Neural Radiance Fields

CS294-158

Spring 2024

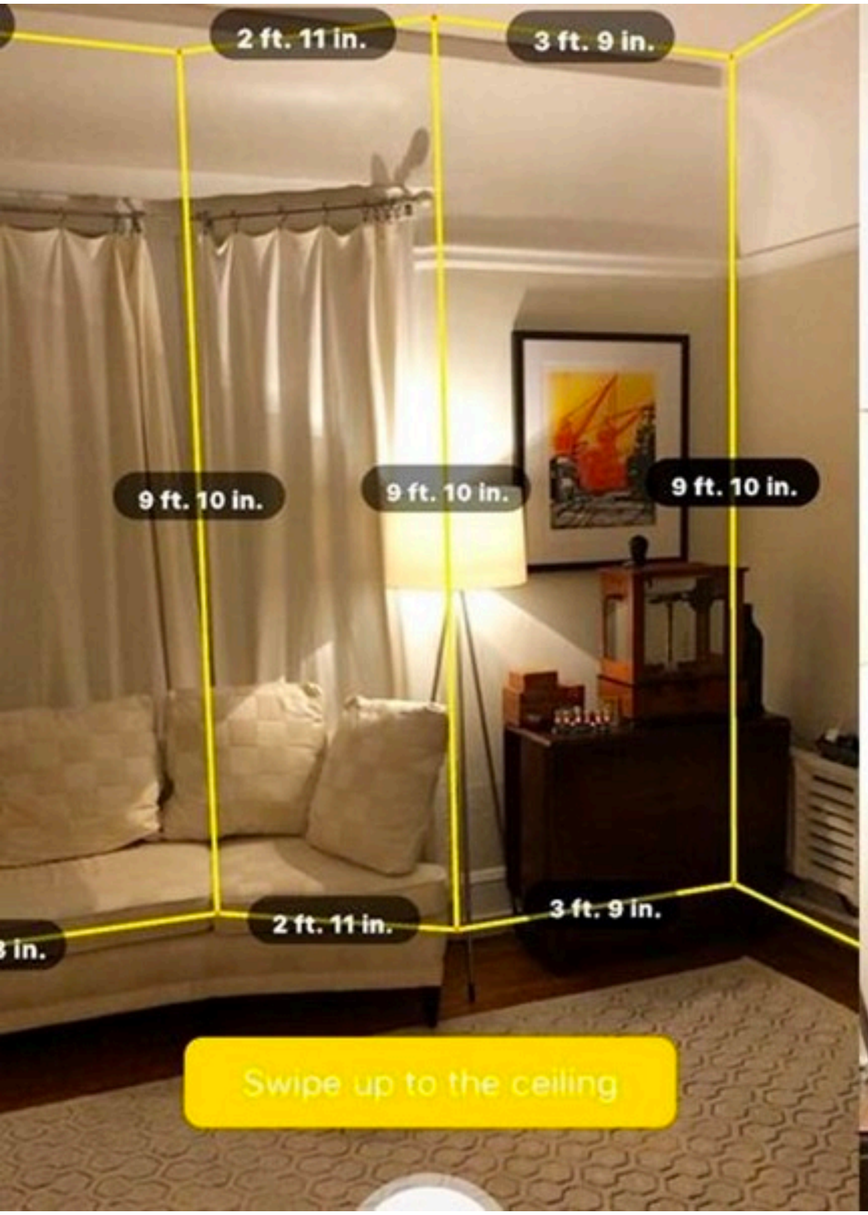
Overview

- NeRF (neural radiance fields) for view synthesis
  - Volume rendering
  - Scene representation
- Reconstruction and generation
  - DreamFusion: generation via reconstruction
  - ReconFusion: reconstruction via generation

What is 3D for?



Measurement

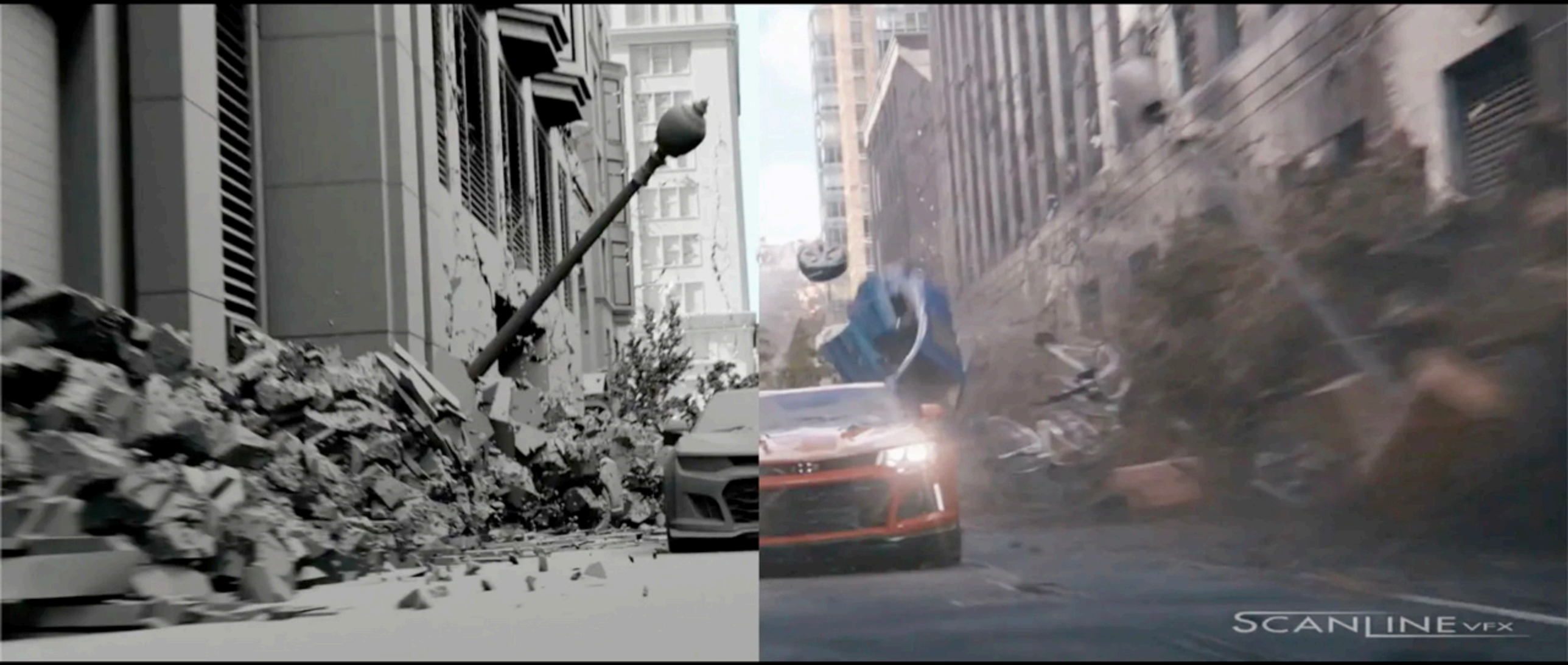






Measurement


Simulation



Simulation



Interface


Messages
Search
Ready to book?
Find your room in
Belfort
NOVELA
NOOK
Explore and Book
All iCloud

Interface

Reconstruction vs Generation

- Reconstruction: take lots of observations or measurements, invert them to recover underlying signal
- Generation: take a few high-level input signals, sample from the conditional distribution learned from a large dataset

Consistency and Controllability

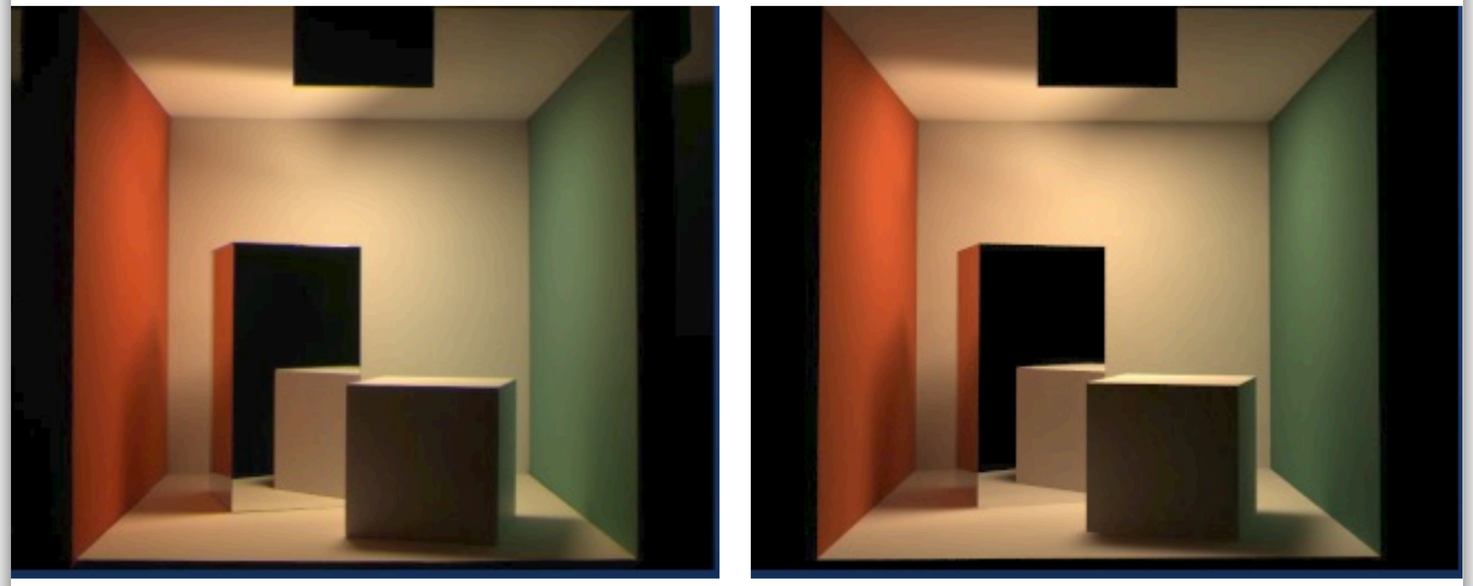


What is NeRF?

NeRF (neural radiance fields) is a method for novel view synthesis

“Forward” rendering

Light Transport and Image Synthesis



Photograph (CCD) vs. computer rendering

Inverse rendering

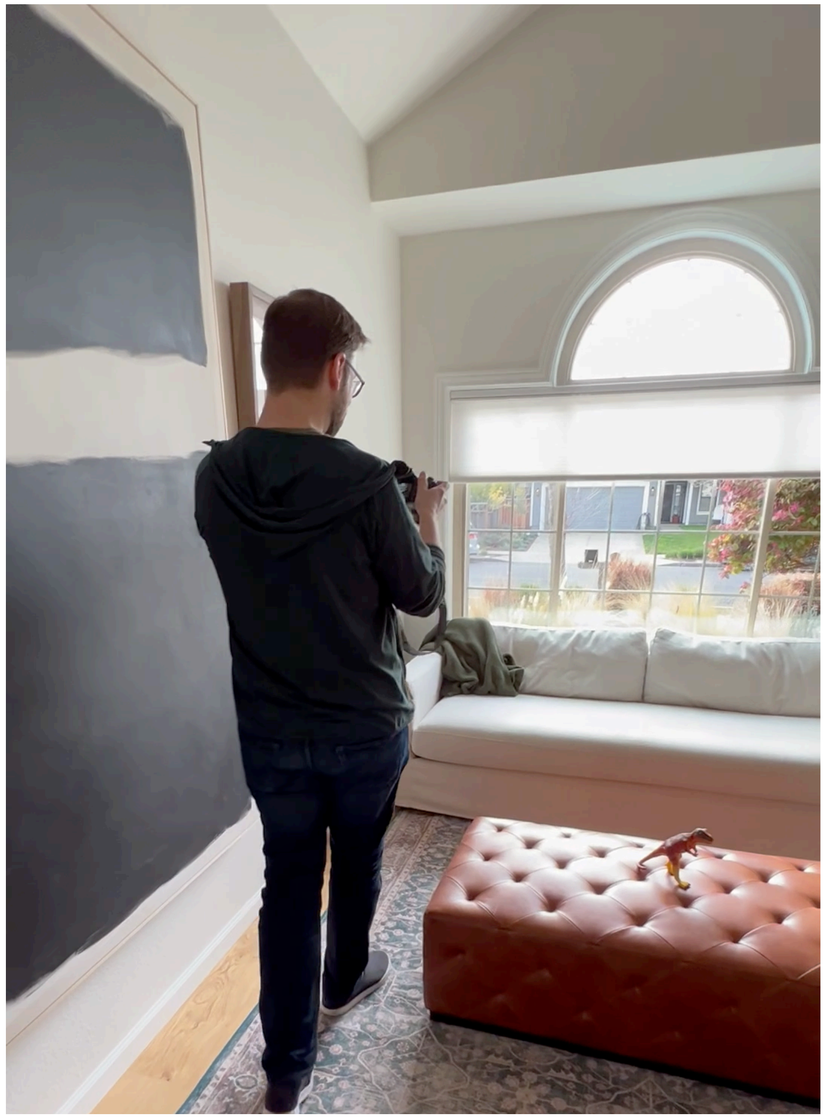


Capture images



Reconstruct the scene
and render new views

Inverse rendering





Subproblem: view synthesis

- With path tracing, we would be able to move the camera but also edit all other attributes of the scene (geometry, materials, lighting, etc.)…
- We (mostly) focus on an easier problem: what if we **only** want to move the camera?



