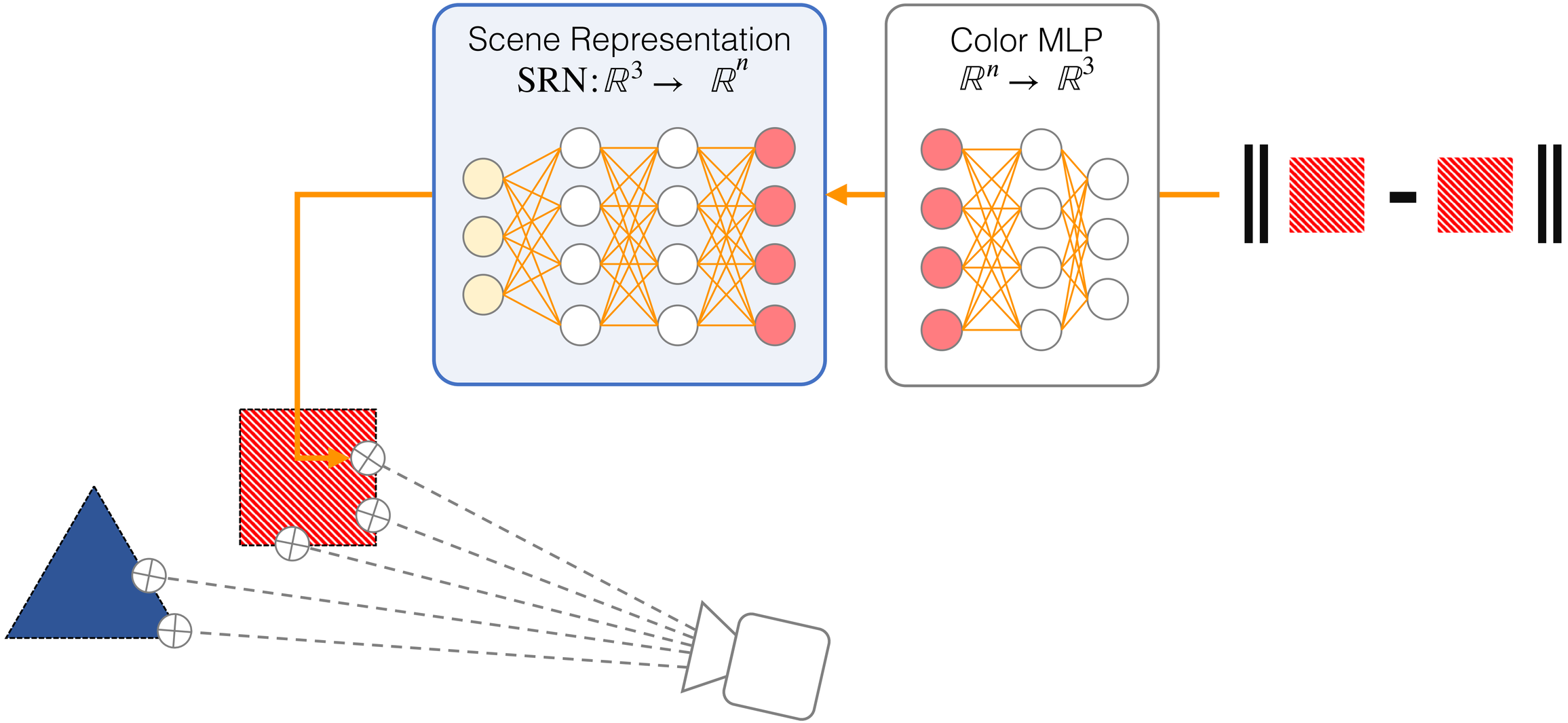
Sphere Tracing Step 1: Intersection Testing.

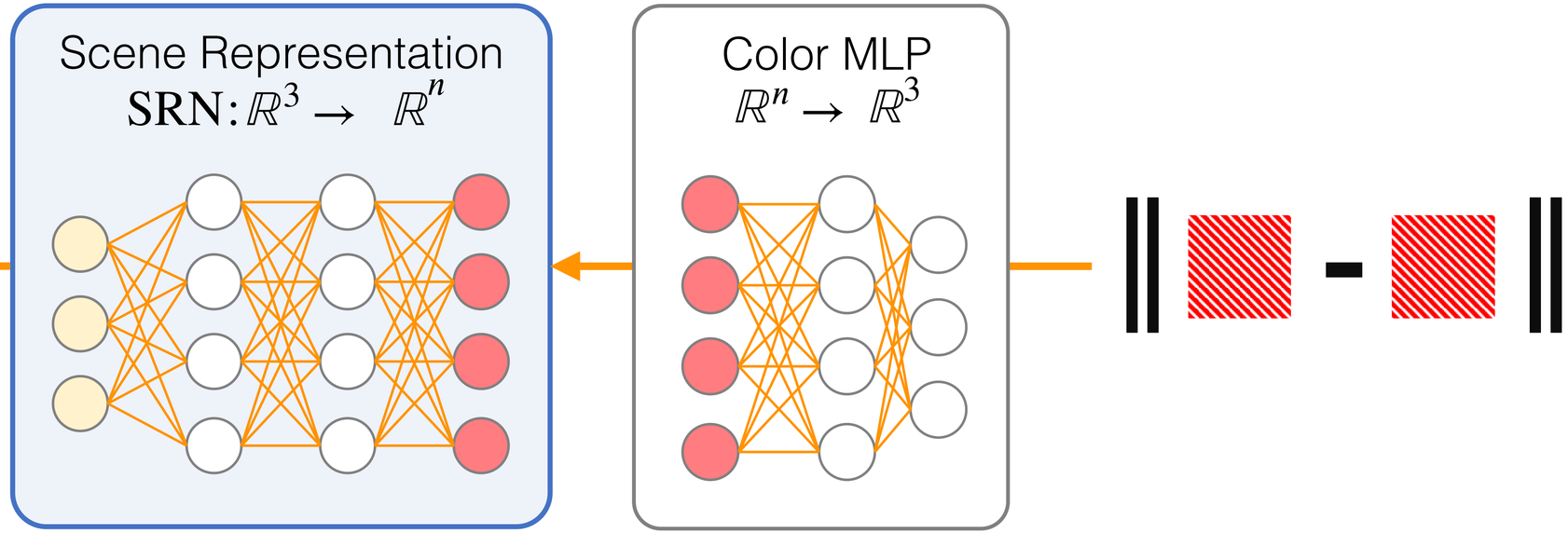
- Length of each ray yields the depth map!

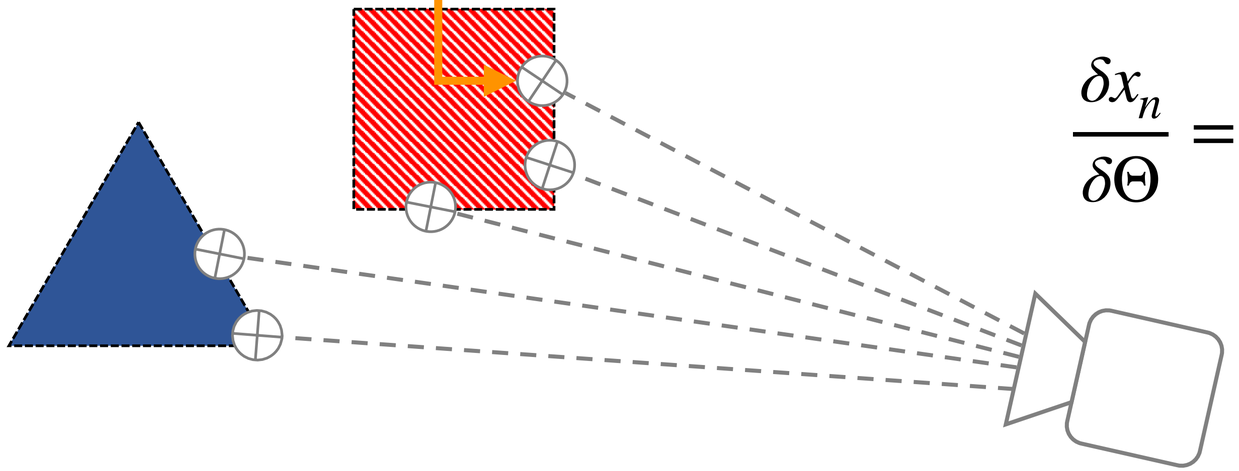


Neural Renderer Step 2: Color Generation




Scene Representation
SRN: $\mathbb{R}^3 \rightarrow \mathbb{R}^n$
Color MLP
$\mathbb{R}^n \rightarrow \mathbb{R}^3$





$$\frac{\delta x_n}{\delta \Theta} = \frac{\delta x_n - 1}{\delta \Theta} + \frac{SDF(SRN(x_{n-1}))}{\delta \Theta} = \cdots$$

- End-to-end differentiable!
- But: Not obvious how these gradients manifest zero-levelsets.