NeRF:
Representing Scenes as Neural Radiance Fields for View Synthesis

ECCV 2020



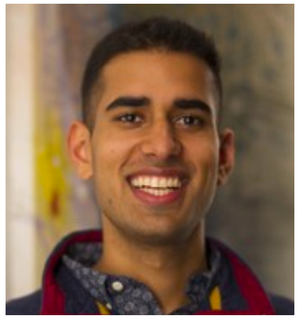
Ben Mildenhall*



UC Berkeley

Google

Pratul Srinivasan*



UC Berkeley

Google

Matt Tancik*



UC Berkeley

Jon Barron



Google Research

Google

Ravi Ramamoorthi



UC San Diego

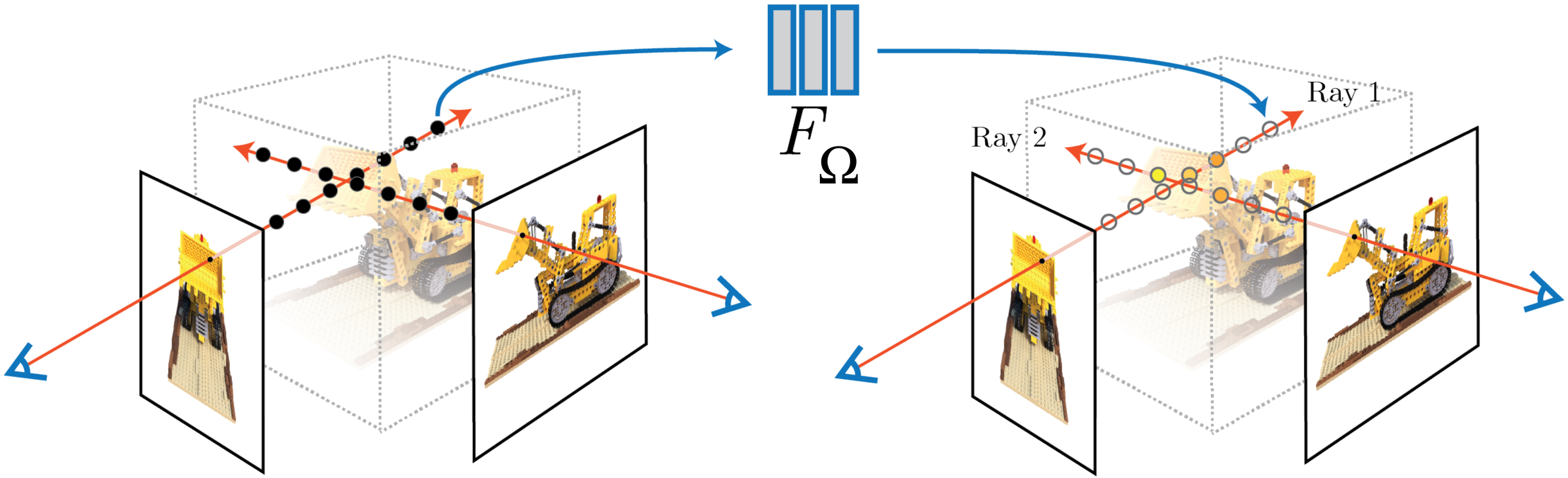
UC San Diego

Ren Ng

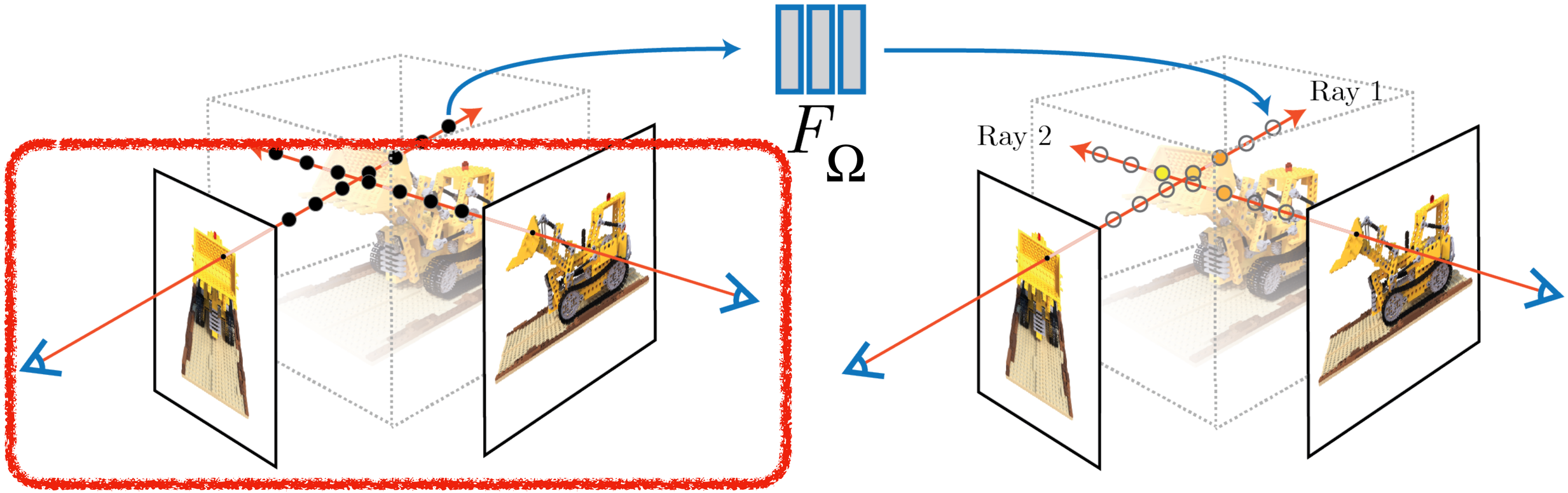


UC Berkeley

NeRF recap

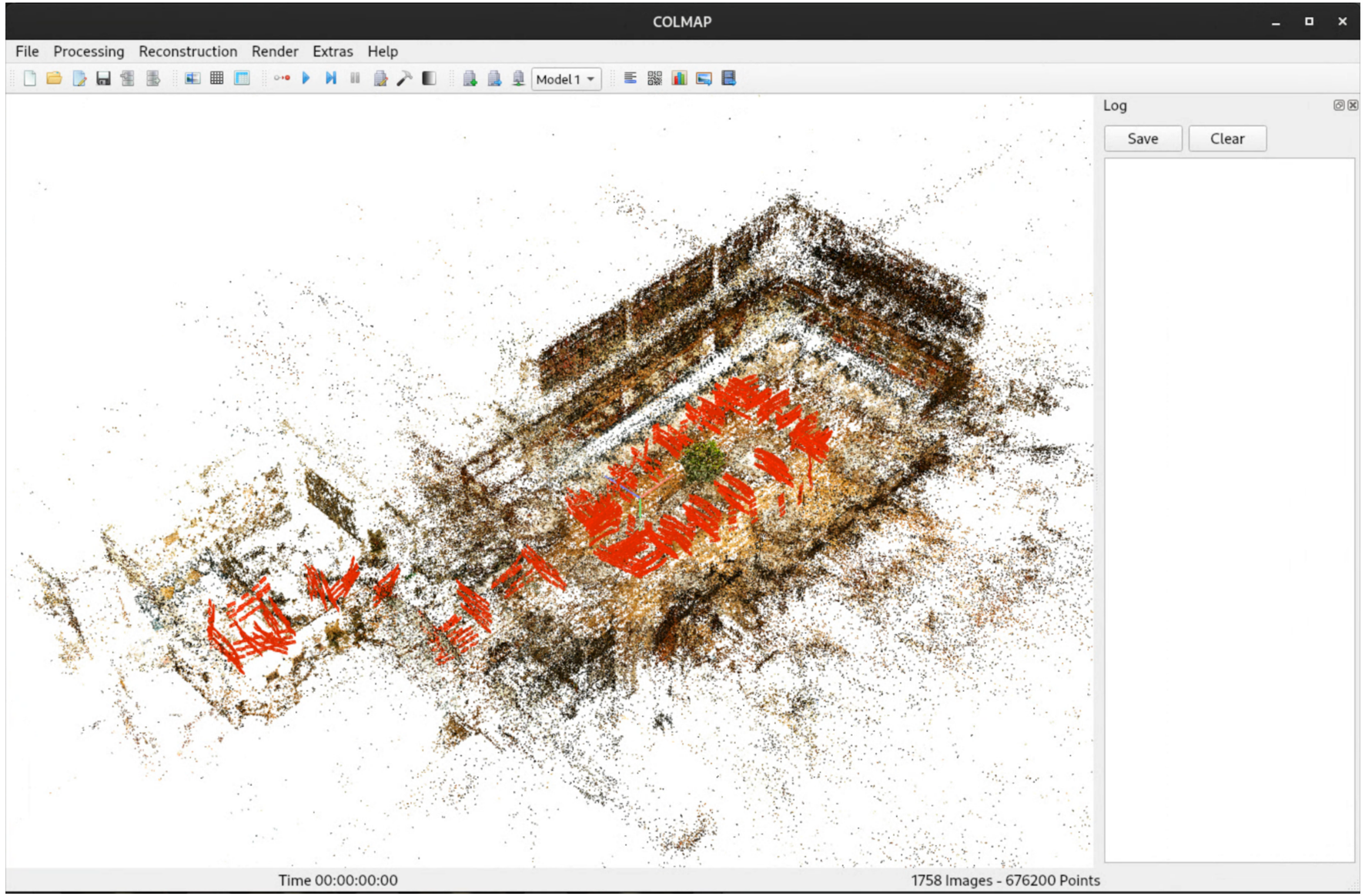


NeRF recap



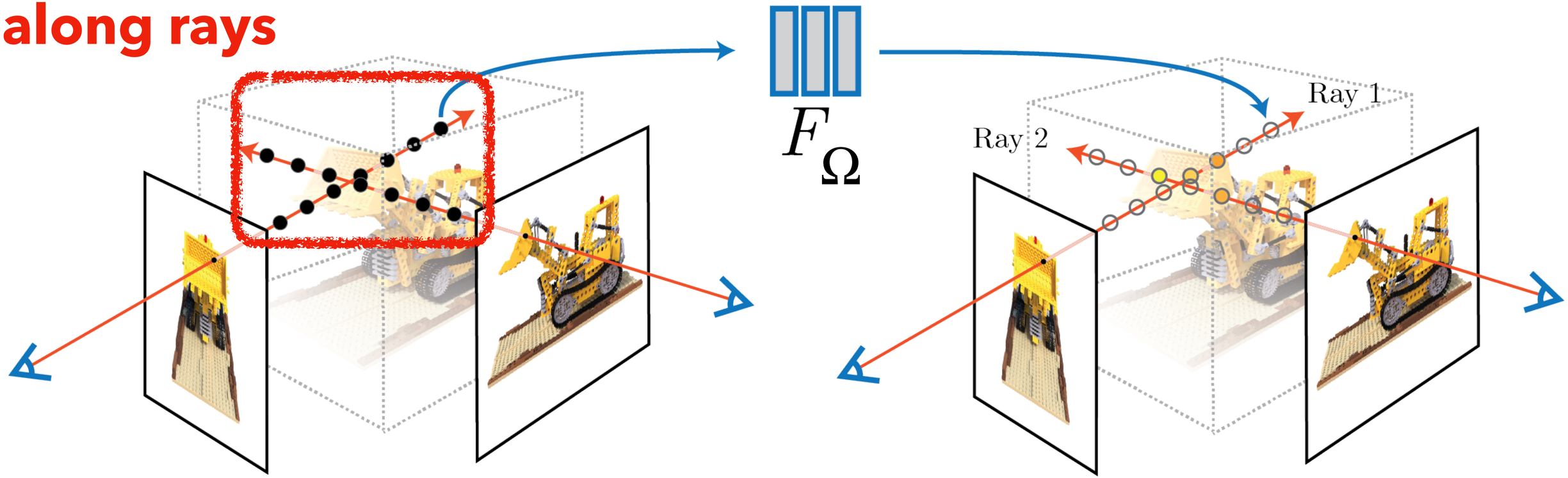
Start with input images and poses

Computing camera poses from images is “solved”***



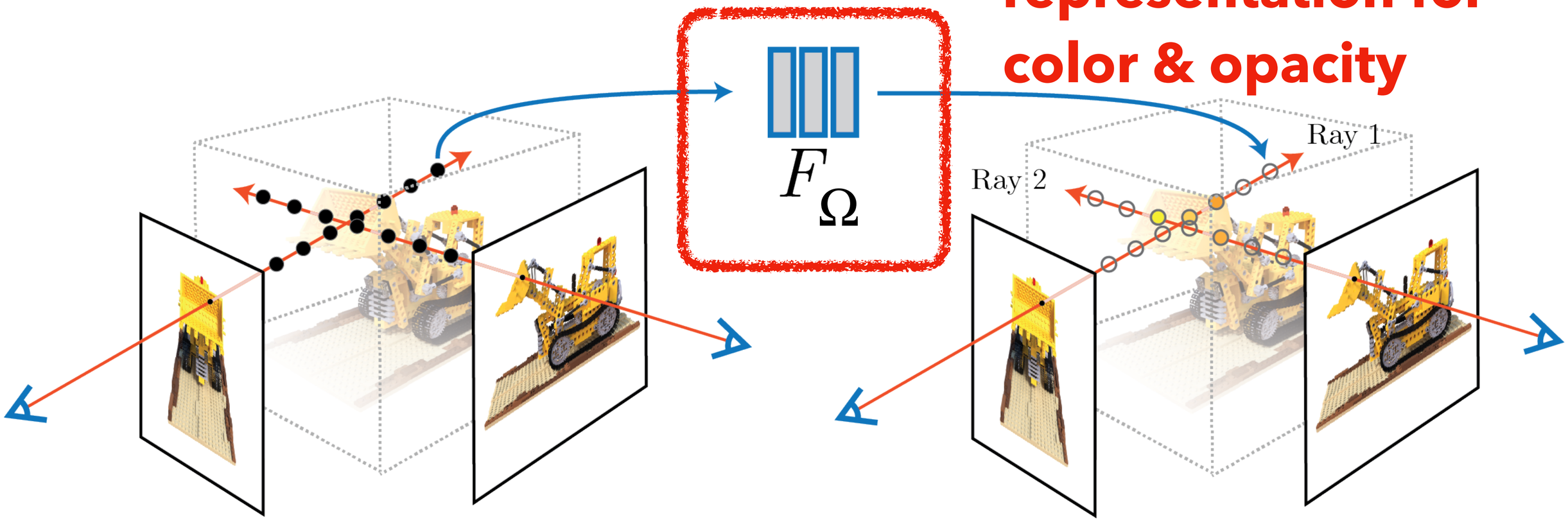
NeRF recap

Sample 3D points along rays

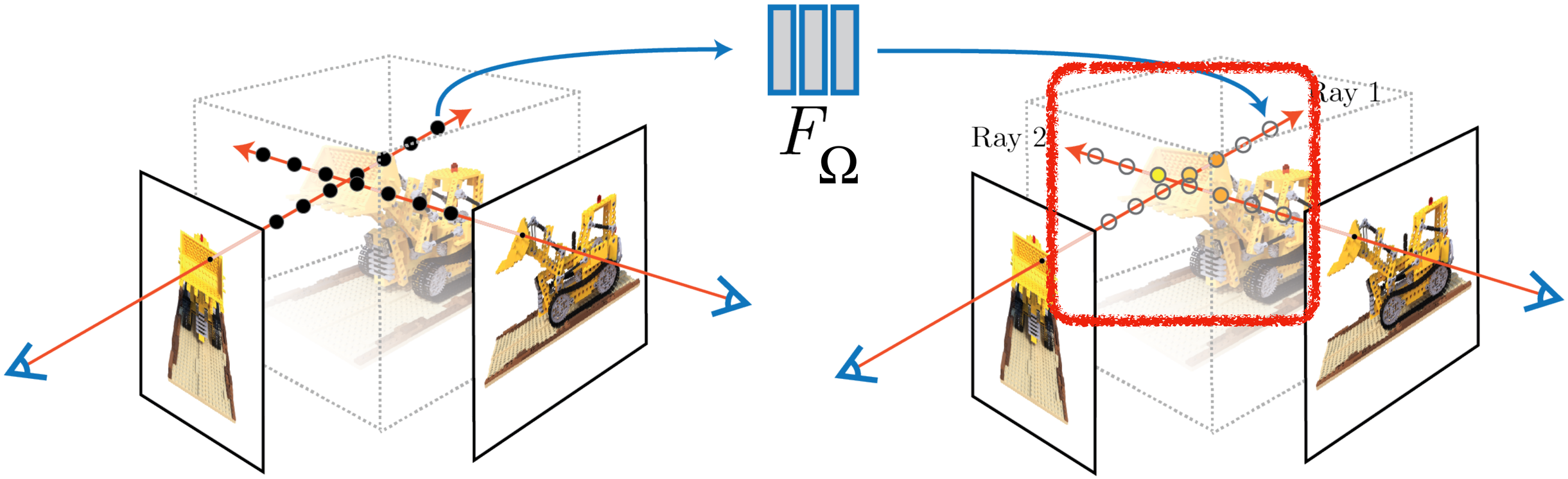


NeRF recap

Query scene representation for color & opacity

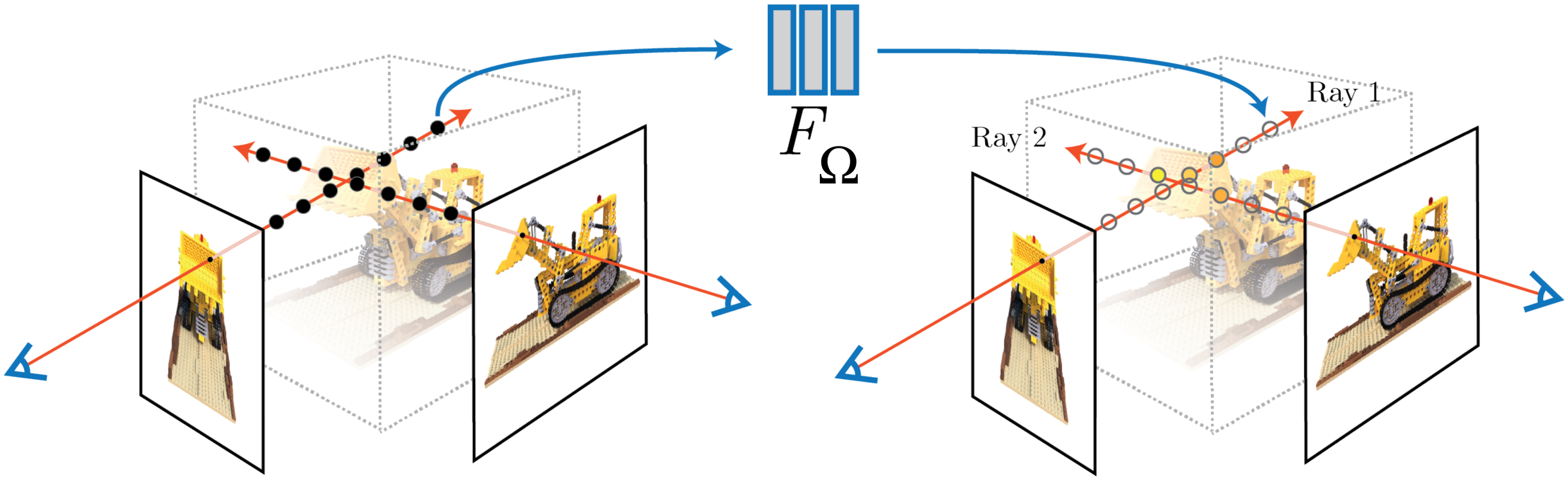


NeRF recap



Use volumetric rendering to compute ray colors

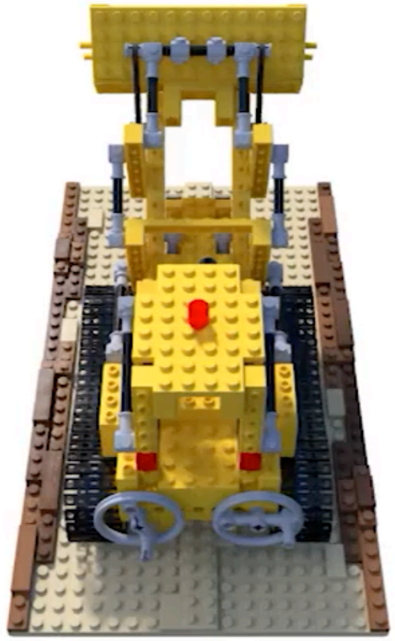
NeRF recap



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

Optimize rendering loss (vs input image pixel colors)





















Dive into…

- Volume rendering
- Scene representation

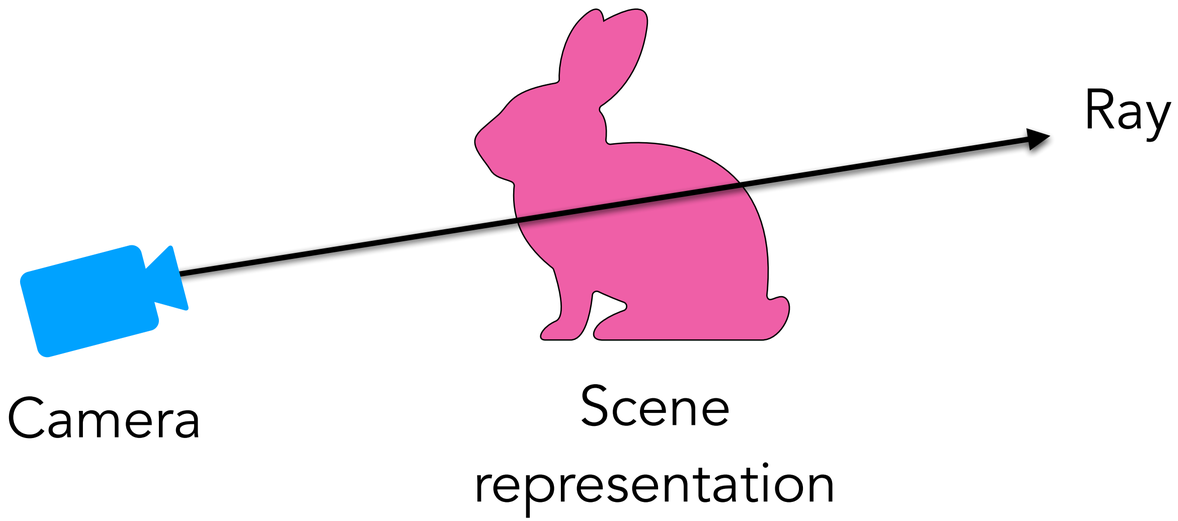
Volume Rendering

Neural **Volumetric** Rendering

continuous, differentiable
rendering model without
concrete ray/surface intersections

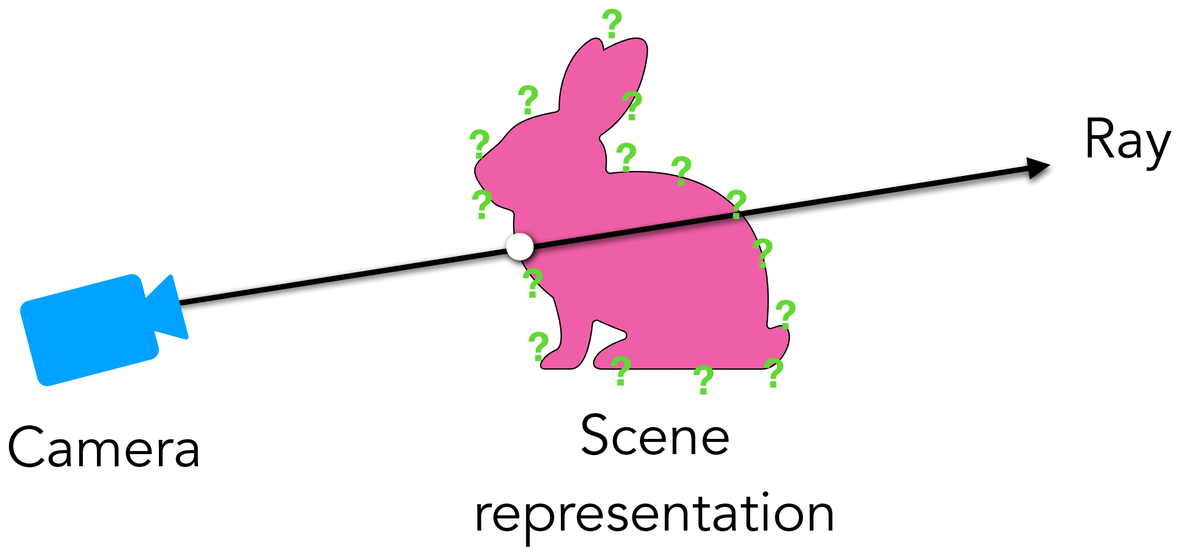


Surface vs. volume rendering



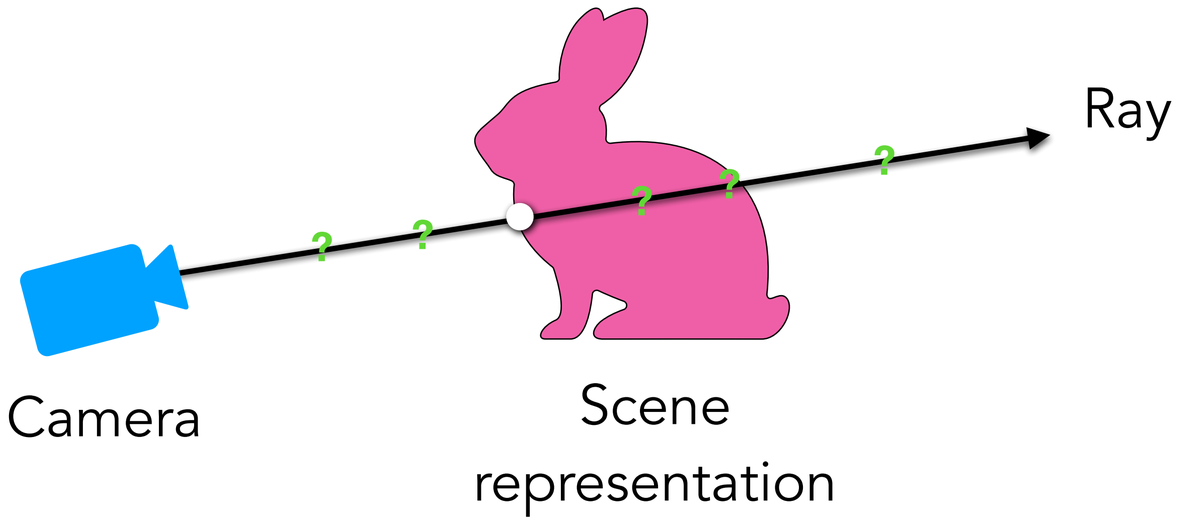
Want to know how ray interacts with scene

Surface vs. volume rendering



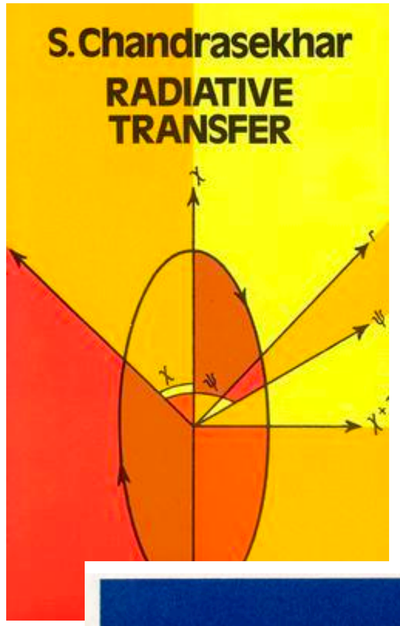
Surface rendering — loop over geometry, check for ray hits

Surface vs. volume rendering



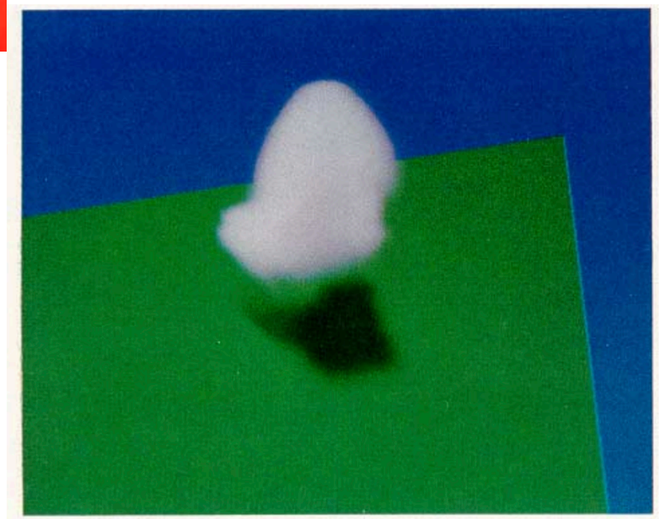
Volume rendering — loop over ray points, query geometry

History of volume rendering



Early computer graphics

- Theory of volume rendering co-opted from physics in the 1980s: absorption, emission, out-scattering/in-scattering

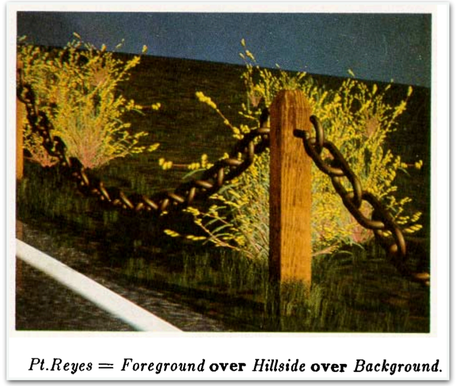


Ray tracing simulated cumulus cloud [Kajiya]

Chandrasekhar 1950, *Radiative Transfer*
Kajiya 1984, *Ray Tracing Volume Densities*

Alpha compositing

- Theory of volume rendering co-opted from physics in the 1980s: absorption, emission, out-scattering/in-scattering

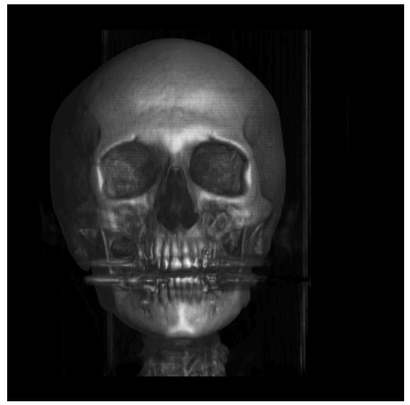


Alpha compositing [Porter and Duff]

- Alpha rendering developed for digital compositing in VFX movie production

Porter and Duff 1984, *Compositing Digital Images*

Volume rendering for visualization



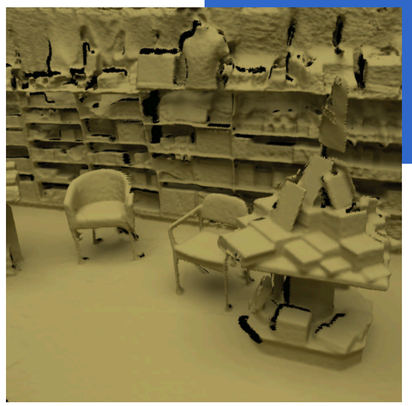
Medical data visualisation [Levoy]

- Theory of volume rendering co-opted from physics in the 1980s: absorption, emission, out-scattering/in-scattering
- Alpha rendering developed for digital compositing in VFX movie production
- Volume rendering applied to visualise 3D medical scan data in 1990s

Chandrasekhar 1950, *Radiative Transfer*
Kajiya 1984, *Ray Tracing Volume Densities*
Porter and Duff 1984, *Compositing Digital Images*
Levoy 1988, *Display of Surfaces from Volume Data*
Max 1995, *Optical Models for Direct Volume Rendering*

Volume rendering for surfaces







Geometry and materials can be stored per-voxel and used with standard surface rendering methods

- Sparse voxel octrees
- Voxel hashing
- Anisotropic radiative transfer

Volume rendering derivations

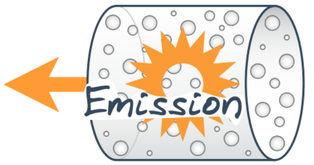


Absorption





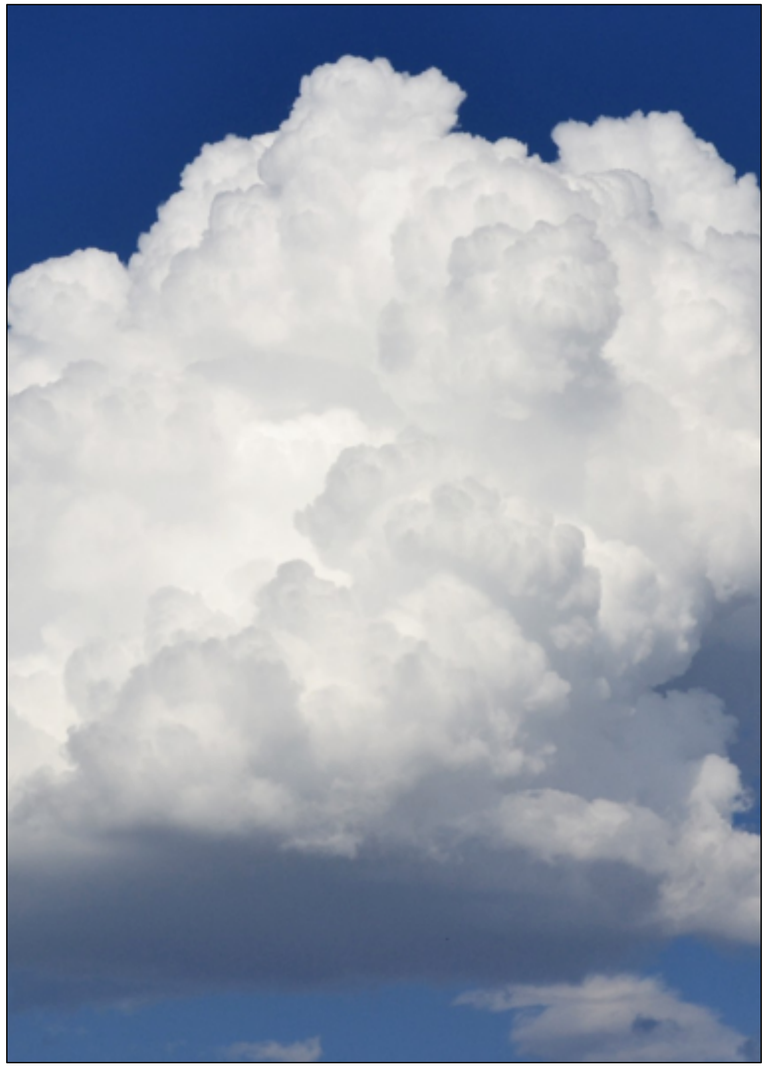
Scattering



Emission



http://commons.wikimedia.org





http://wikipedia.org

Slide credit: Novak et al 2018, *Monte Carlo methods for physically based volume rendering*

Simplify

Absorption | **Scattering** | **Emission**



http://commons.wikimedia.org





http://wikipedia.org

Slide credit: Novak et al 2018, *Monte Carlo methods for physically based volume rendering*

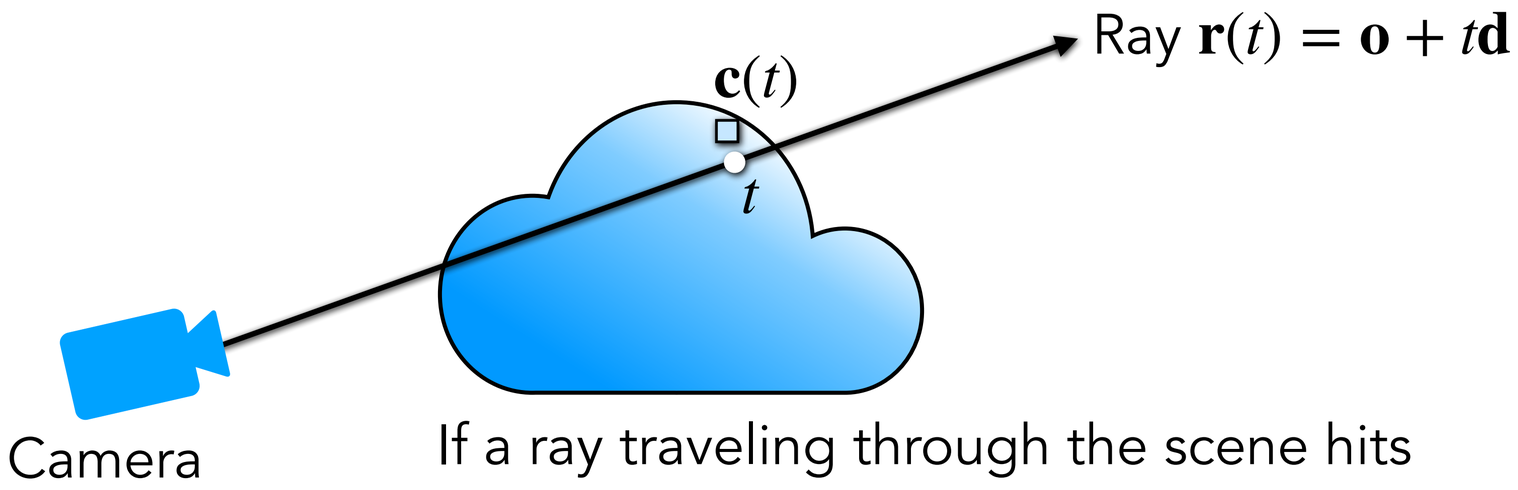
Volumetric formulation for NeRF



Scene is a cloud of tiny colored particles

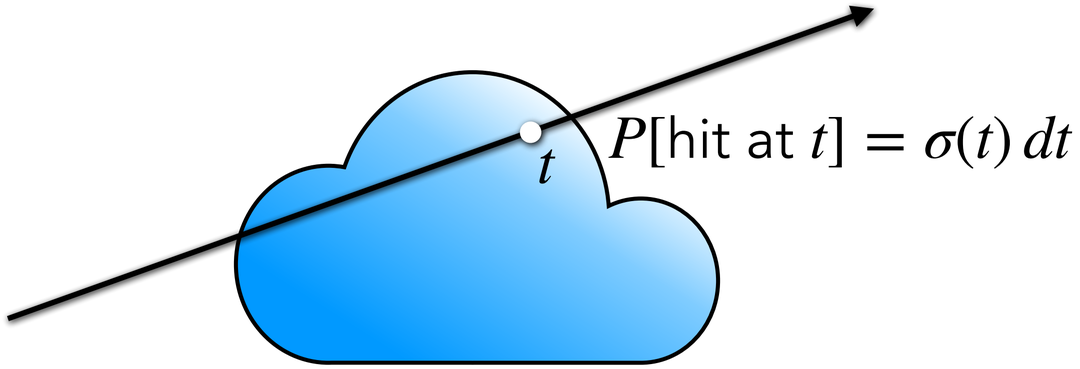
Max and Chen 2010, *Local and Global Illumination in the Volume Rendering Integral*

Volumetric formulation for NeRF



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

What does it mean for a ray to “hit” the volume?

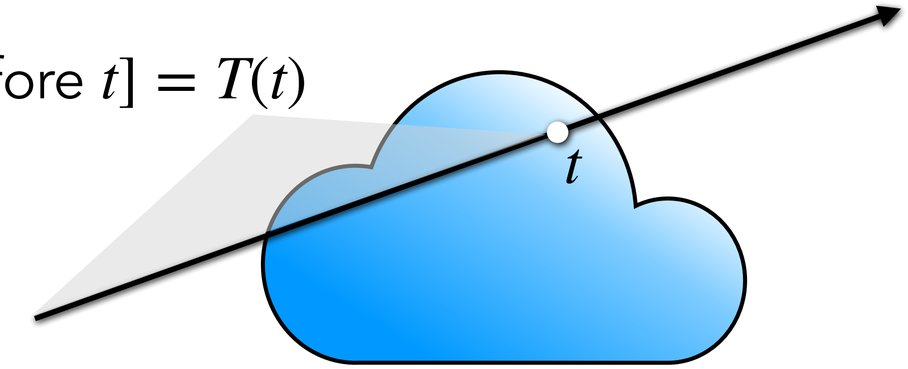


This notion is *probabilistic:* chance that ray hits a particle in a small interval around t is $\sigma(t)\,dt$.

σ is called the “volume density”

Probabilistic interpretation

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

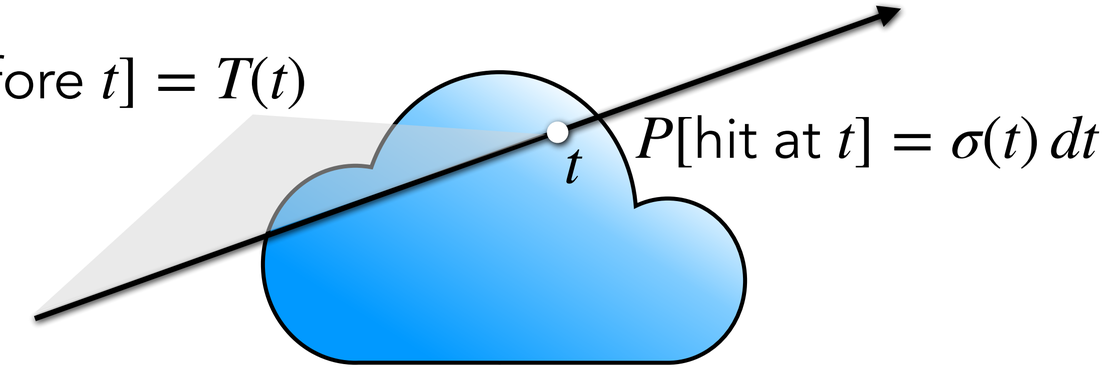


To determine if t is the *first* hit along the ray, need to know $T(t)$: the probability that the ray makes it through the volume up to t.

$T(t)$ is called "transmittance"

Probabilistic interpretation

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

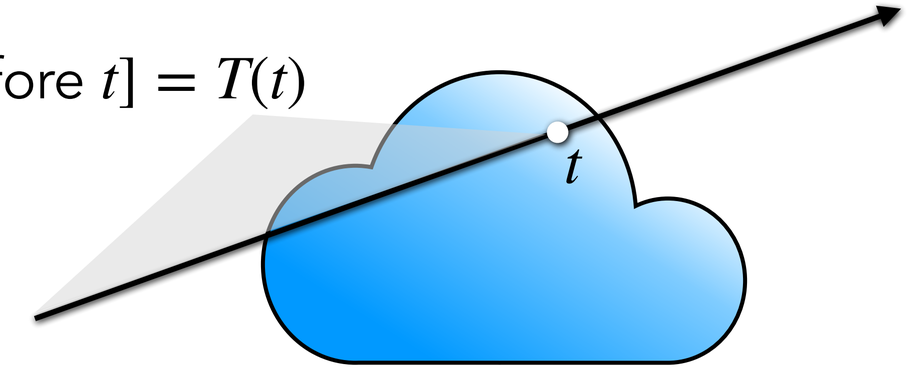


The product of these probabilities tells us how much you see the particles at t:

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

Calculating T given σ

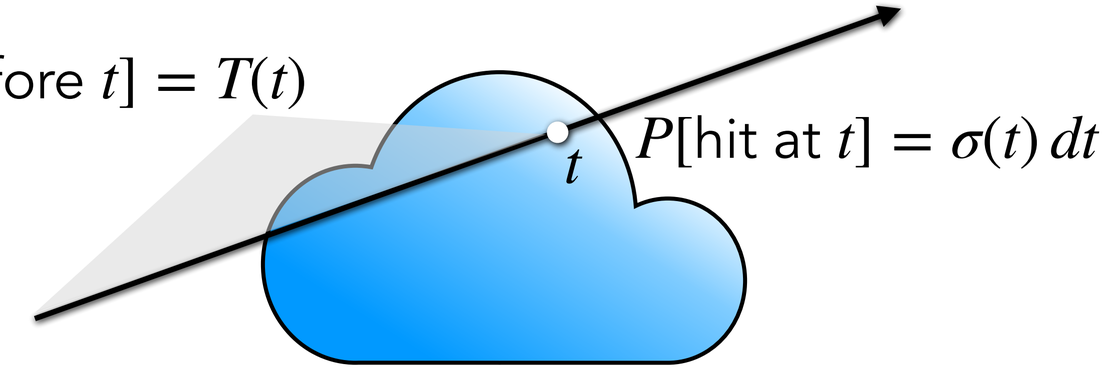
$P[\text{no hits before } t] = T(t)$



If σ is known, T can be computed… How?