SFDF/implicit functions & differentiability

Differentiable Rendering of Neural SDFs through Reparameterization

Sai Praveen Bangaru
MIT CSAIL
USA
sbangaru@mit.edu

Michaël Gharbi
Adobe Research
USA
mgharbi@adobe.com

Tzu-Mao Li
UC San Diego
USA
tzli@ucsd.edu

Fujun Luan
Adobe Research
USA
fluan@adobe.com

Kalyan Sunkavalli
Adobe Research
USA
sunkaval@adobe.com

Miloš Hašan
Adobe Research
USA
mihasan@adobe.com

Sai Bi
Adobe Research
USA
sbi@adobe.com

Zexiang Xu
Adobe Research
USA
zexu@adobe.com

Gilbert Bernstein
MIT CSAIL &
UC Berkeley
USA

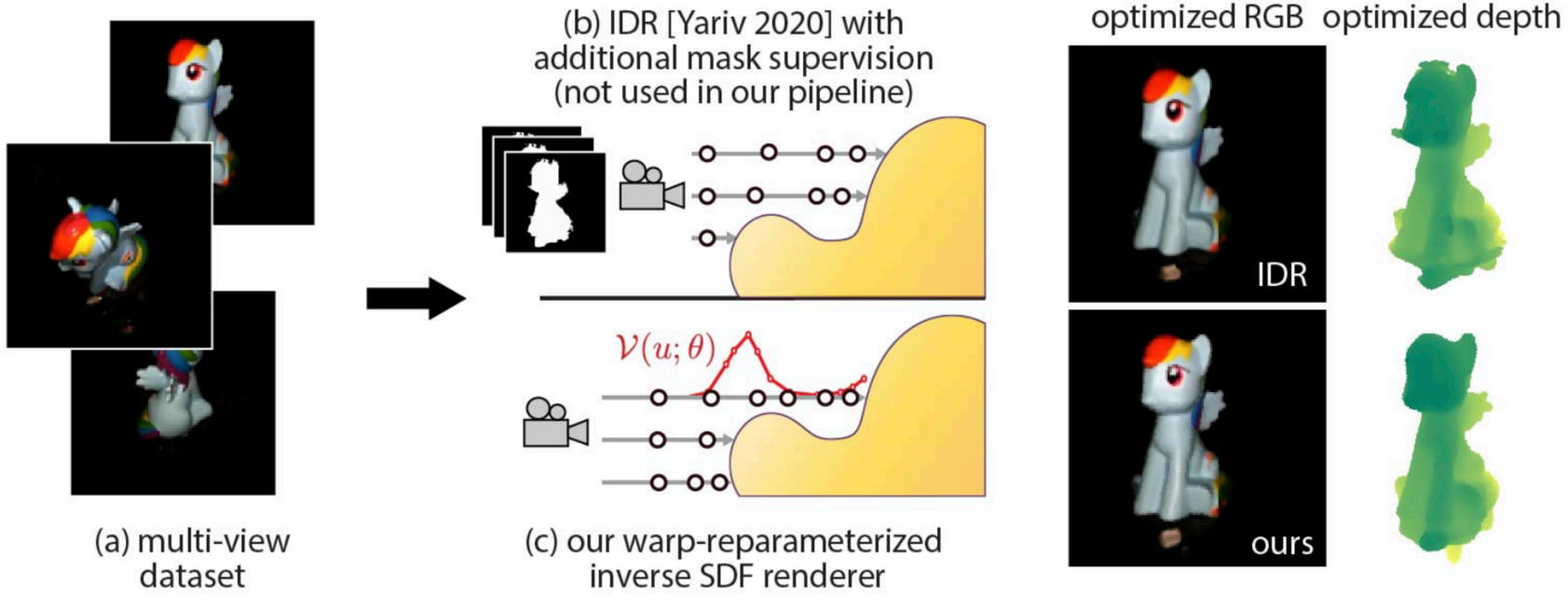
Frédo Durand
MIT CSAIL
USA
fredo@mit.edu

Same problem as triangles: discontinuities at object boundaries get missed

Most techniques “cheat” and require masks as input

Harder to fix than for triangles because we don’t have a notion of edge

See our work at https://people.csail.mit.edu/sbangaru/projects/dsdf-2022/index.html



Solution: boundary-aware warp field turns boundary integral into area integral

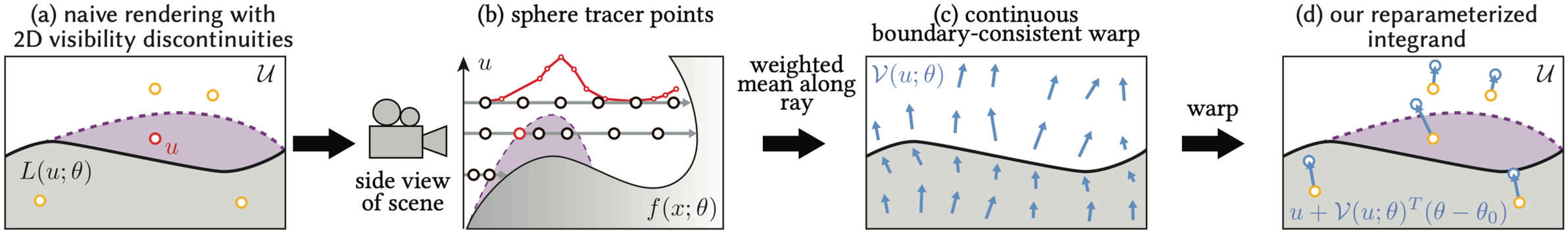


Figure 2: Overview. A standard SDF rendering pipeline is generally discontinuous, which means there are points u where the rendering function $L(u;\theta)$ is not differentiable in θ, highlighted in red (a). Our method uses intermediate points from a sphere tracer (b) applied to an SDF f, to compute a warp function $\mathcal{V}$ (c). Using this warp, we reparameterize the integration domain to avoid discontinuities (d), which allows us to compute correct gradients of the rendering equation. The key to achieving this is to design the warp $\mathcal{V}$ so it is continuous in u everywhere, and satisfies some consistency criterion on the geometric boundaries.

Further reading on computing gradients of sphere-tracing based renderers

Actually messier than it seems because of boundary discontinuities

Differentiable Rendering of Neural SDFs through Reparameterization
Bangaru et al. 2022

Differentiable Volumetric Rendering
Niemeyer et al. 2020

Differentiable signed distance function rendering
Vicini et al. 2022

Other ray marching acceleration

Original NERF: 2-scale approach

Coarse level : 64 uniform samples

Fine level: 128 additional samples

  importance-sampled based on the coarse level

Instant NGP

Skip empty voxels

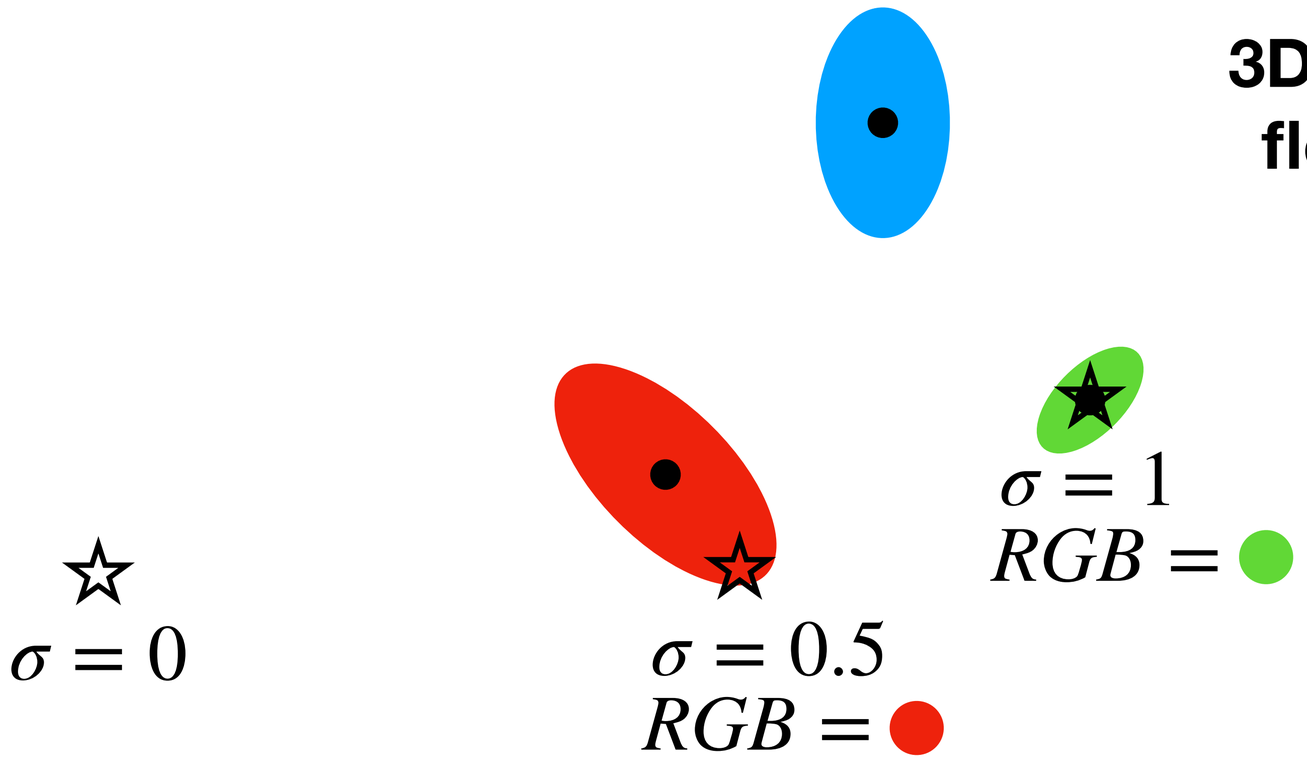
Maintain an occupancy grid in addition to neural feature grids

- Multiscale, exponential spacing to handle infinite scenes
- Maintained by querying density from features + MLP

“Pragmatic”