Calculating T given σ

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



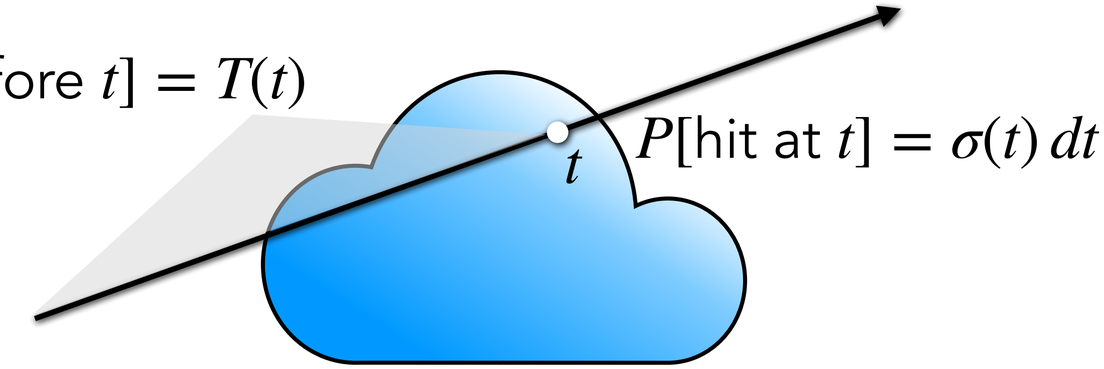
$P[\text{hit at } t] = \sigma(t)\,dt$

σ and T are related by the probabilistic fact that

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

Calculating transmittance T

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

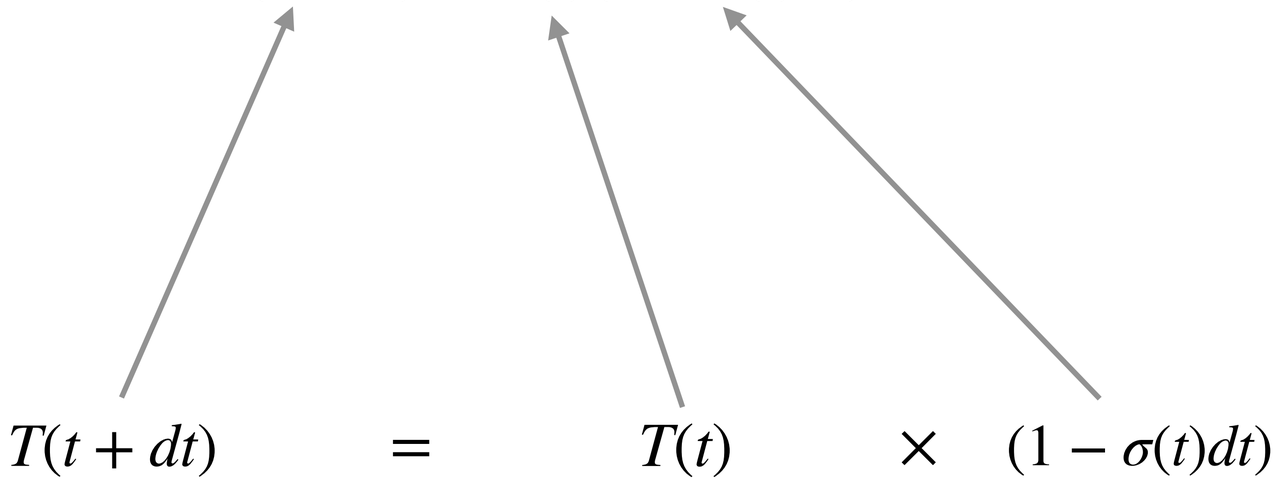


σ and T are related by the probabilistic fact that

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

Calculating transmittance T

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



Solve for T

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

Solve for T

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

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

Solve for T

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

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

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

Solve for T

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

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

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

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

Solve for T

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

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

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

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

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

Density is score function!

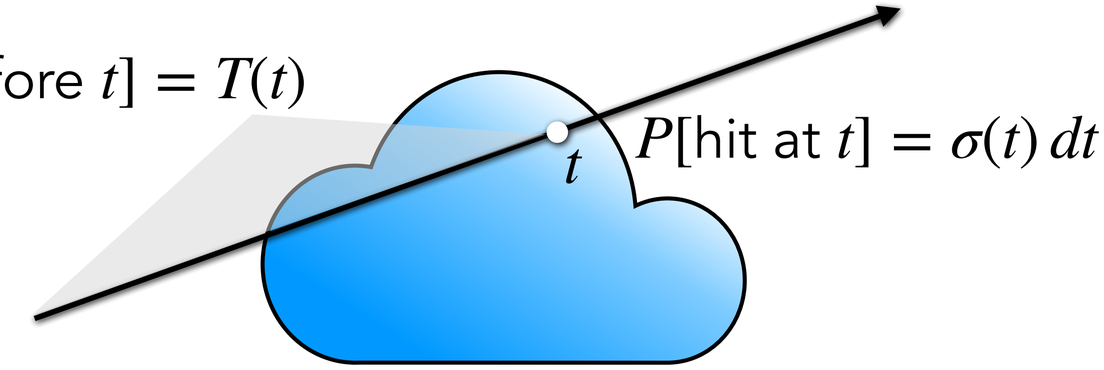
An interesting observation is that density is actually the negative score function of T:

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

This is why it doesn't need normalization!

PDF for ray termination

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



Finally, we can write the probability that a ray terminates at t as a function of only sigma

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

$$= T(t)\sigma(t)dt$$

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

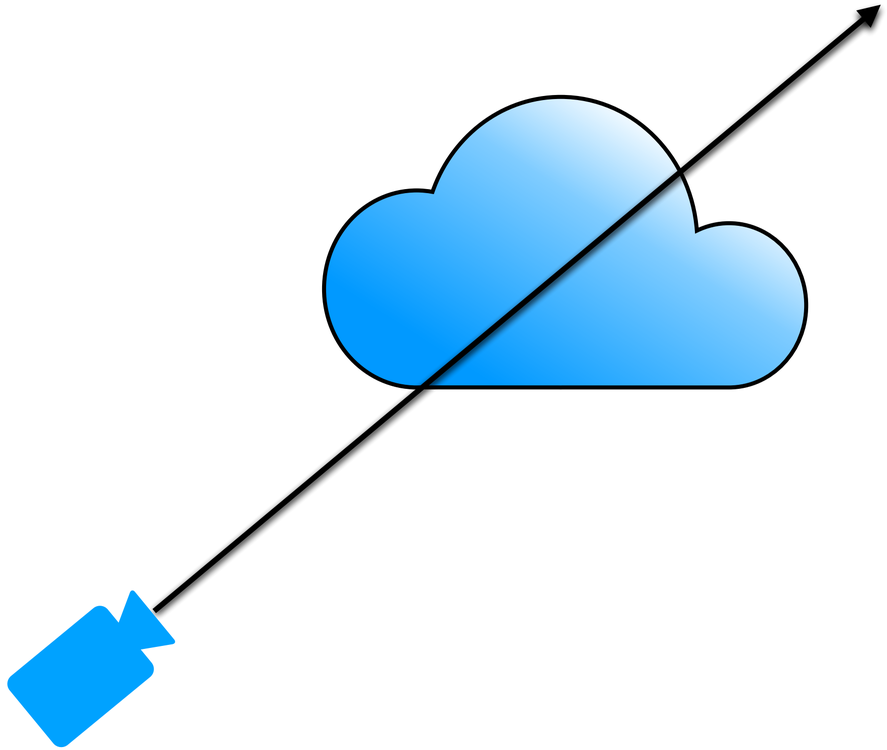
Expected value of color along ray

This means the expected color returned by the ray will be

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

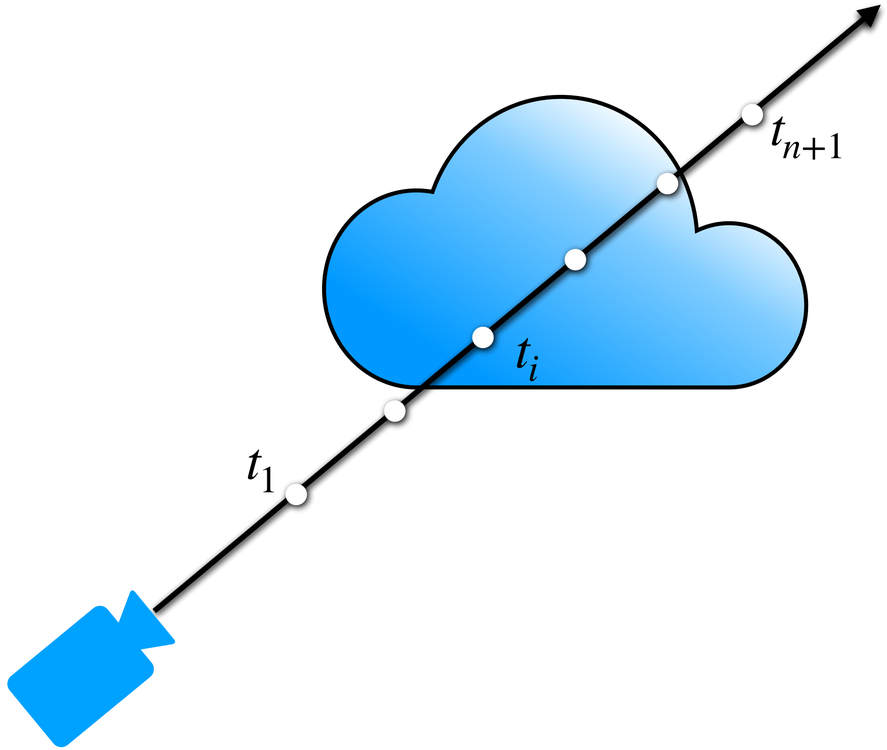
Note the nested integral!

Approximating the nested integral



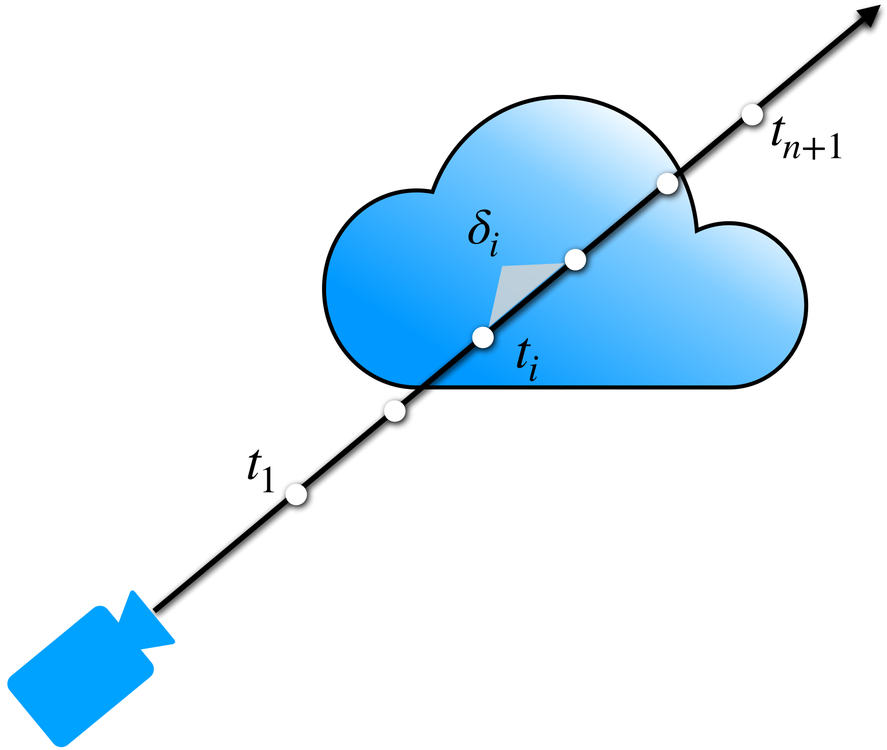
We use quadrature to approximate the nested integral,

Approximating the nested integral



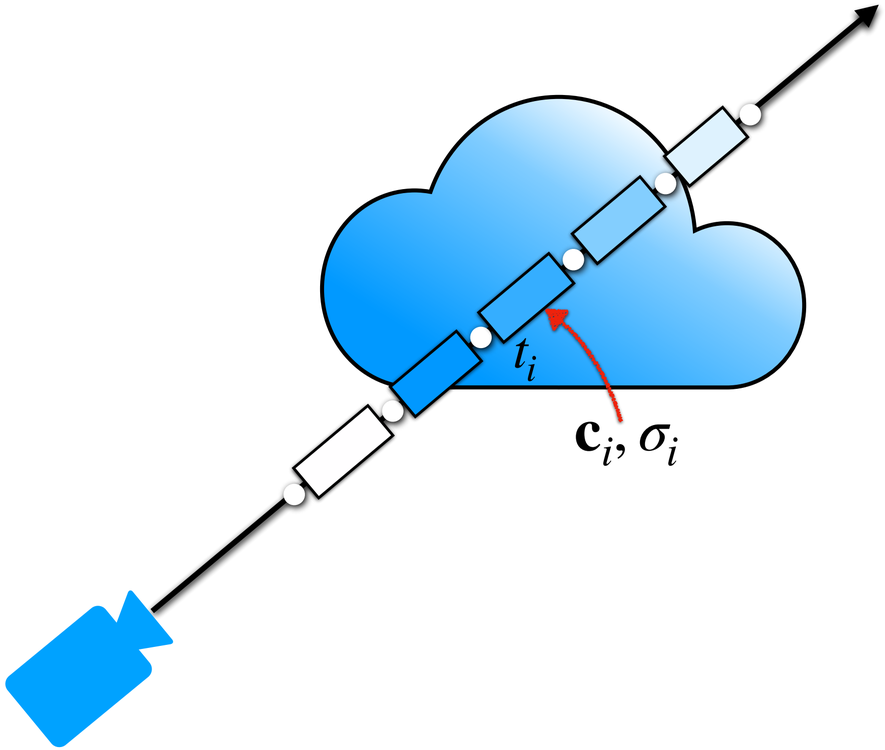
We use quadrature to approximate the nested integral,
splitting the ray up into n segments with endpoints $\{t_1, t_2, \ldots, t_{n+1}\}$

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 approximate volume density and color as constant within each interval

Deriving quadrature estimate

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

This allows us to break the outer integral

Deriving quadrature estimate

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

This allows us to break the outer integral
into a sum of analytically tractable integrals

Deriving quadrature estimate

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

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

Deriving quadrature estimate

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

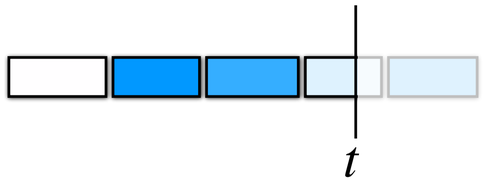
Caveat: piecewise constant density and color
do not imply constant transmittance!

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

Evaluating T for piecewise constant density

$$\text{For } t \in [t_i, t_{i+1}],\ T(t) = \exp\left(-\int_{t_1}^{t_i} \sigma_i \, ds\right) \exp\left(-\int_{t_i}^{t} \sigma_i \, ds\right)$$

We need to evaluate at continuous t values
that can lie *partway through* an interval

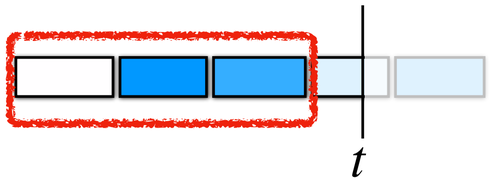


Evaluating T for piecewise constant density

$$\text{For } t \in [t_i, t_{i+1}],\ T(t) = \exp\left(-\int_{t_1}^{t_i} \sigma_i \, ds\right) \exp\left(-\int_{t_i}^{t} \sigma_i \, ds\right)$$

$$\exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right) = T_i$$

"How much light is blocked by all previous segments?"

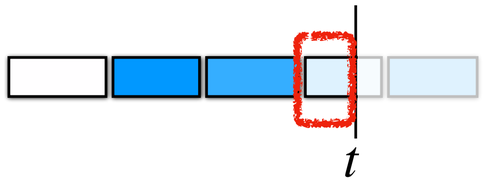


Evaluating T for piecewise constant density

$$\text{For } t \in [t_i, t_{i+1}],\ T(t) = \exp\left(-\int_{t_1}^{t_i} \sigma_i \, ds\right) \exp\left(-\int_{t_i}^{t} \sigma_i \, ds\right)$$

"How much light is blocked partway through the current segment?"

$$\exp\left(-\sigma_i(t - t_i)\right)$$



Deriving quadrature estimate

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

Deriving quadrature estimate

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

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

Deriving quadrature estimate

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

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

Integrate

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

Deriving quadrature estimate

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

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

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

Cancel σ_i

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

Connection to alpha compositing

$$= \sum_{i=1}^{n} T_i \mathbf{c}_i \underbrace{(1 - \exp(-\sigma_i \delta_i))}_{\text{segment opacity } \alpha_i}$$

Connection to alpha compositing

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

$$\text{opacity } \alpha_i$$

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

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

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

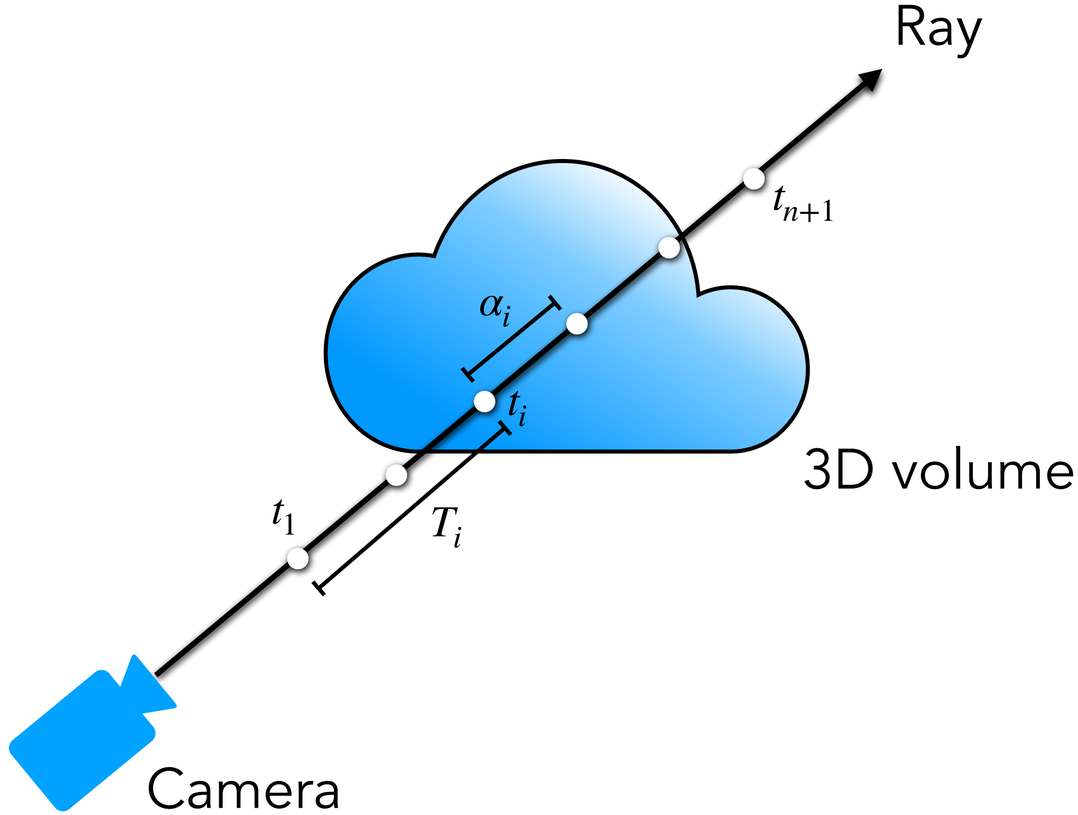
weights

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*:

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



Volume rendering is trivially differentiable

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

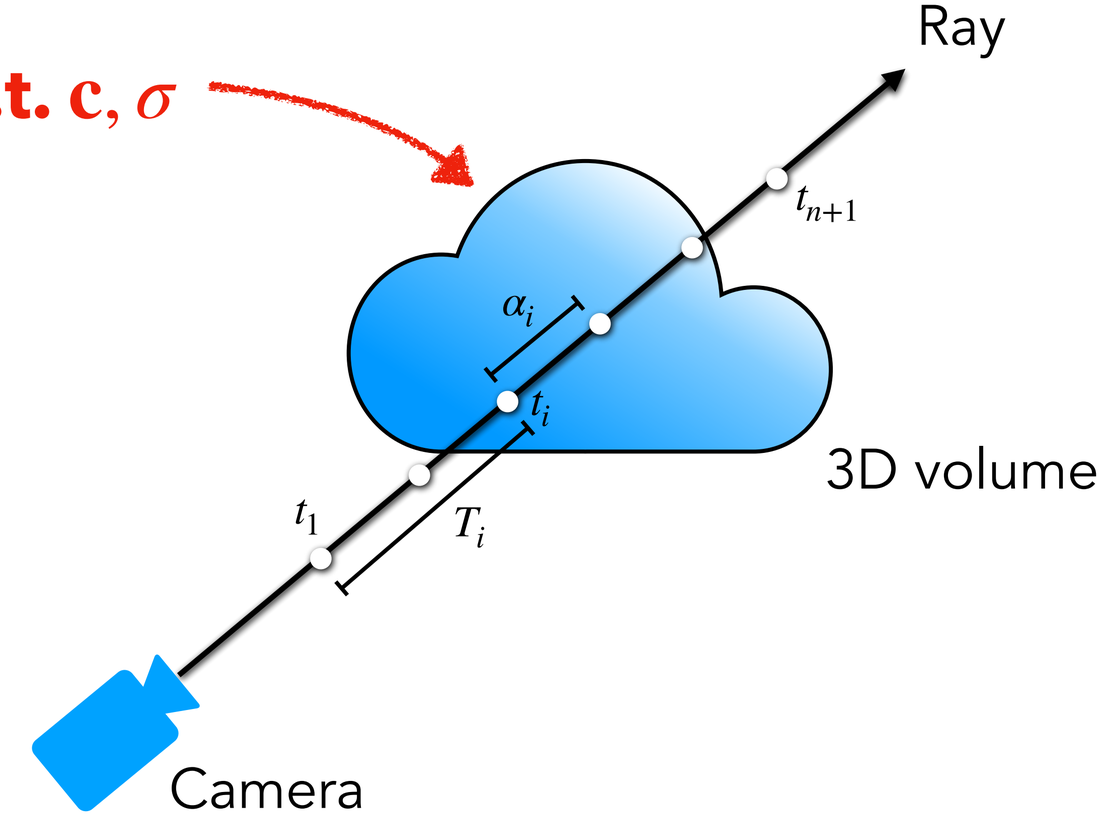
differentiable w.r.t. $\mathbf{c}$, σ

How much light is blocked earlier along ray:

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

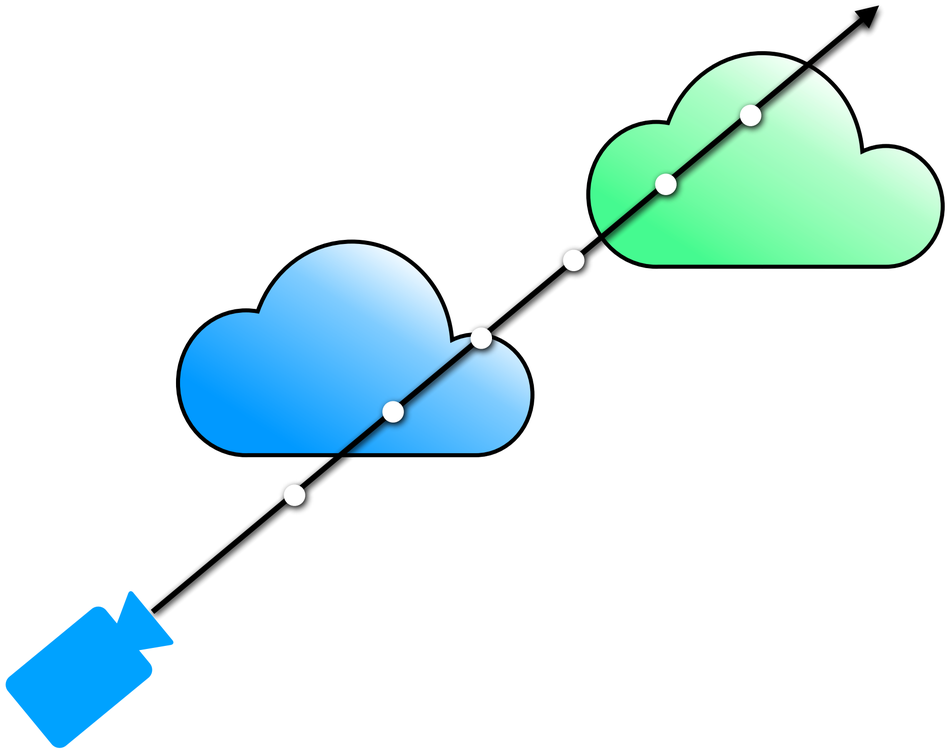
How much light is contributed by ray segment *i*:

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

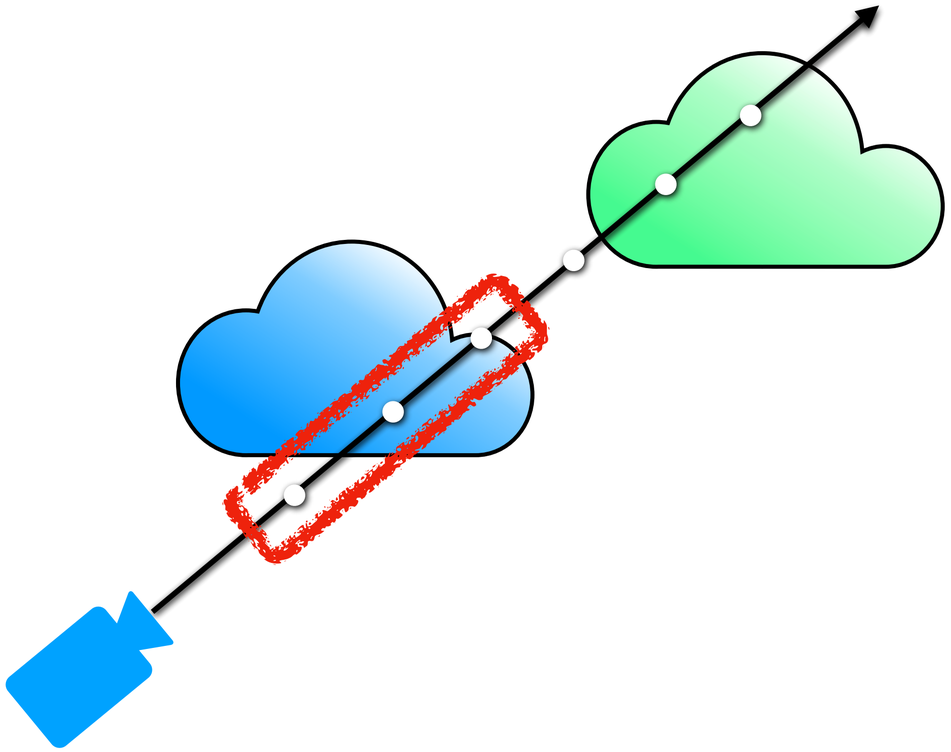


Further points on volume rendering

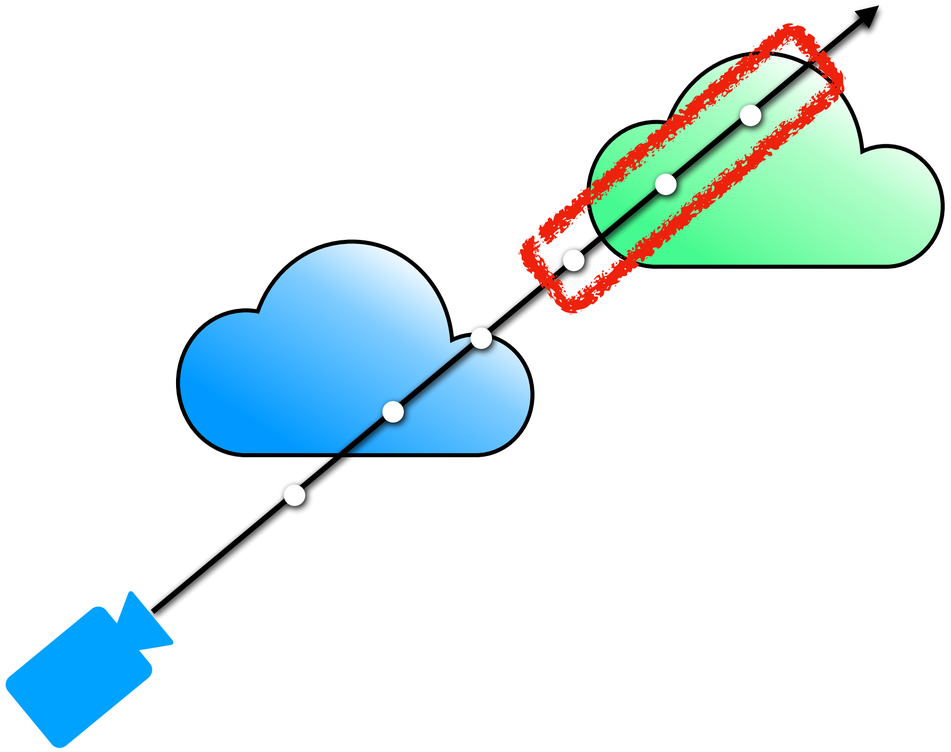
Alpha mattes and compositing



Alpha mattes and compositing



Alpha mattes and compositing



Alpha mattes and compositing







Mildenhall*, Srinivasan*, Tancik* et al 2020, *NeRF*

Poole et al 2022, *DreamFusion*

Tang et al 2022, *Compressible-composable NeRF via Rank-residual Decomposition*

Rendering weight PDF is important

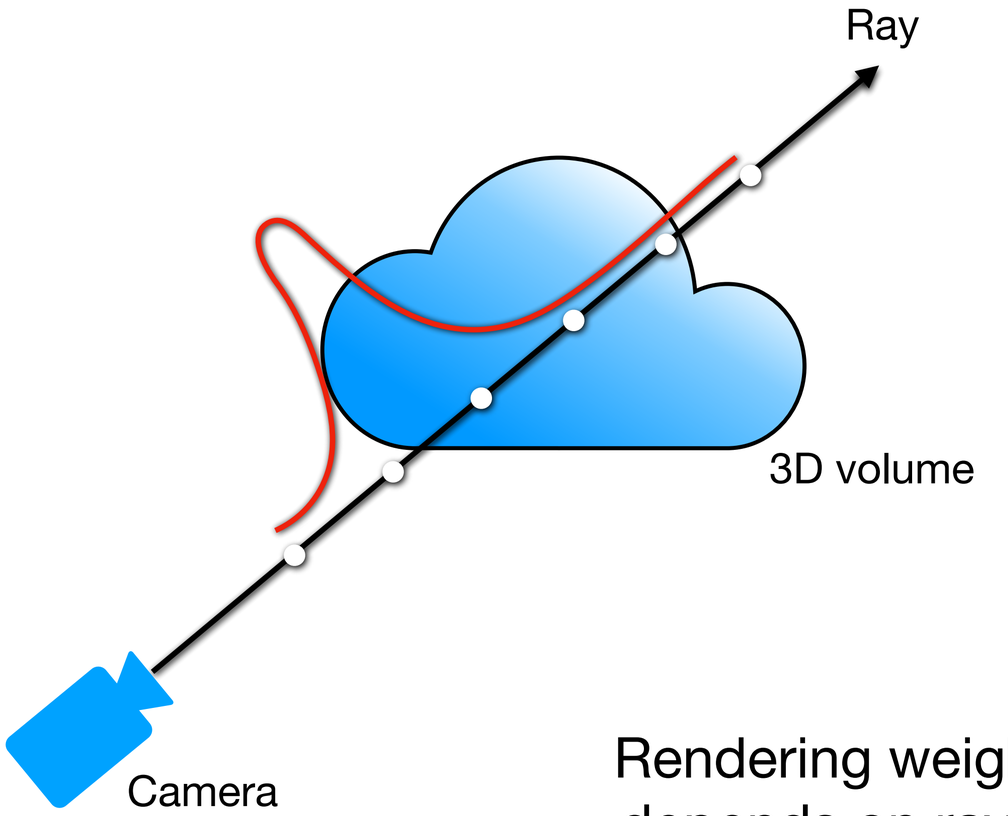
Remember, expected color is equal to

$$\int T(t)\sigma(t)\mathbf{c}(t)\,dt \approx \sum_i T_i \alpha_i \mathbf{c}_i$$

$T(t)\sigma(t)$ and $T_i\alpha_i$ are “rendering weights” — probability distribution along the ray (continuous and discrete, respectively)

Visual intuition — rendering weights not just 3D function

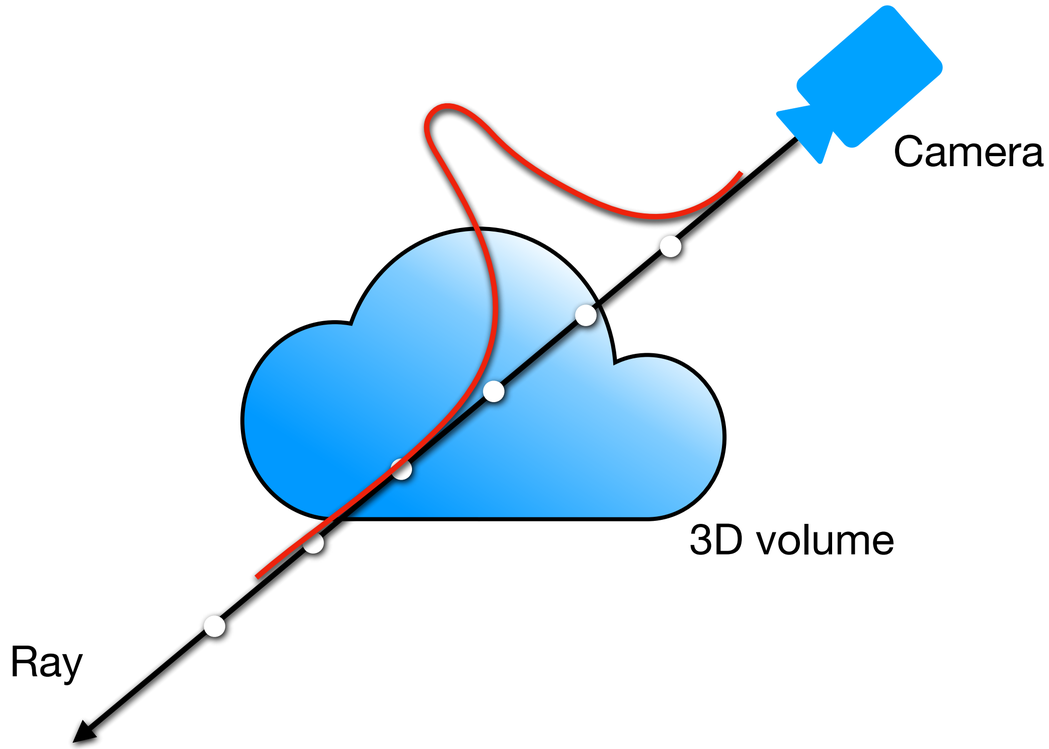
$$C \approx \sum_{i=1}^{N} T_i \alpha_i c_i$$



Rendering weights are not a 3D function — depends on ray, because of tranmisttance!

Visual intuition — rendering weights not just 3D function

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



Rendering weights are not a 3D function — depends on ray, because of tranmisttance!

Rendering weight PDF is important — depth

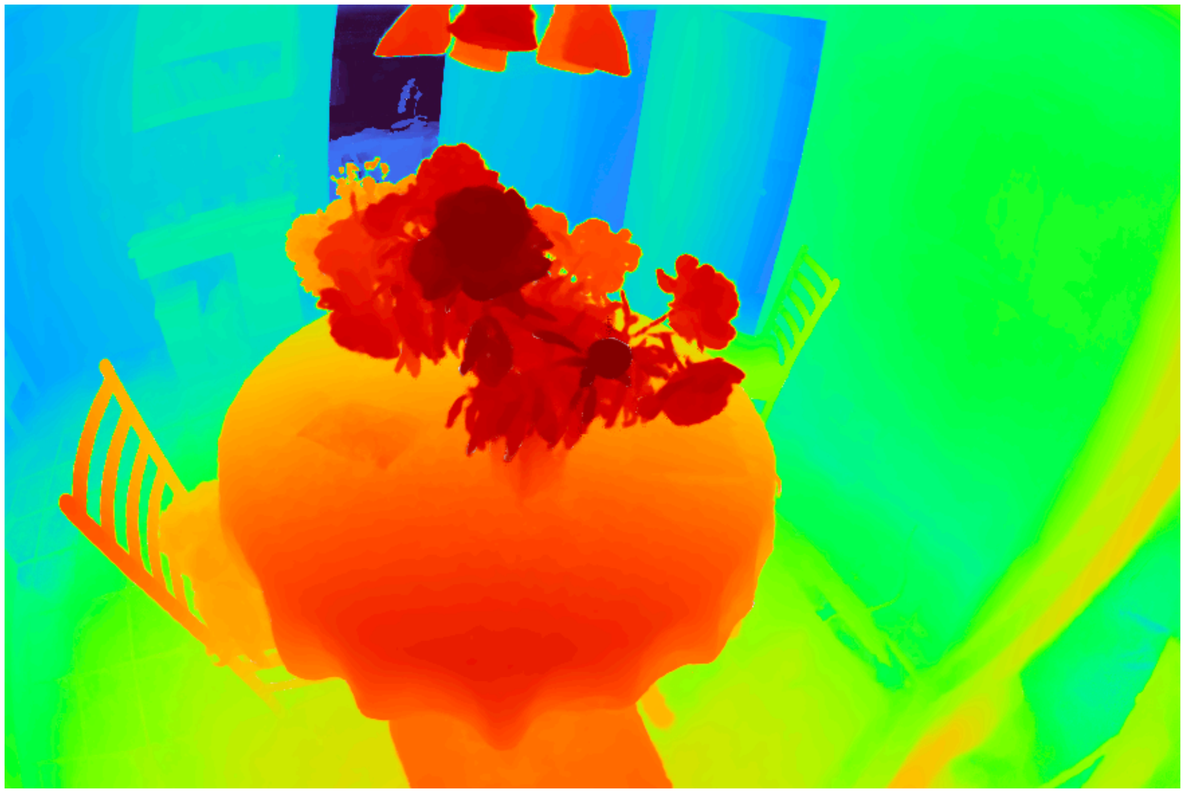
We can use this distribution to compute expectations for other quantities,
e.g. “expected depth”:

$$\bar{t} = \sum_i T_i \alpha_i t_i$$

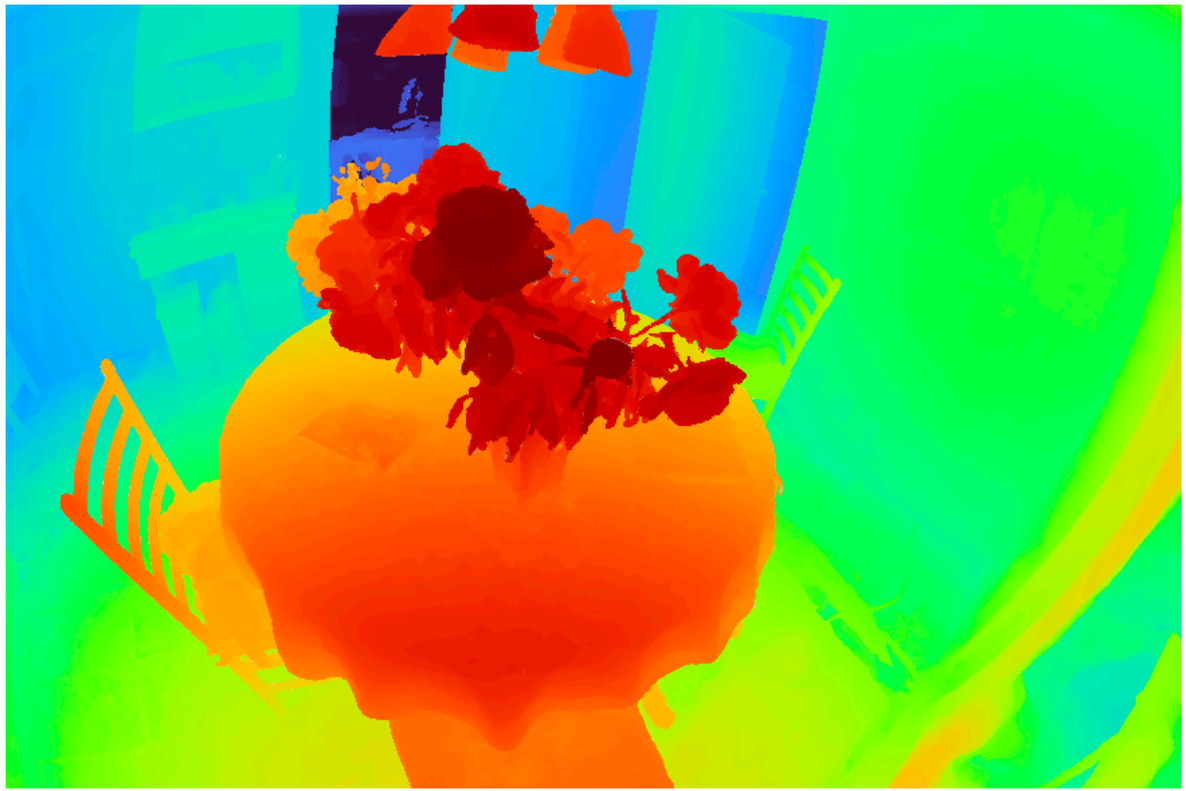
This is often how people visualise NeRF depth maps.