3D Gaussian Splatting

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



Anisotropic Volumetric 3D Gaussians



Final Rendering

3D Gaussian Visualization

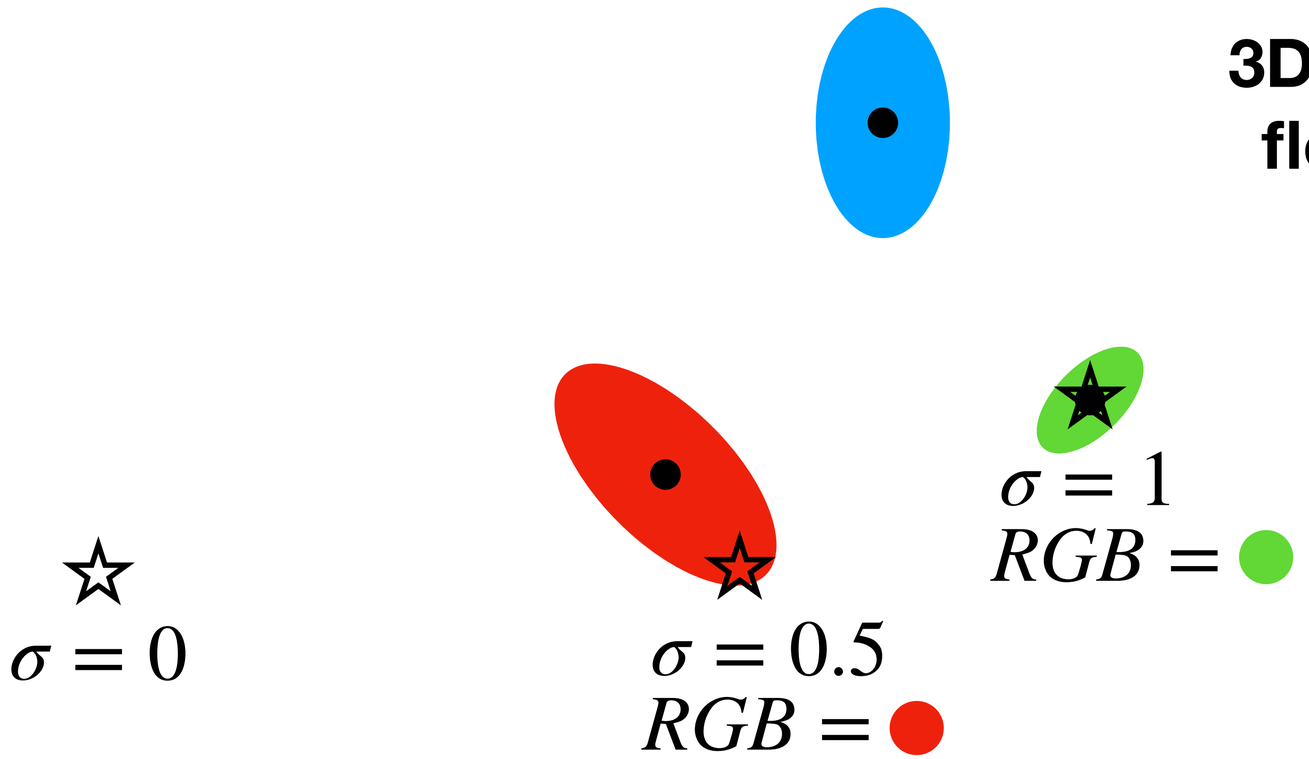
Anisotropic Volumetric 3D Gaussians



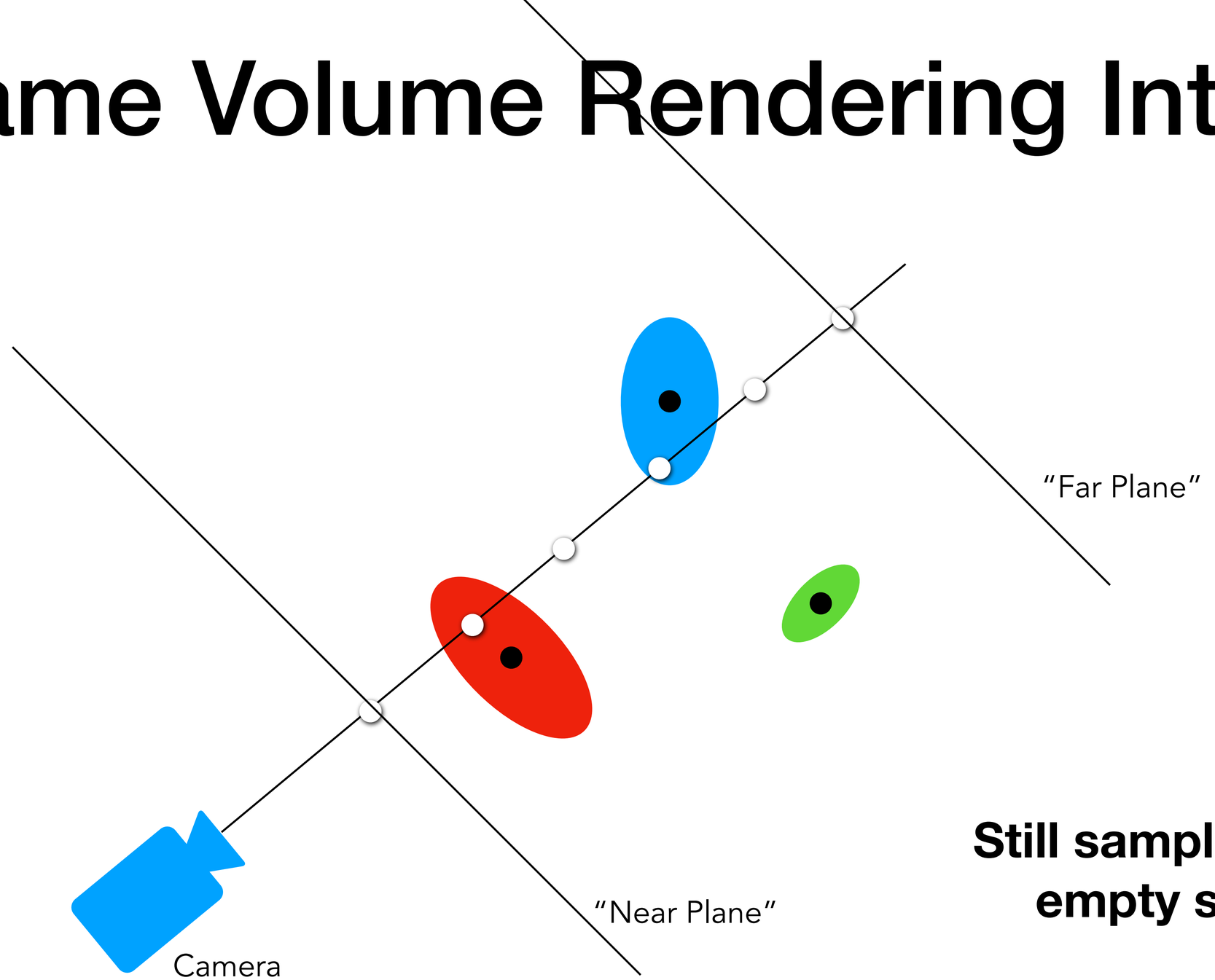
Final Rendering

3D Gaussian Visualization

How to Render?



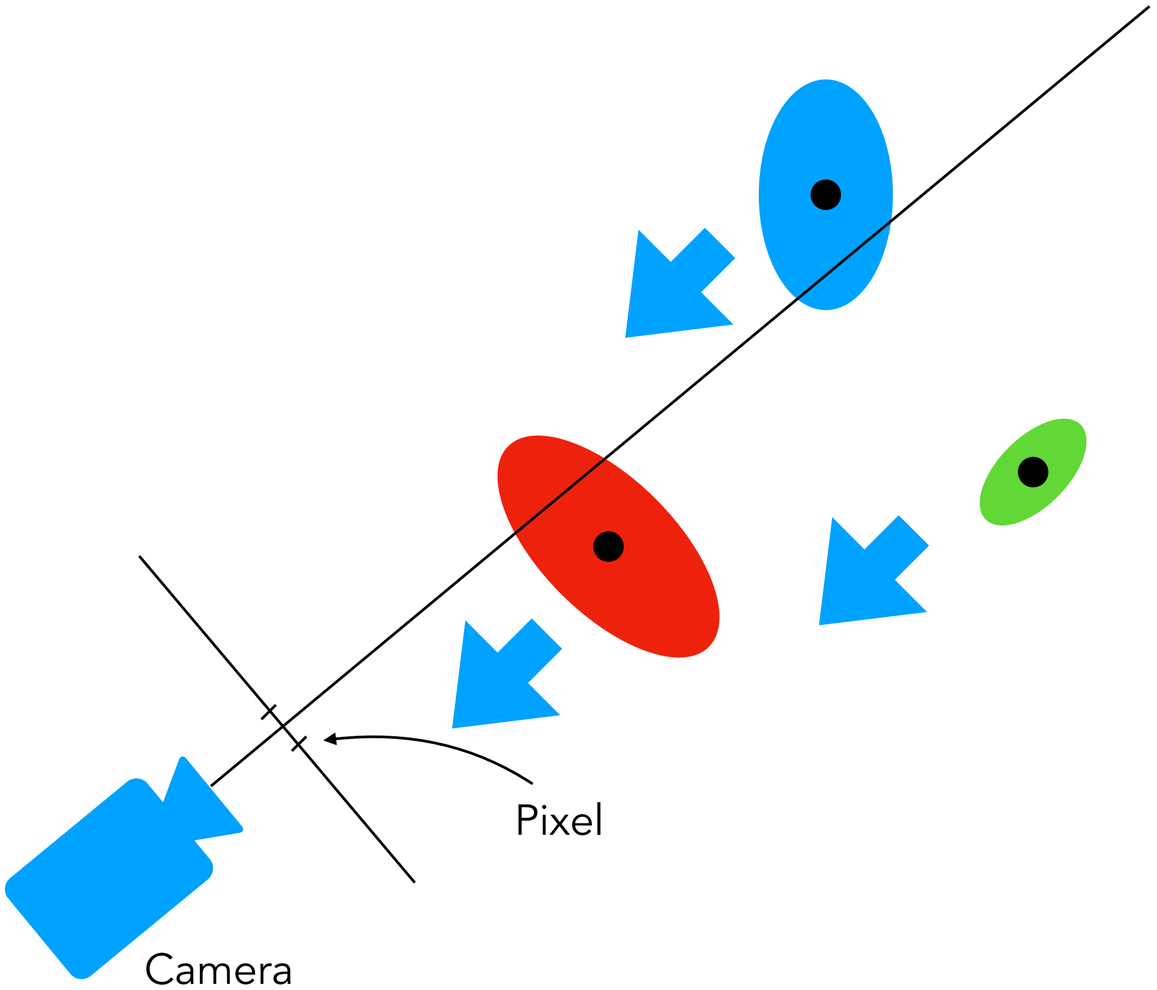
Same Volume Rendering Integral!



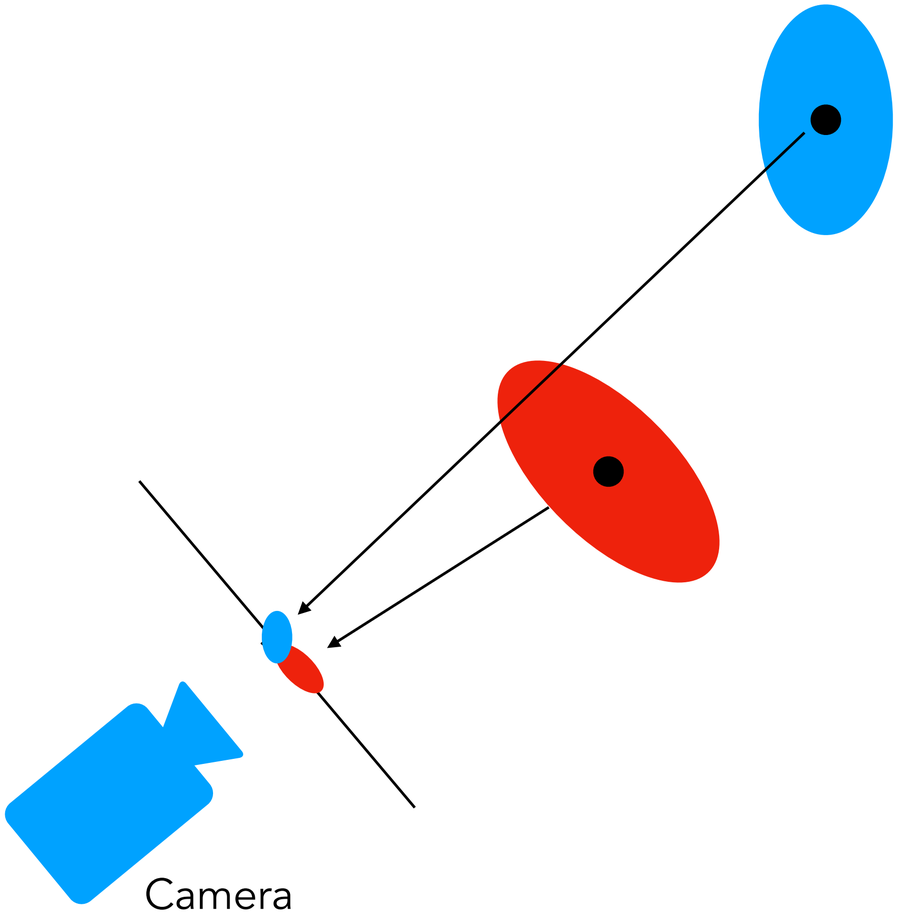
Still sampling lots of empty space…

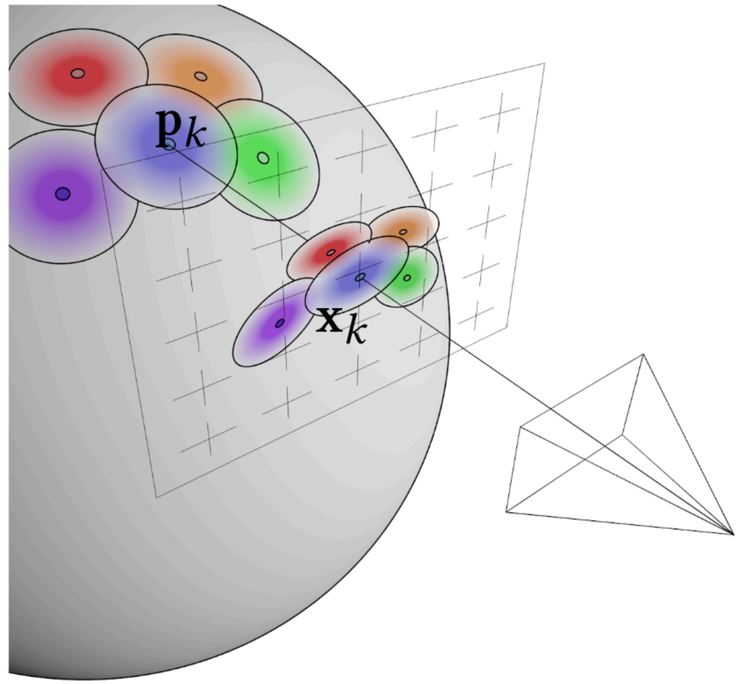
Instead: Rasterization

Project onto image primitive by primitive

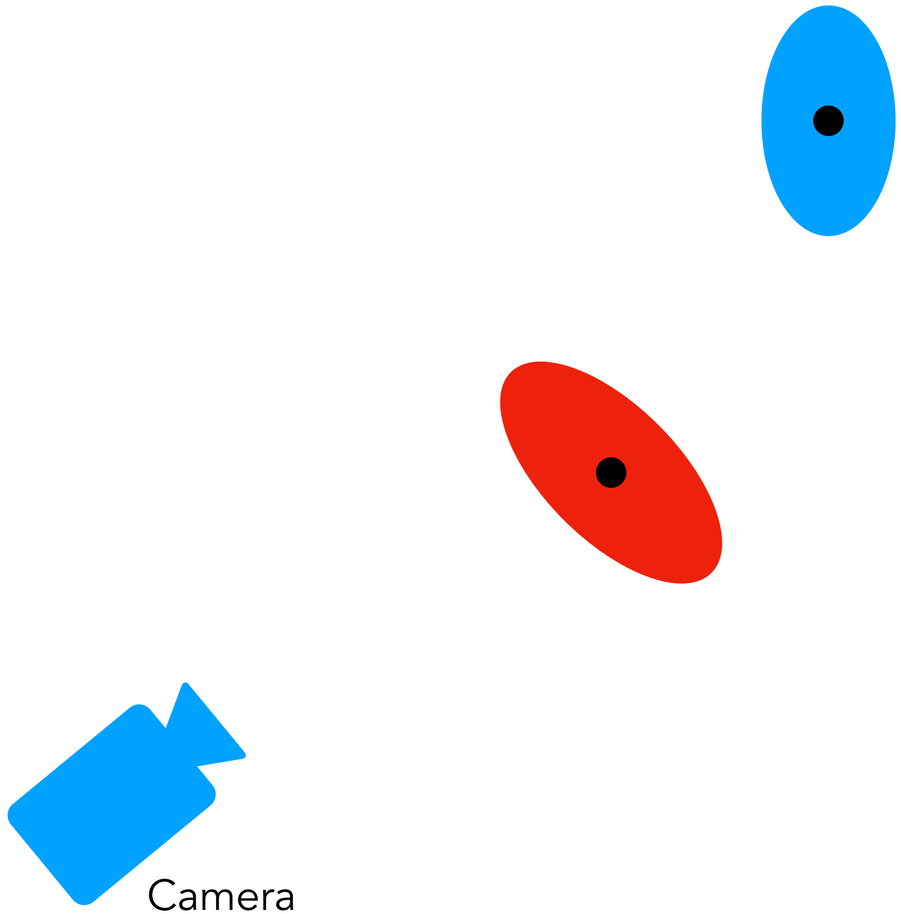


Projected 3D Gaussian makes 2D Gaussian!





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!):

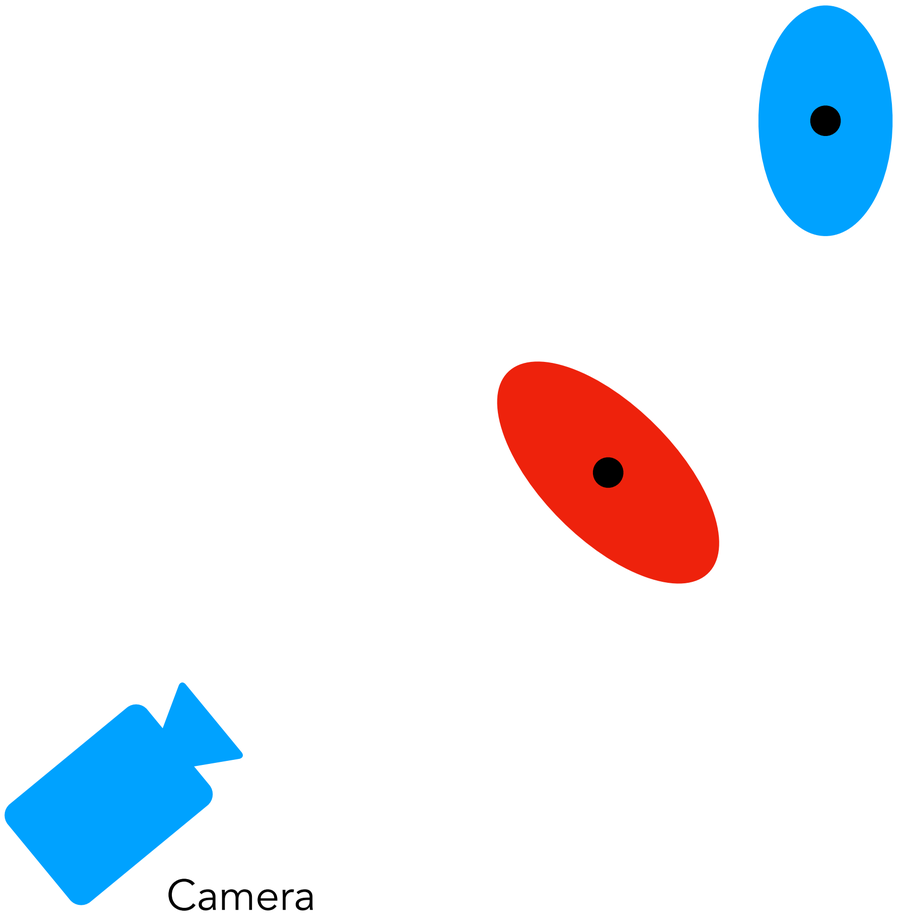
$$\mathcal{G}_{\mathbf{V}}(\Phi^{-1}(\mathbf{u}) - \mathbf{p}) = \frac{1}{|\mathbf{M}^{-1}|}\mathcal{G}_{\mathbf{M}\mathbf{V}\mathbf{M}^T}(\mathbf{u} - \Phi(\mathbf{p}))$$