Alternatively, other statistics like mode or median can be used.

Rendering weight PDF is important — depth

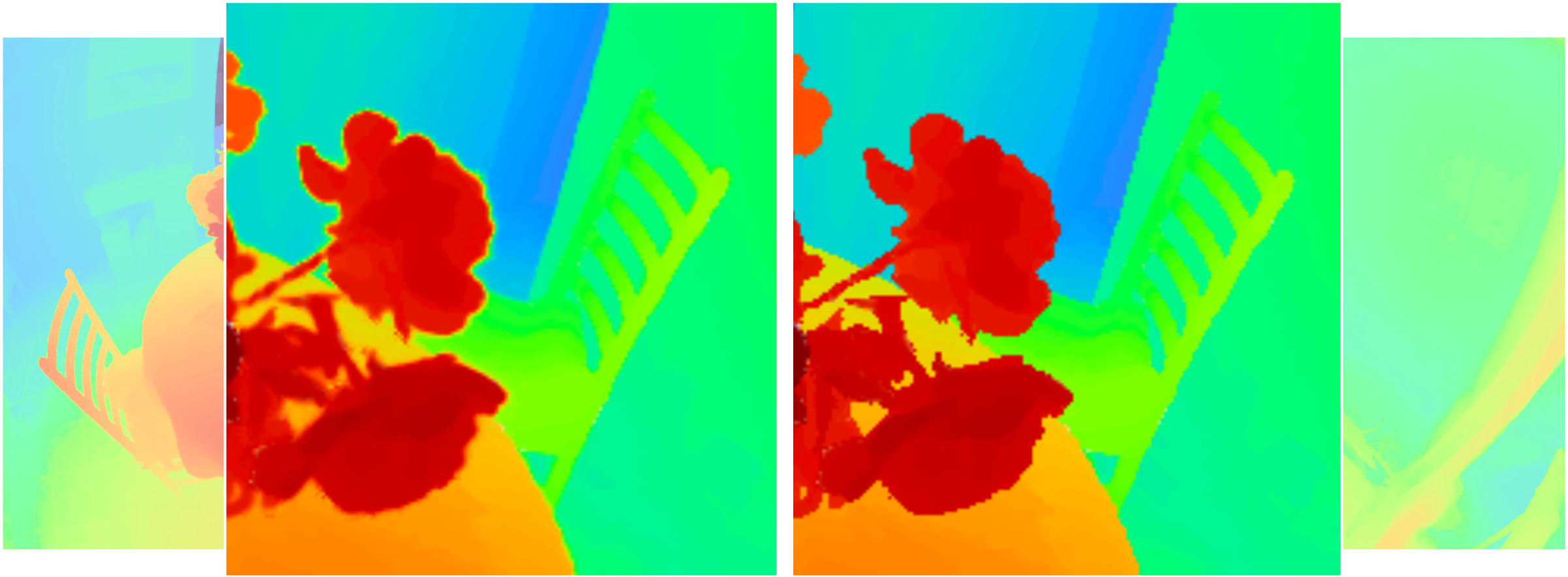


Mean depth



Median depth

Rendering weight PDF is important — depth



Mean depth

Median depth

Volume rendering other quantities

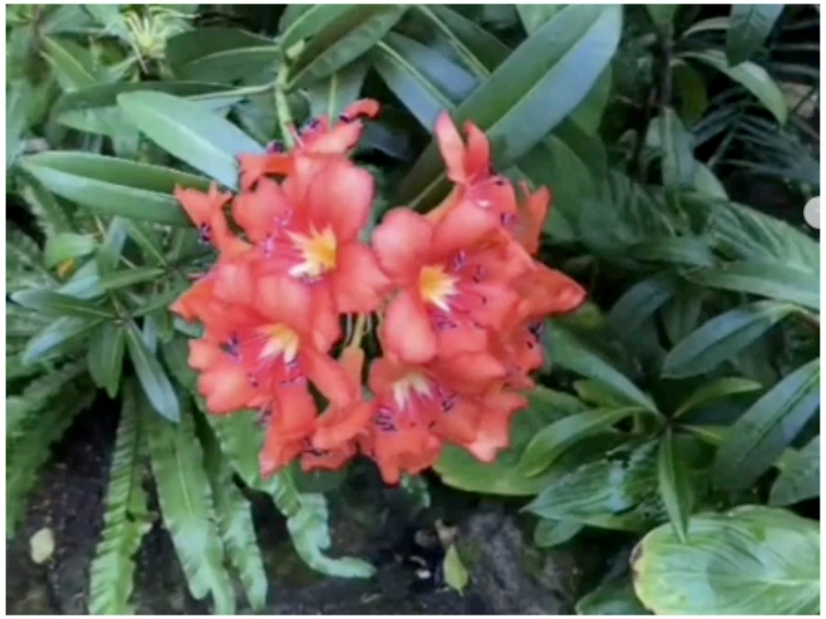
This idea can be used for any quantity we want to “volume render” into a 2D image.

If $\mathbf{v}$ lives in 3D space (semantic features, normal vectors, etc.)

$$\sum_i T_i \alpha_i \mathbf{v}_i$$

can be taken per-ray to produce 2D output images.

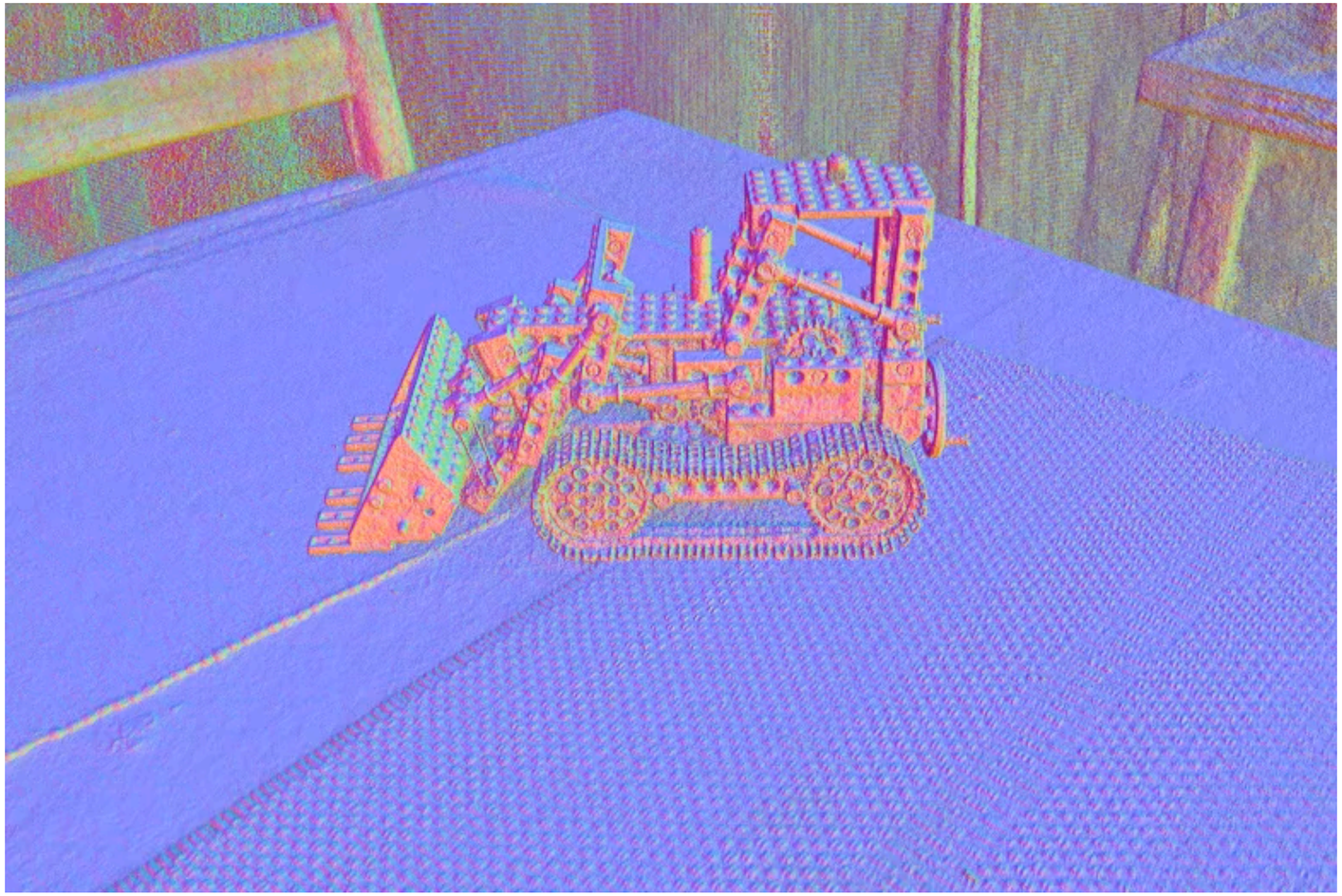
Volume rendering other quantities



Various works have used this idea to render higher-level semantic feature maps (e.g., *Feature Field Distillation* and *Neural Feature Fusion Fields*).

Kobayashi et al 2022, *Decomposing NeRF for Editing via Feature Field Distillation*

Density as geometry

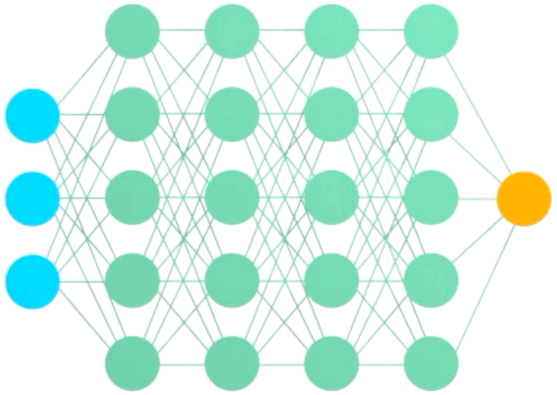


Normal vectors (from analytic gradient of density)

Scene Representation

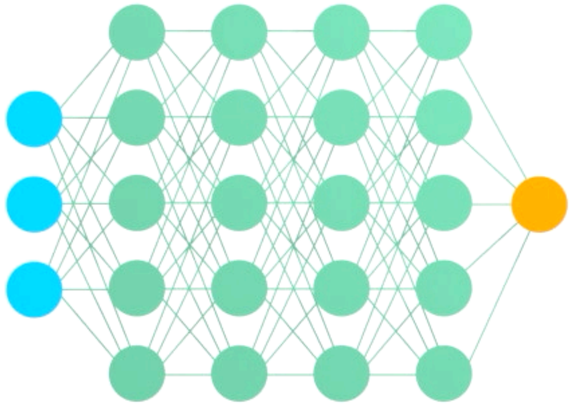
Neural Volumetric Rendering

learned scene representation



Coordinate Based Neural Network

Input Coordinate



Value at Coordinate

Multi Layer Perceptron

MLP

Neural networks as a continuous shape representation

Occupancy Networks
(Mescheder et al. 2019)

$(x, y, z) \rightarrow$ occupancy

DeepSDF
(Park et al. 2019)

$(x, y, z) \rightarrow$ distance

Scene Representation Networks
(Sitzmann et al. 2019)

$(x, y, z) \rightarrow$ latent vec. (color, dist.)

Differentiable Volumetric Rendering
(Niemeyer et al. 2020)

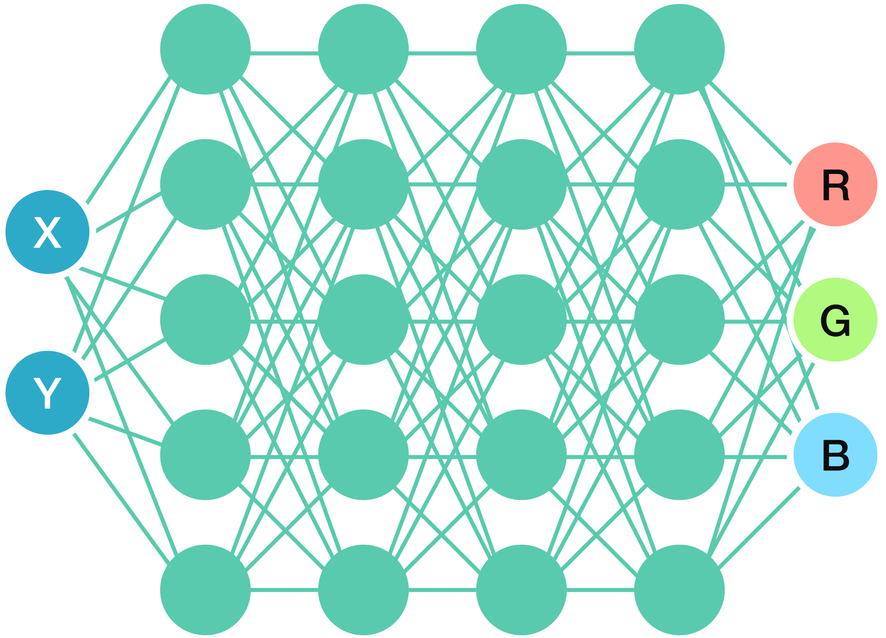
$(x, y, z) \rightarrow$ color, occ.

Challenge:

How to get MLPs to represent higher frequency functions?