Integrate along axis:

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

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

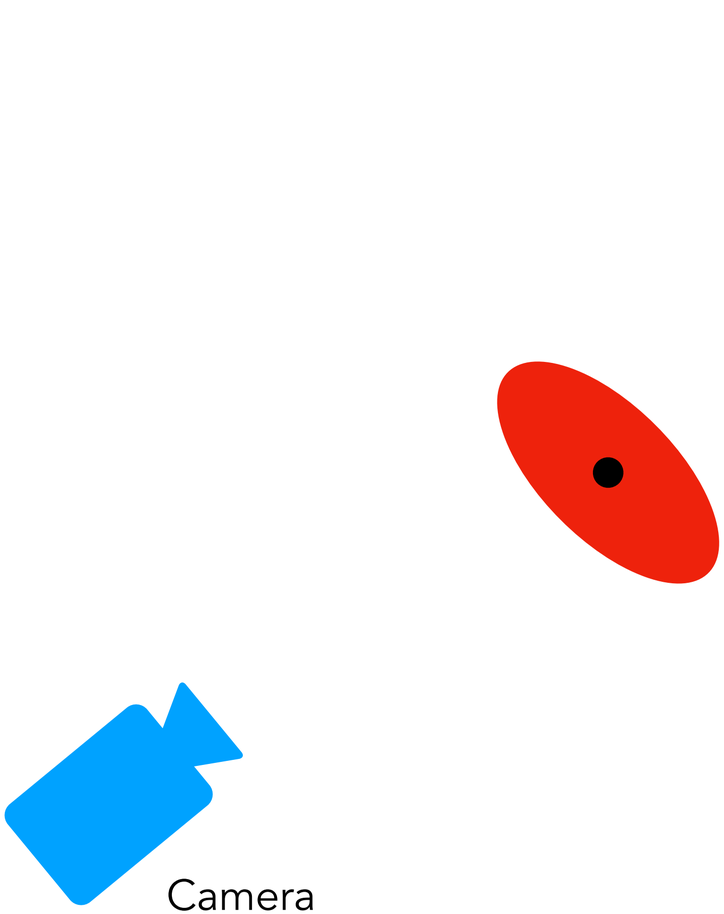
Step 1: Transform Gaussians into Camera Coordinates



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 :/

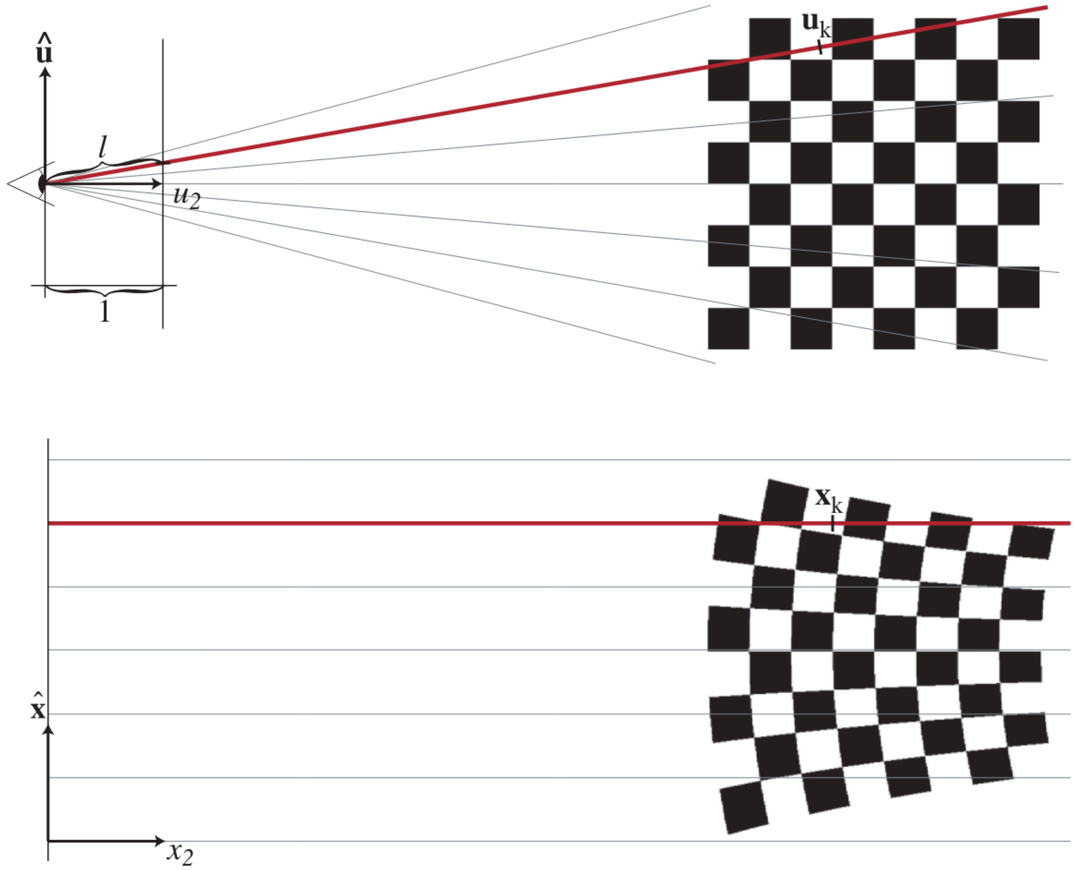
$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \mathbf{m}(\mathbf{u}) = \begin{pmatrix} u_0/u_2 \\ u_1/u_2 \\ \|(u_0, u_1, u_2)^T\| \end{pmatrix}$$

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

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:

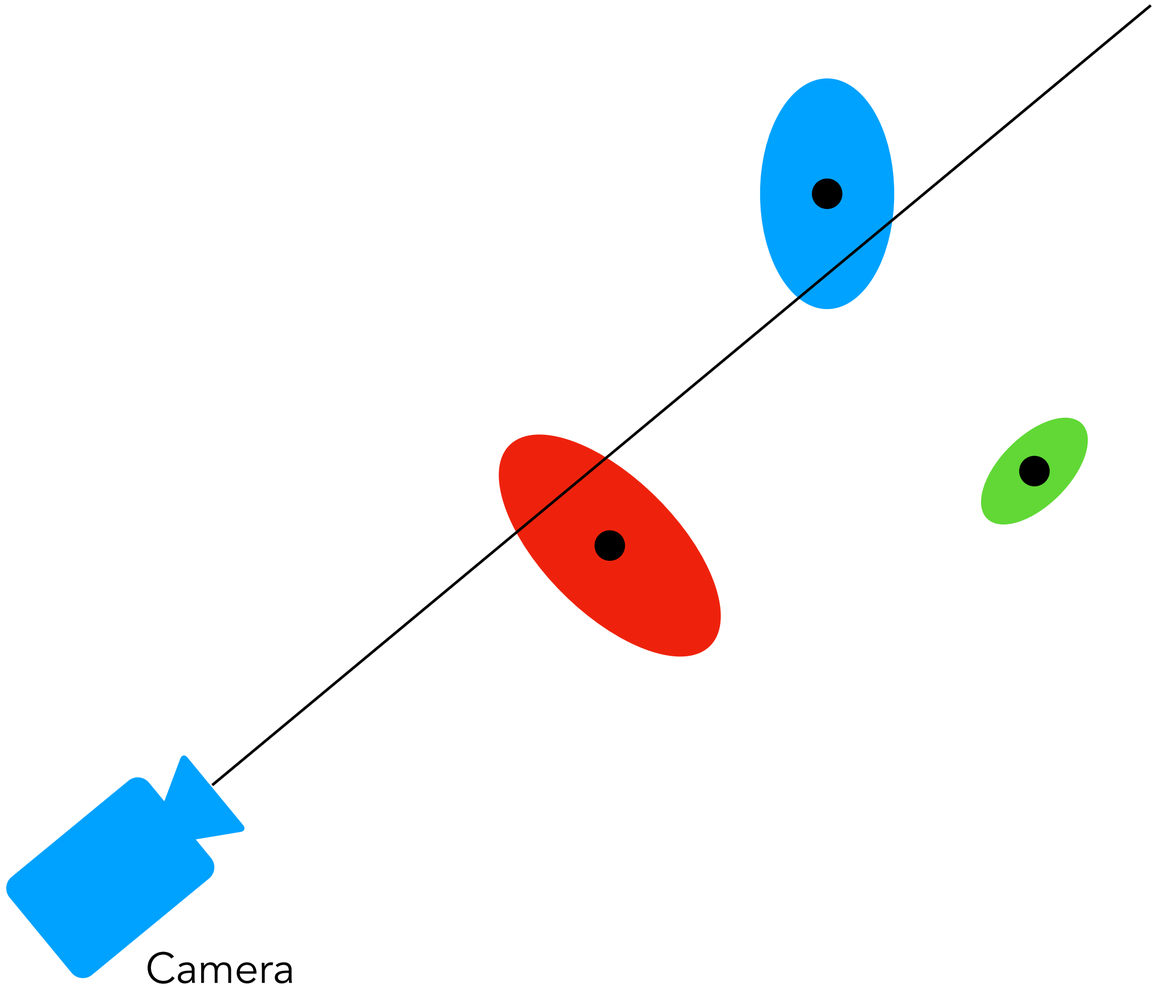
$$\frac{1}{|\mathbf{W}^{-1}||\mathbf{J}^{-1}|}\mathcal{G}_{\mathbf{V}_k}(\mathbf{x} - \mathbf{x}_k)$$

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

Finally, can integrate along rays:

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

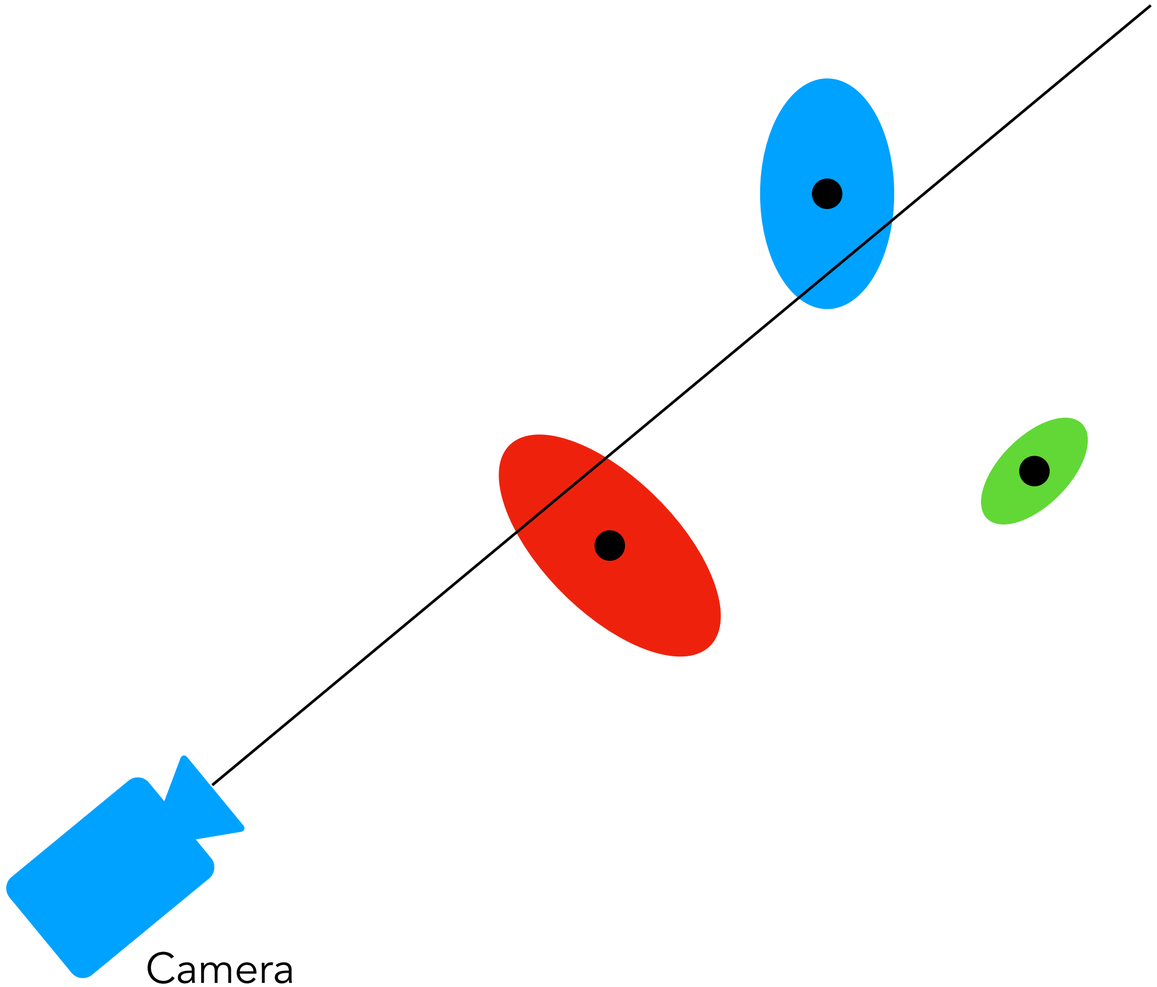
Extra ingredient: sort front to back



Equivalent to ray marching,
but global for the whole scene/image

In practice implemented per tile

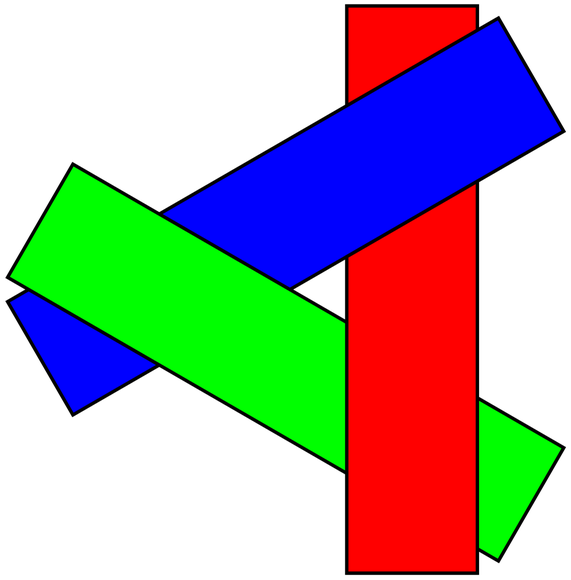
Extra ingredient: sort front to back



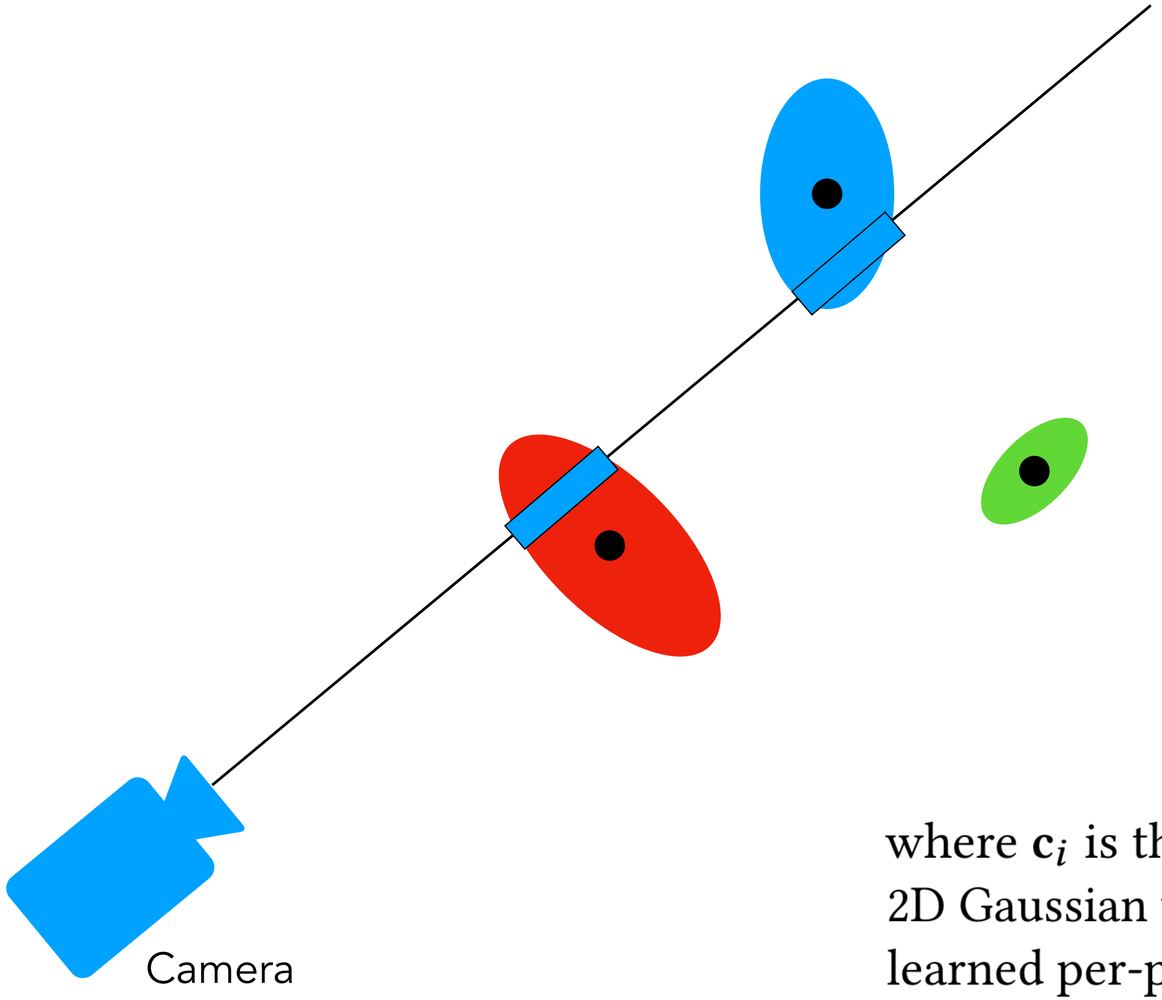
Equivalent to ray marching,
but global for the whole scene/image