Rahaman et al. 2019, Basri et al. 2020

Image Representation





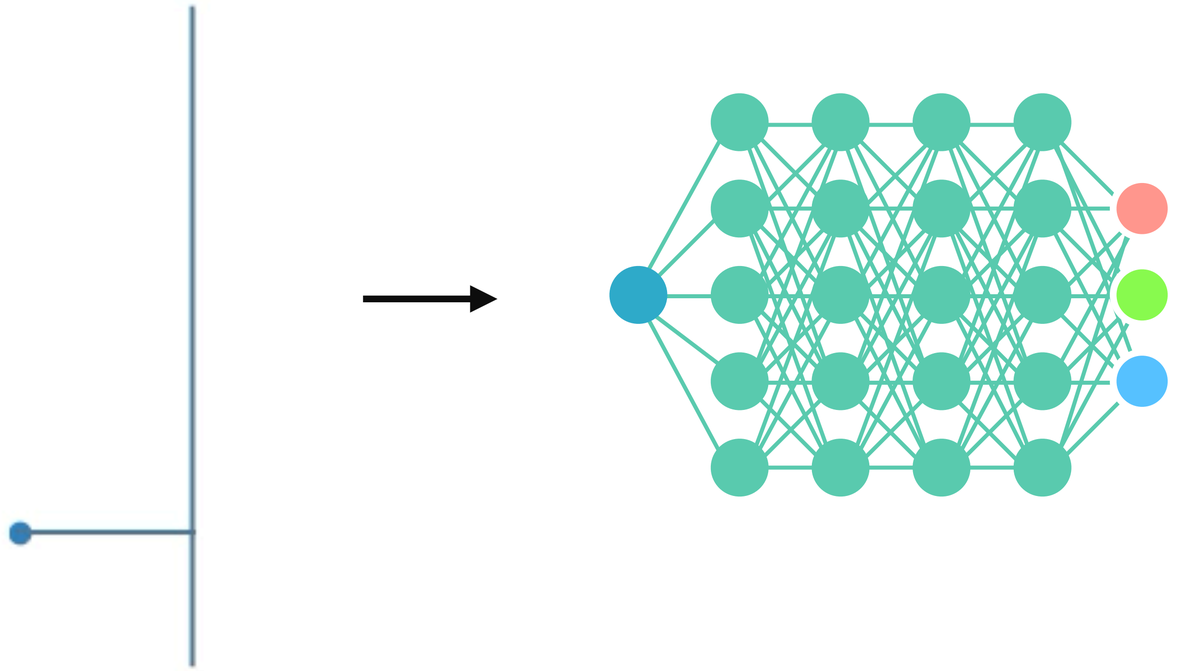
MLP output



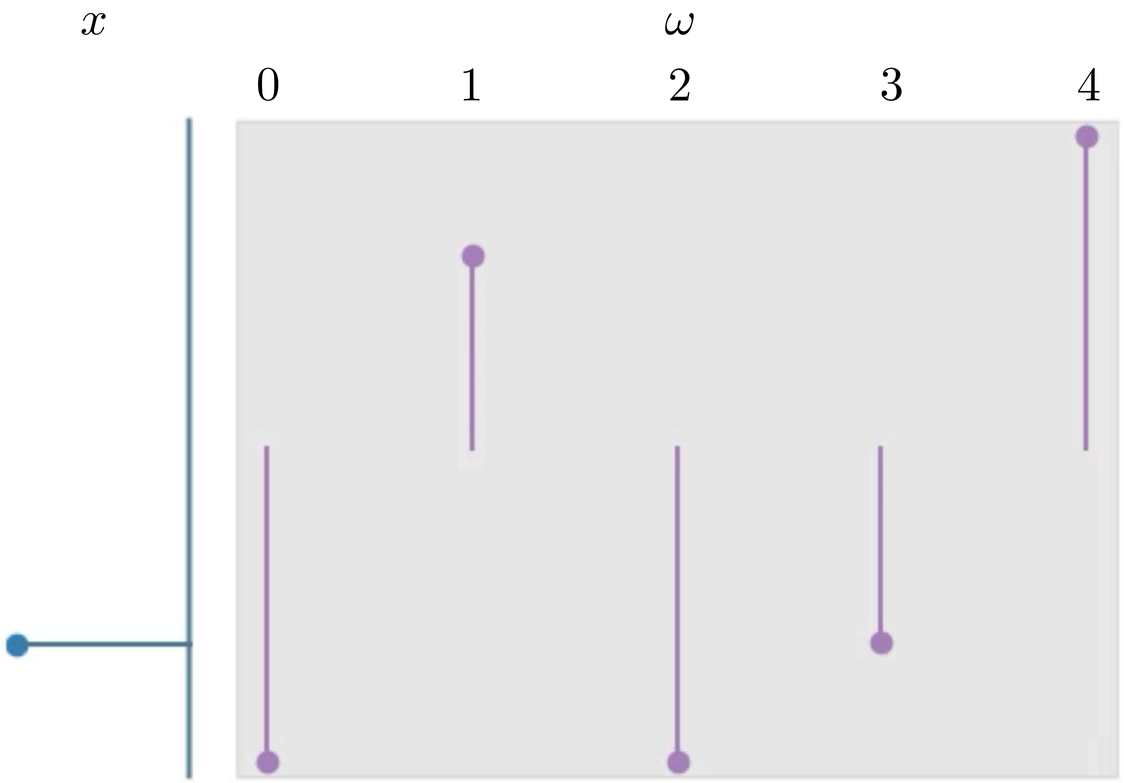
Supervision image

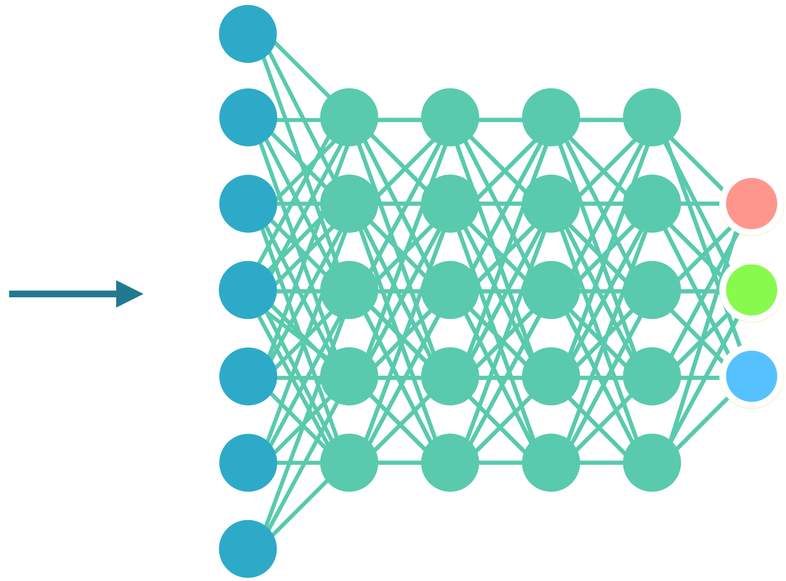
Standard input

x



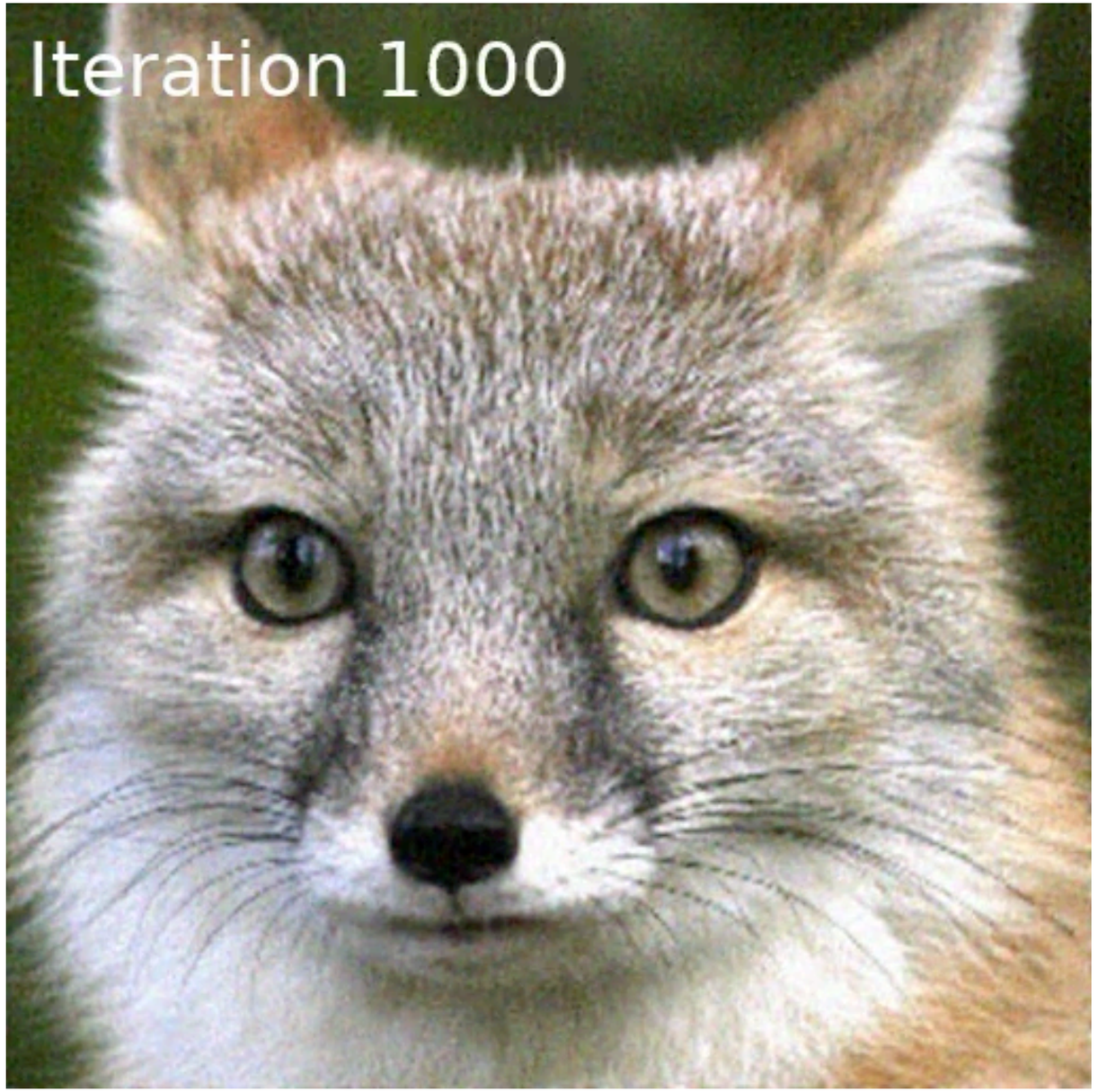
Standard input Positionally Encoded input







Standard MLP



MLP with Fourier features

Why does positional encoding help?



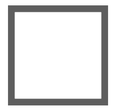
Target Image

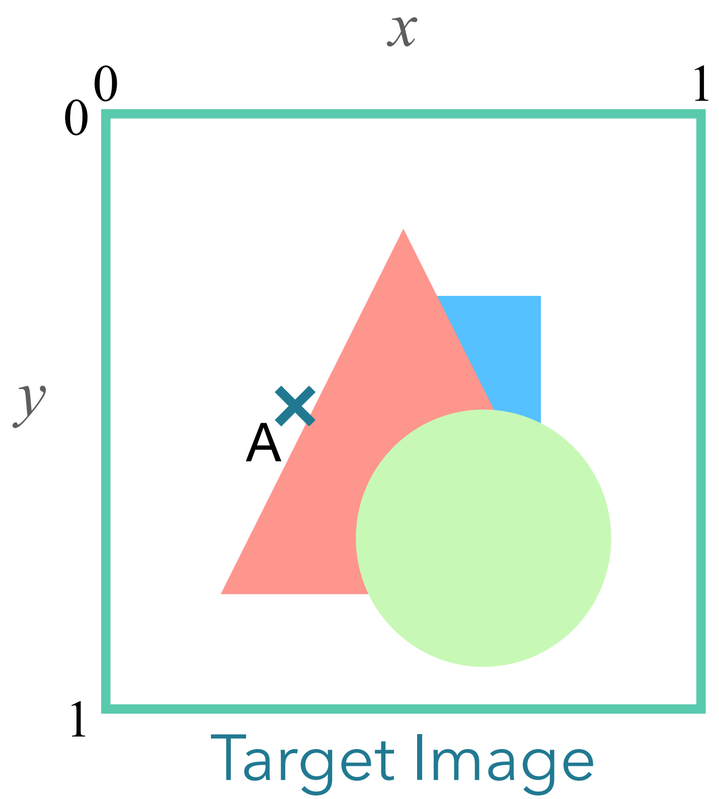
Why does positional encoding help?

Input

	x	y
A	.36	.5

Target

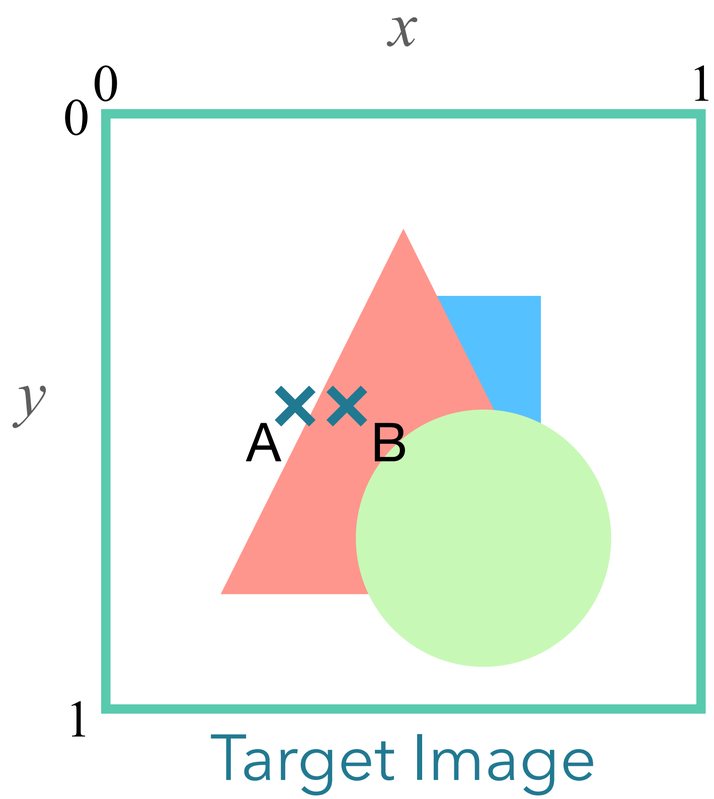




Target Image

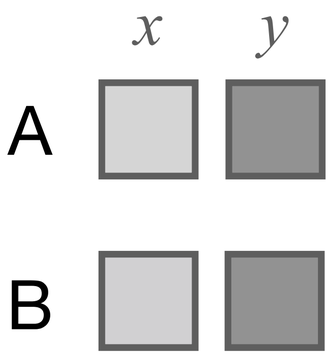
Why does positional encoding help?

	Input		Target
	x	y	
A	.36	.5	
B	.38	.5	

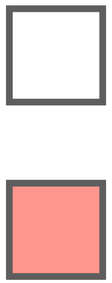


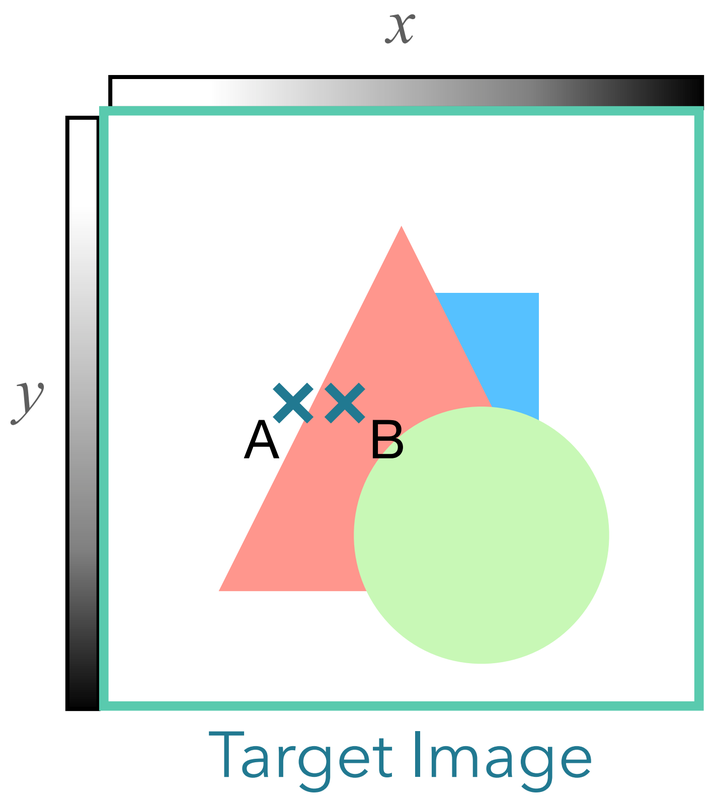
Why does positional encoding help?

Input



Target



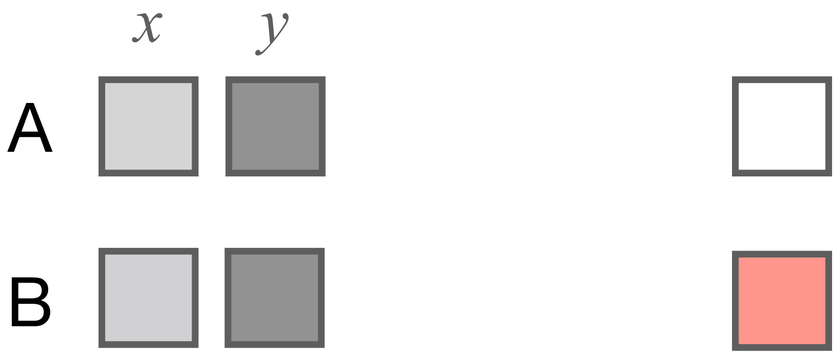


Target Image

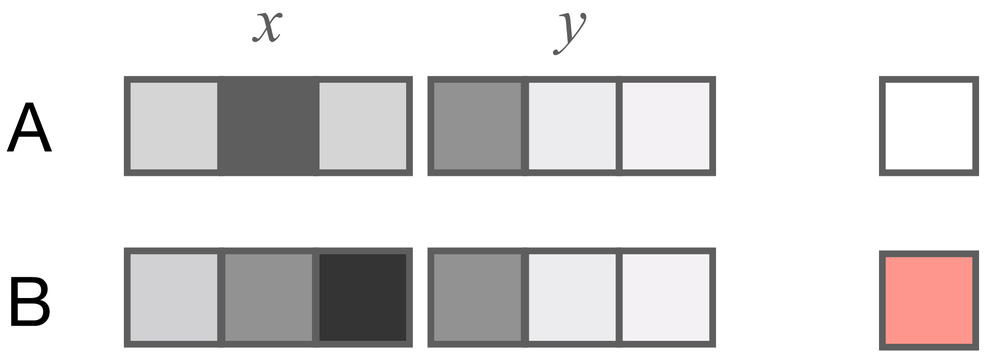
Why does positional encoding help?