Detail: no guarantee to have a global order

Usually ignored



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



Caveat: not fully correct when Gaussians overlap

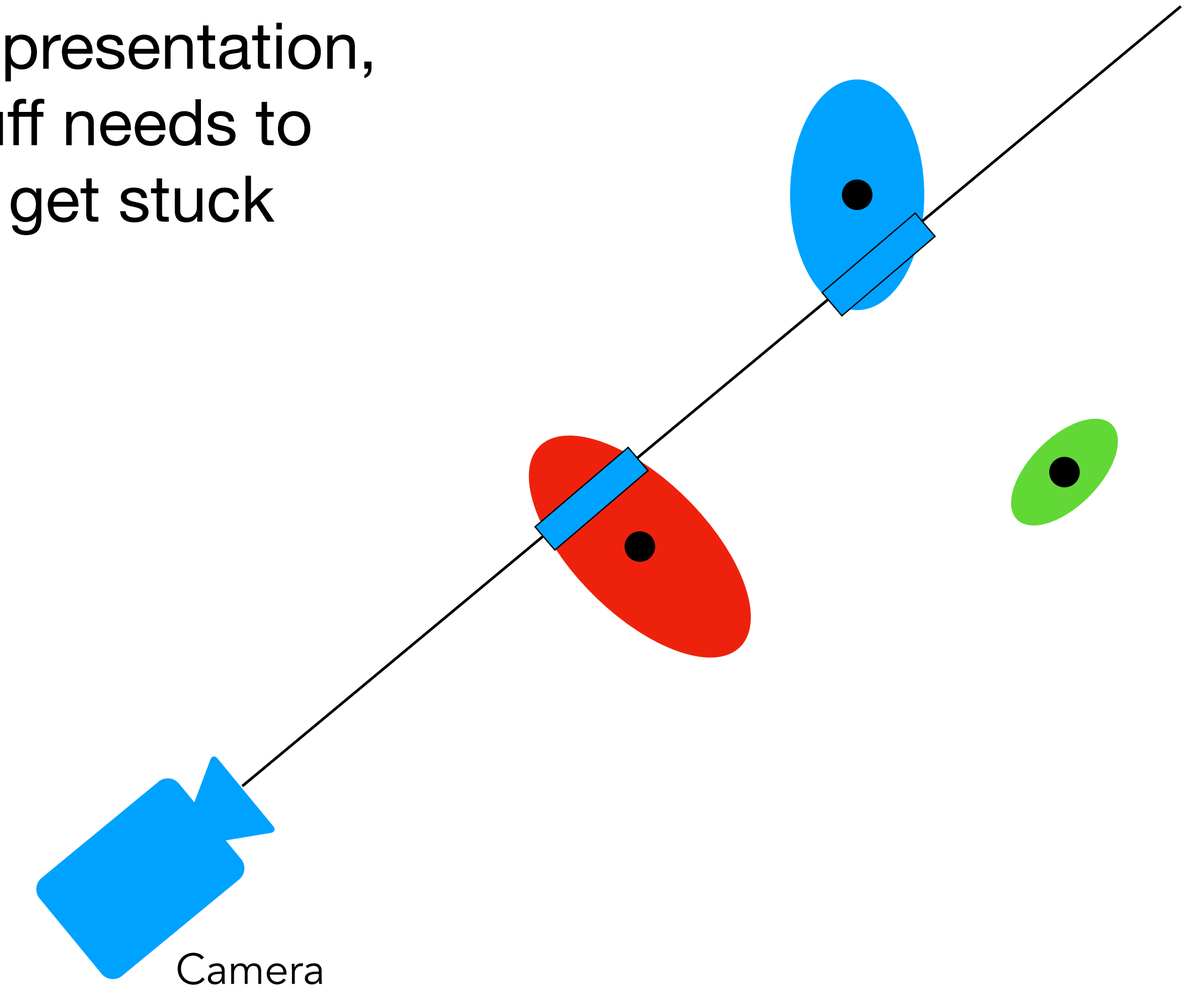
Compounded by the issue of sorting

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

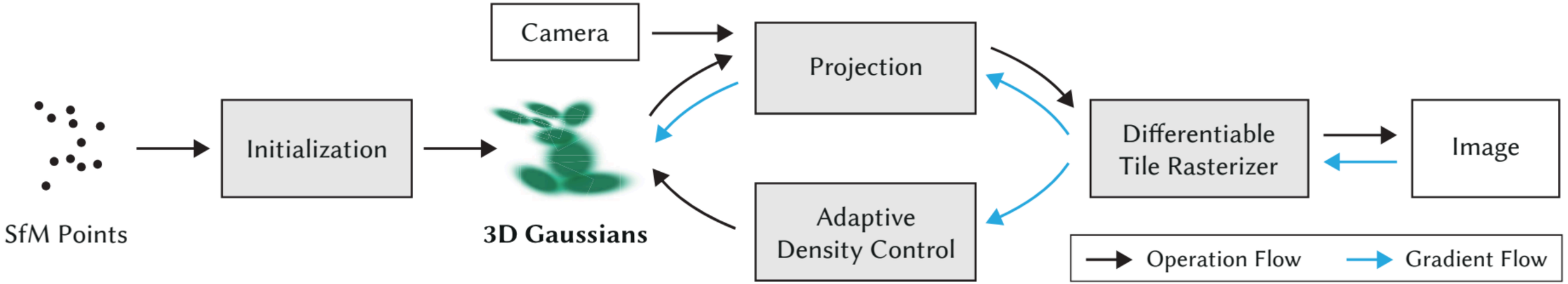
where $\mathbf{c}_i$ is the color of each point and α_i is given by evaluating a 2D Gaussian with covariance Σ [Yifan et al. 2019] multiplied with a learned per-point opacity.

Problem: Local minima…

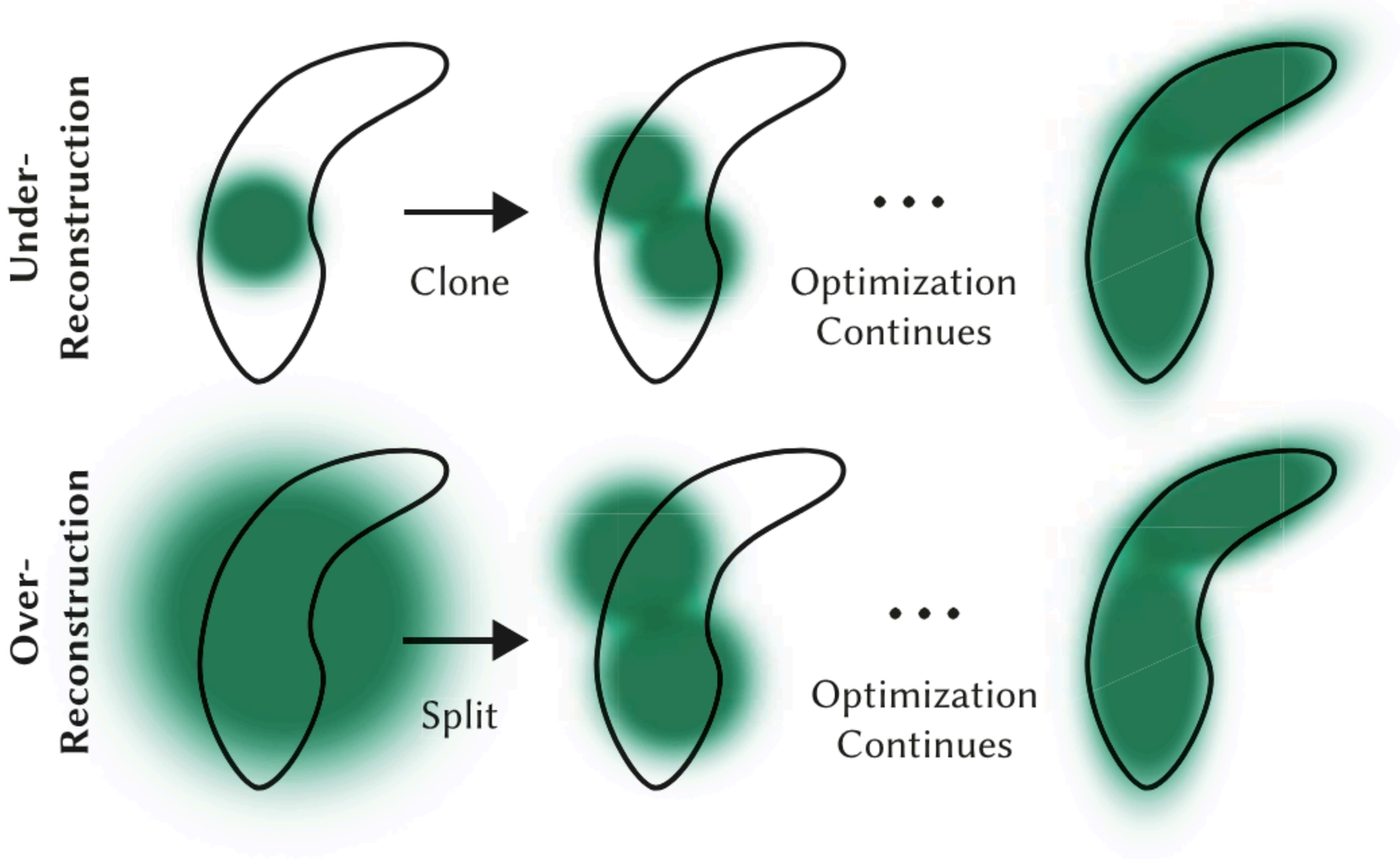
It’s a Lagrangian representation, loss is Eulerian, stuff needs to move around, may get stuck



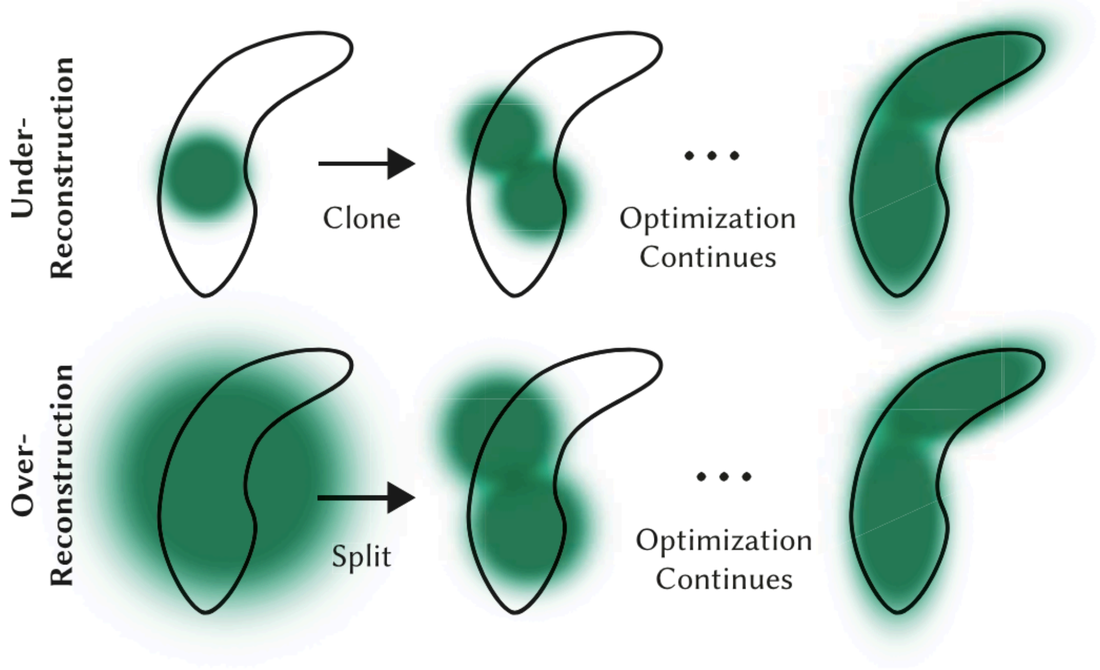
Fix 1: Start from SFM point cloud.



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

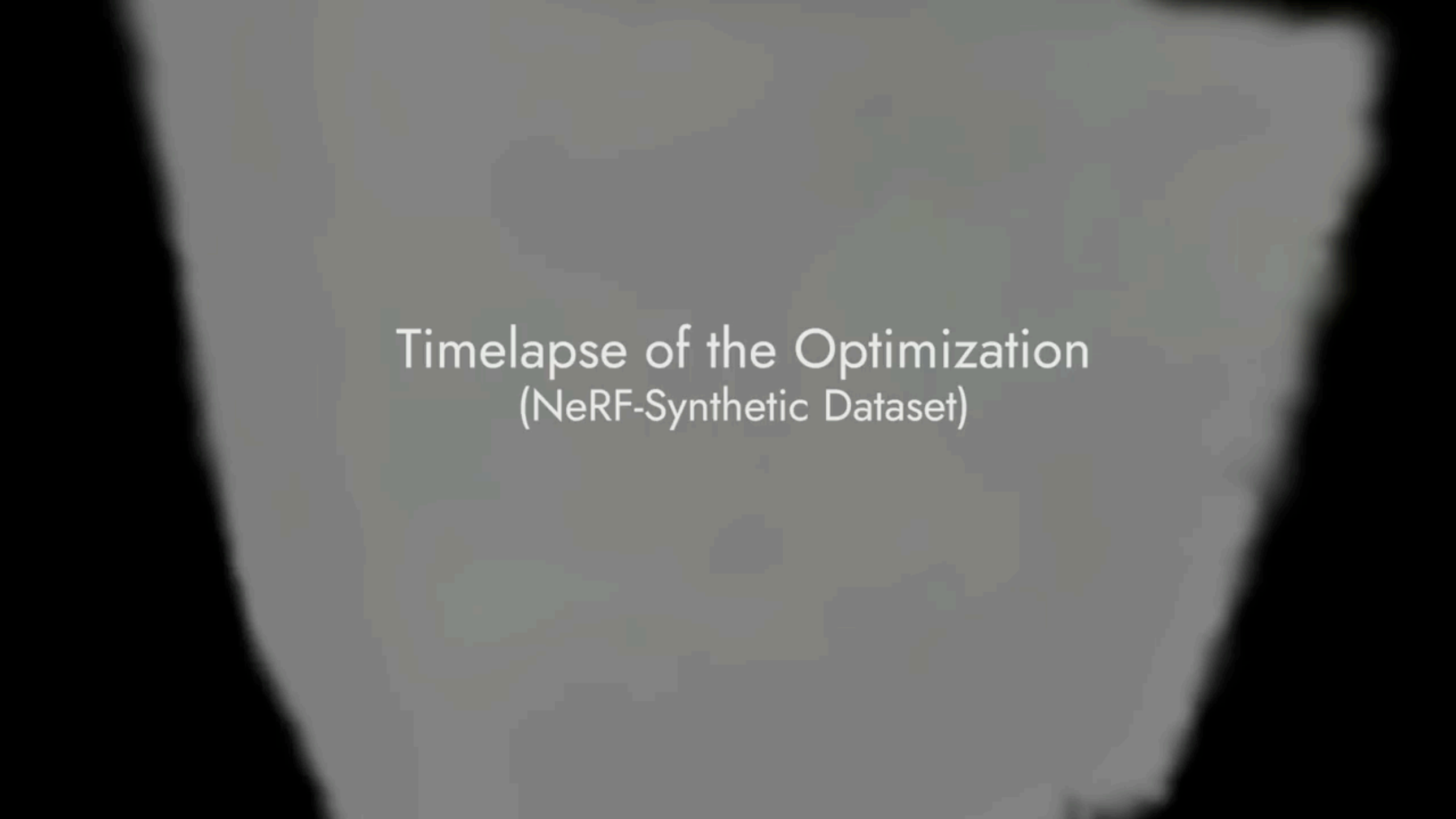


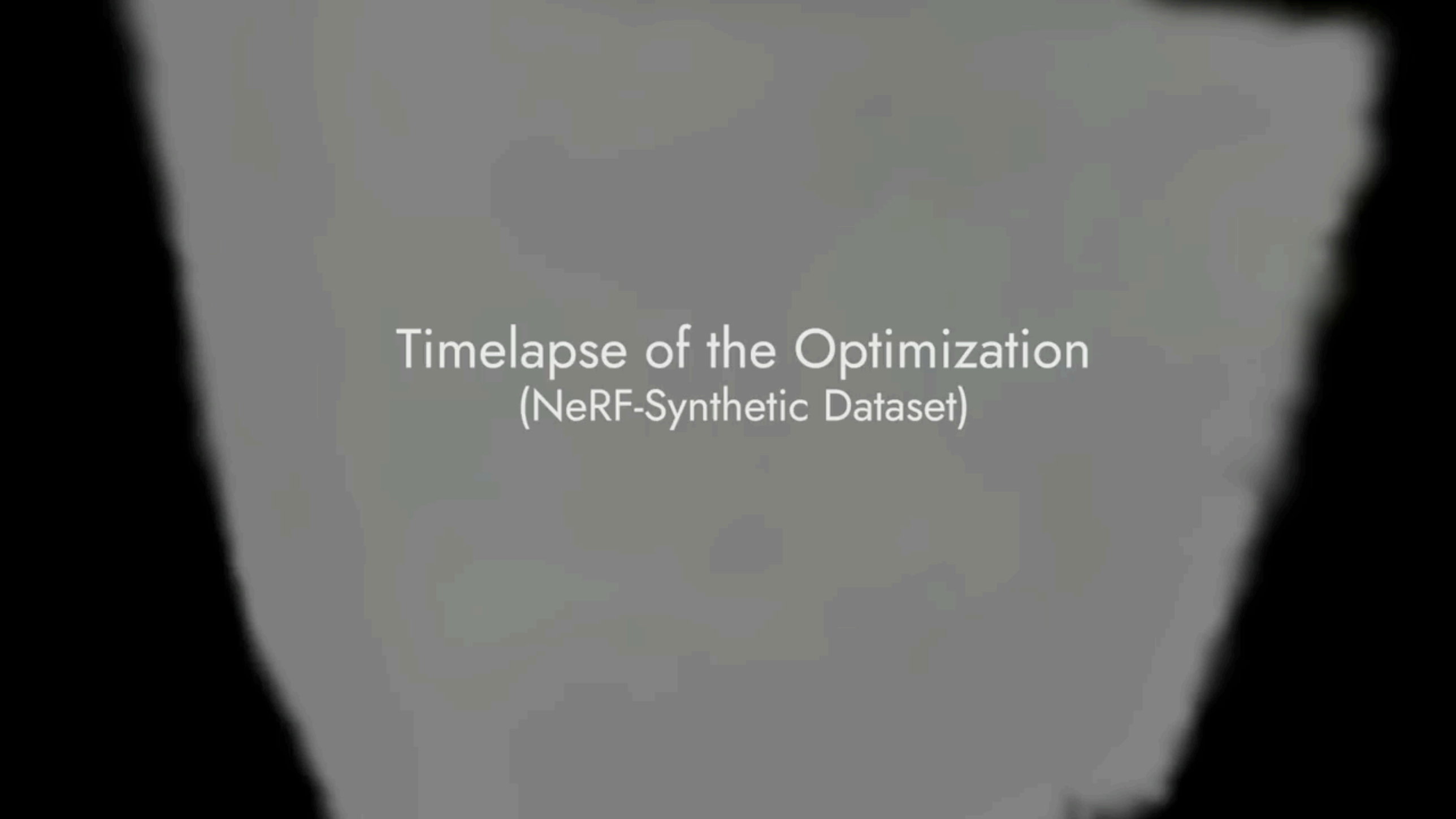
Fix 2: Heuristic *pruning* and *spawning* operations



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