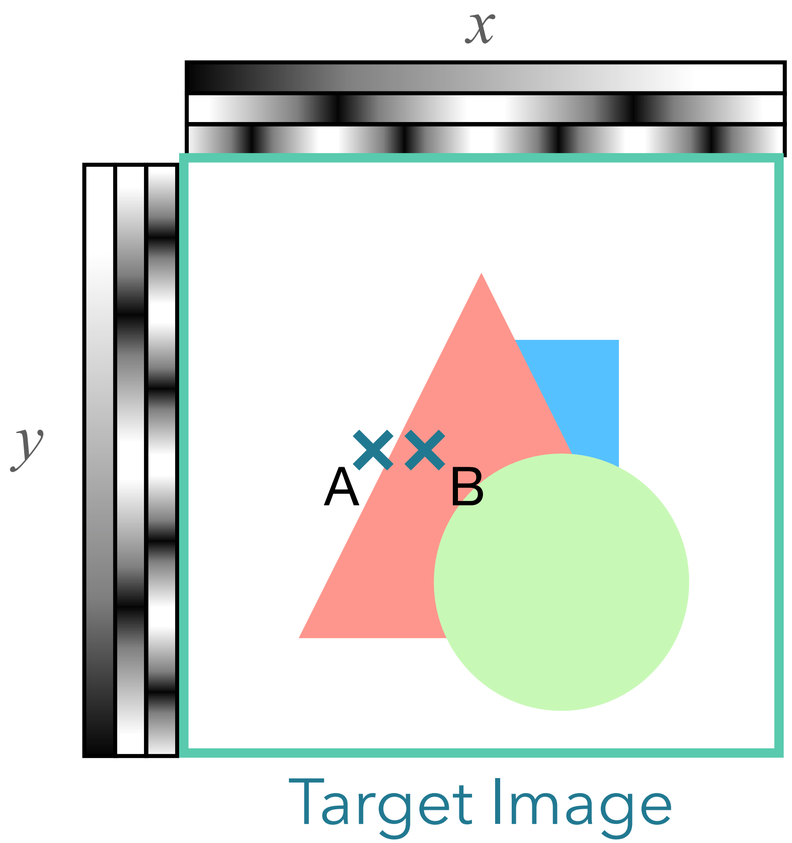
Input

Target



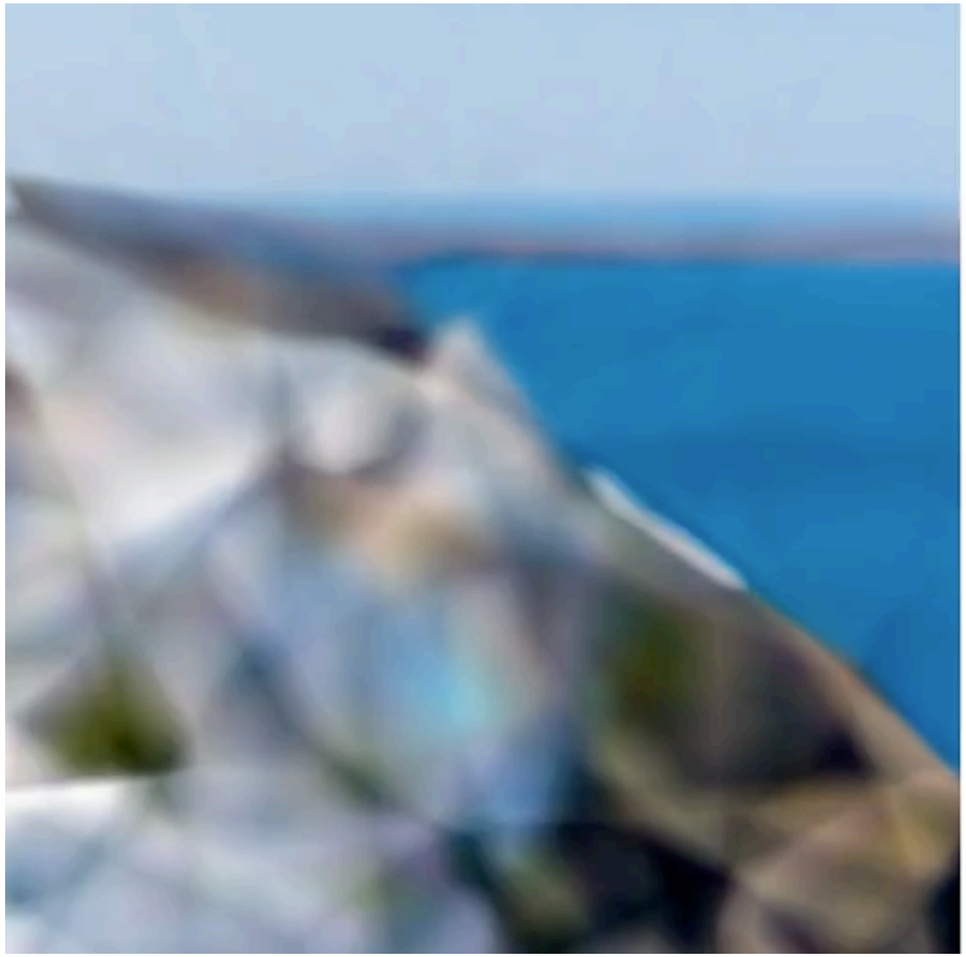
With Positional Encoding



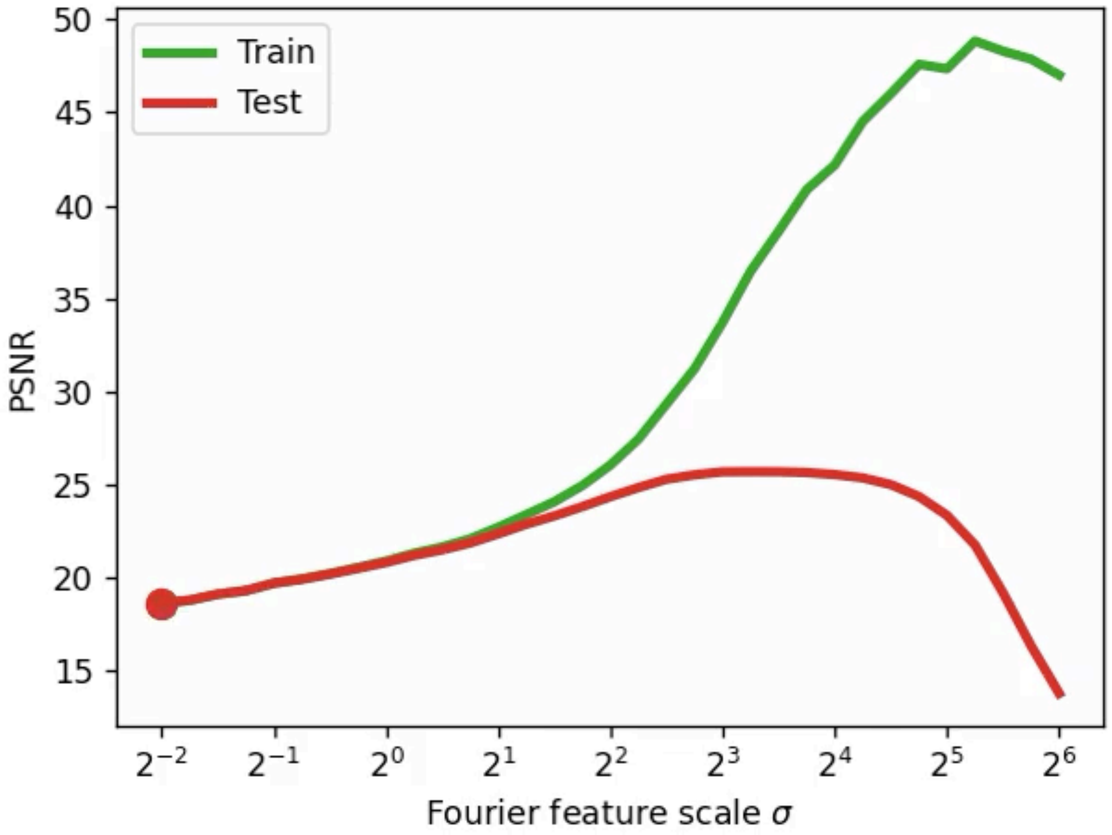


Target Image

Performance depends on max encoding frequency



Network output

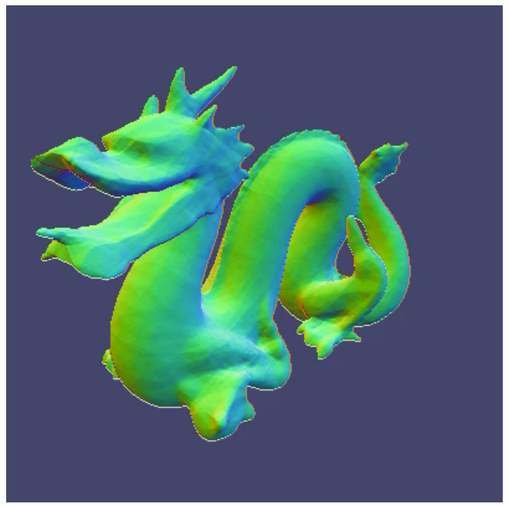


Performance vs. scale value

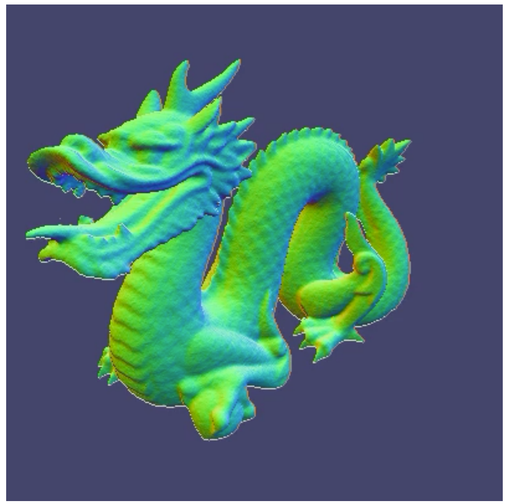
Tancik*, Srinivasan*, Mildenhall* et al. 2020, *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains*

Coordinate-based MLPs can replace any low-dimensional array

Without Encoding



With Encoding



3D Shape

NeRF with and without positional encoding



NeRF (Naive)

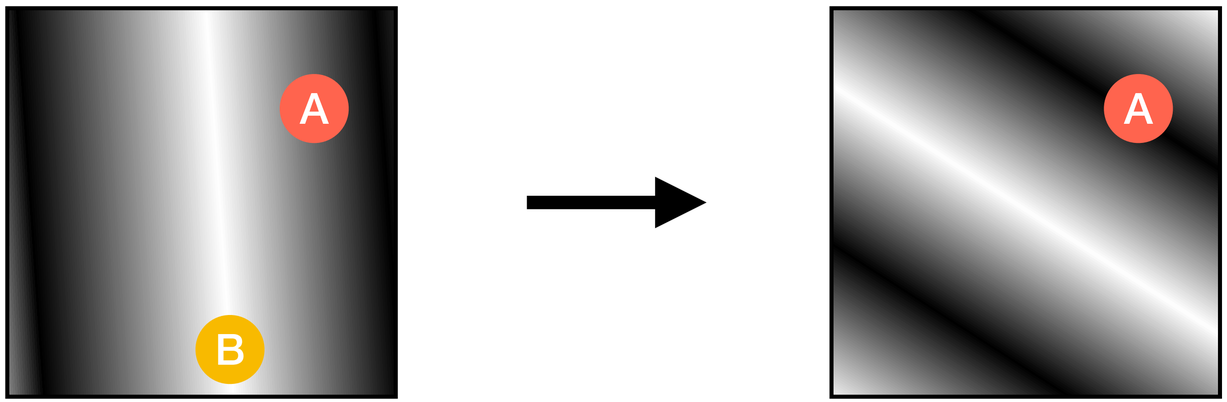


NeRF (with positional encoding)

Can we learn the encoding?

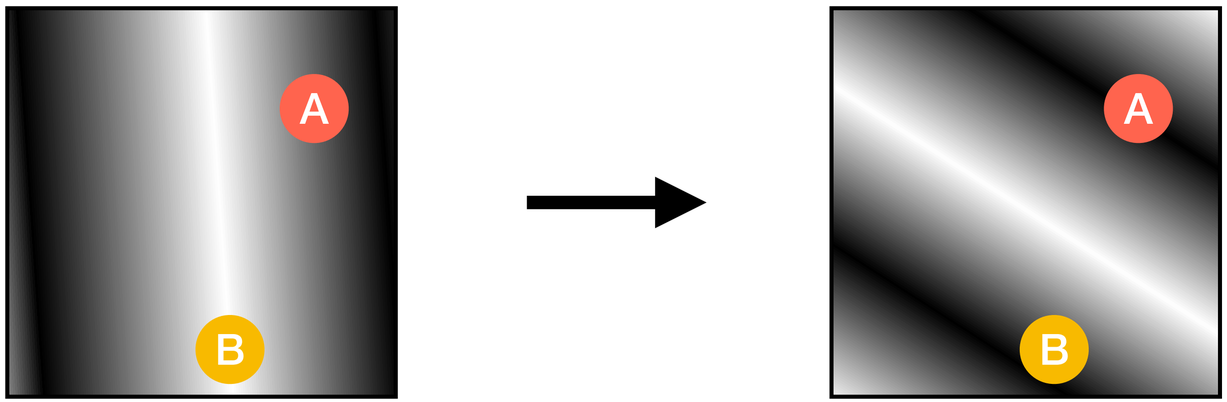
Learnable Encodings

Lets try optimizing phase and frequency s.t. A goes to ■



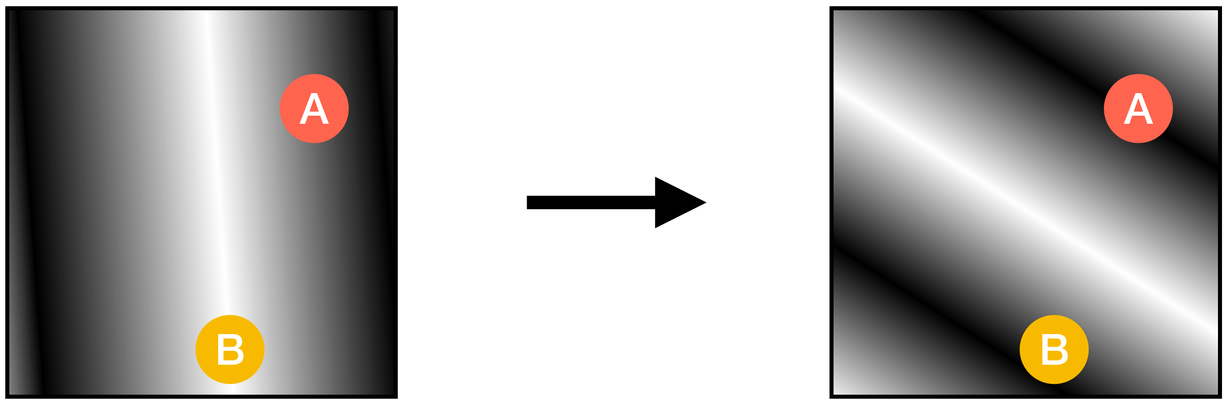
Learnable Encodings

Lets try optimizing phase and frequency s.t. A goes to ■



Learnable Encodings

Lets try optimizing phase and frequency s.t. A goes to ■



B also changes which makes optimization difficult

Desired: Learnable encoding with local extent

Desired: Learnable encoding with local extent

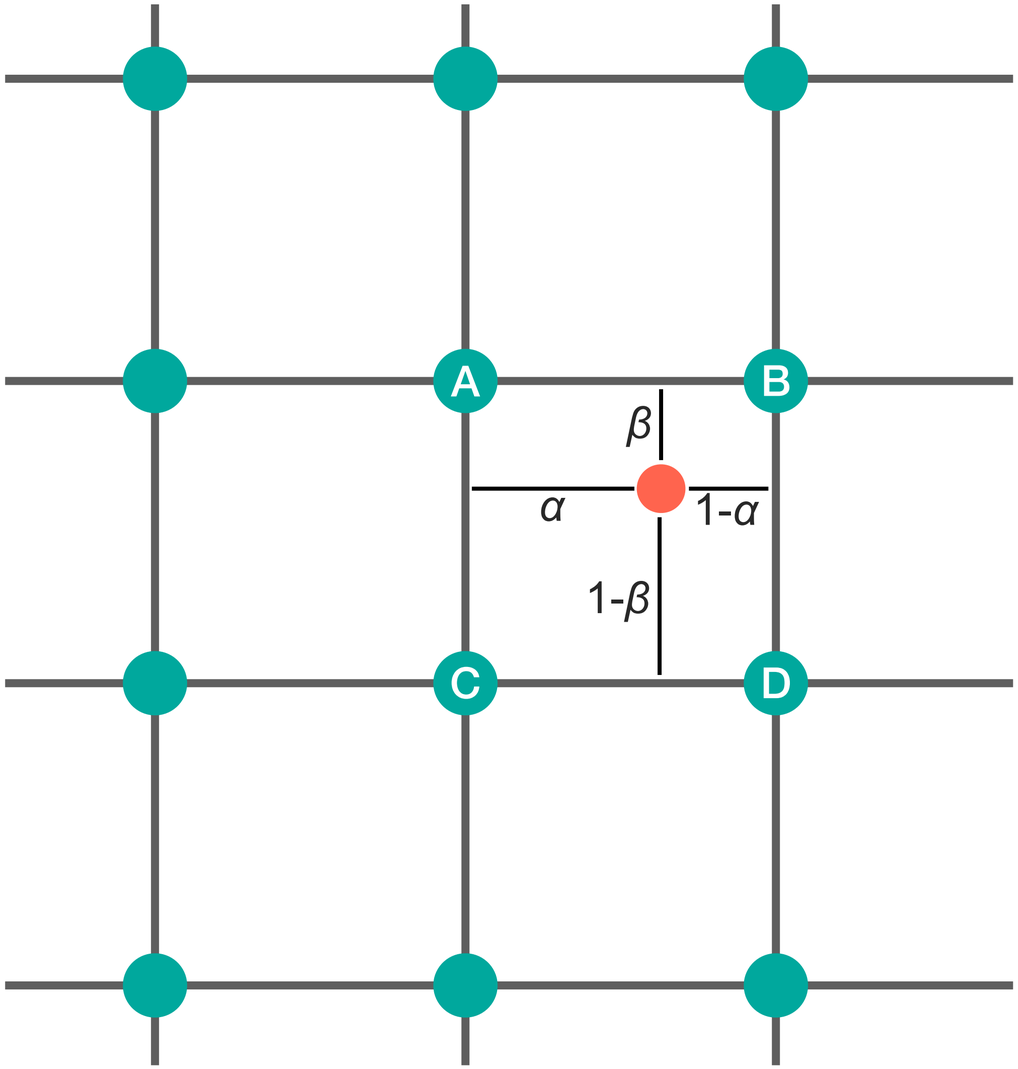
Solution: Feature Grids (i.e. voxel)

Desired: Learnable encoding with local extent

Solution: Feature Grids (i.e. voxel)

Challenge: Resolution

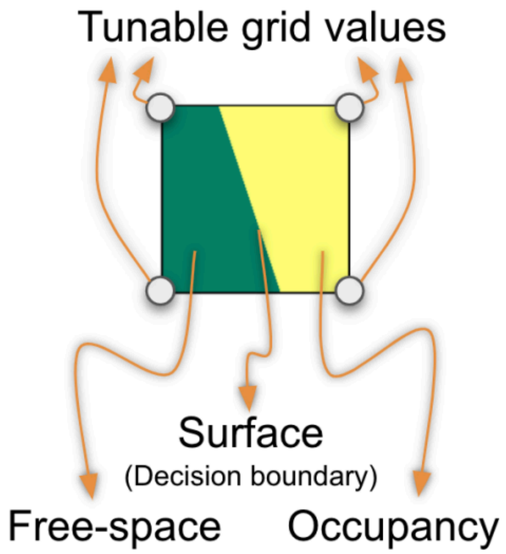
Make continuous via interpolation

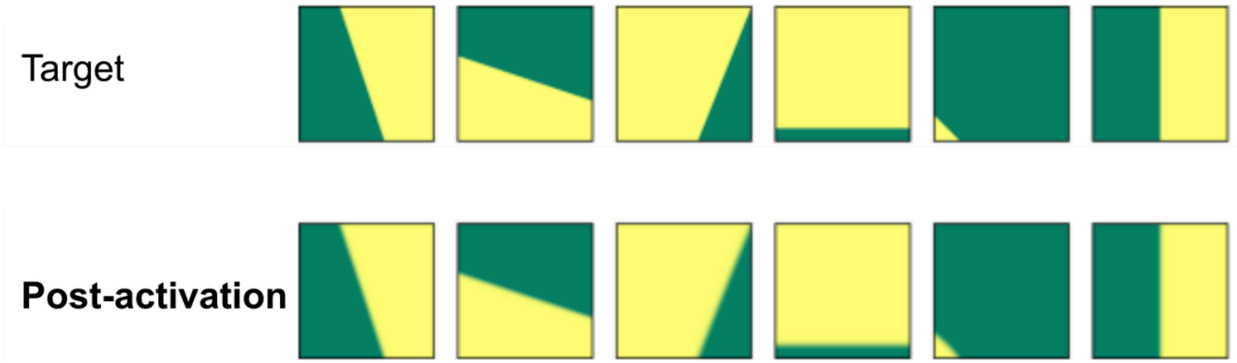


$$\bullet = \beta(\alpha A + (1 - \alpha B))$$
$$+(1 - \beta)(\alpha C + (1 - \alpha D))$$

Feature grids can be effective

Toy task for a 2D grid cell







Sun et al. 2021, Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction

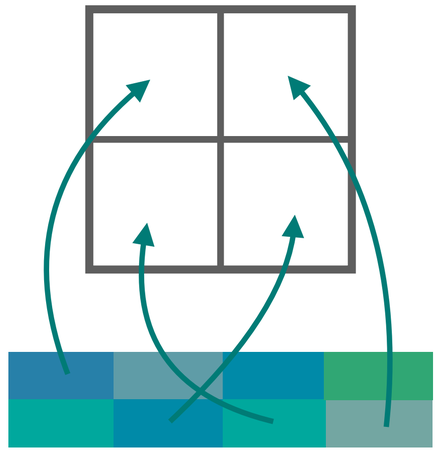
Compression Techniques



Sparsity

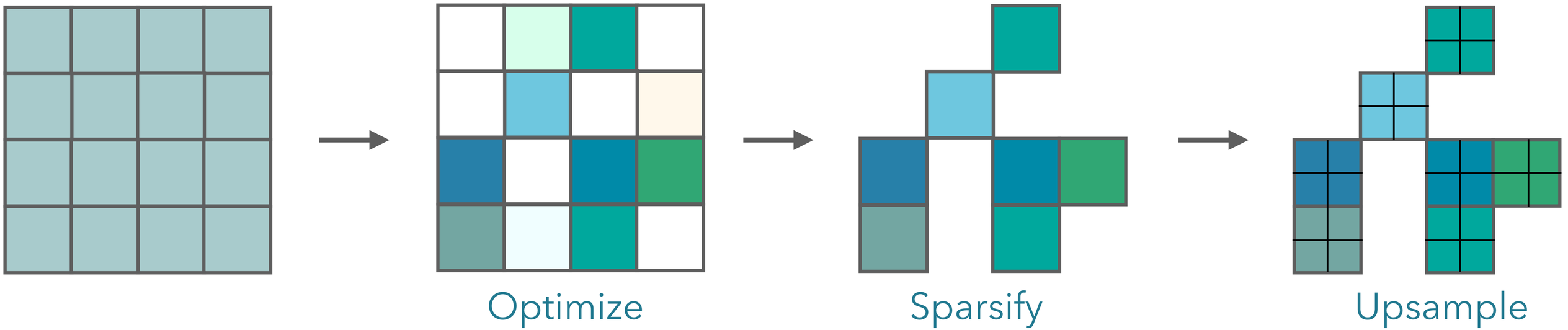


Low Rank



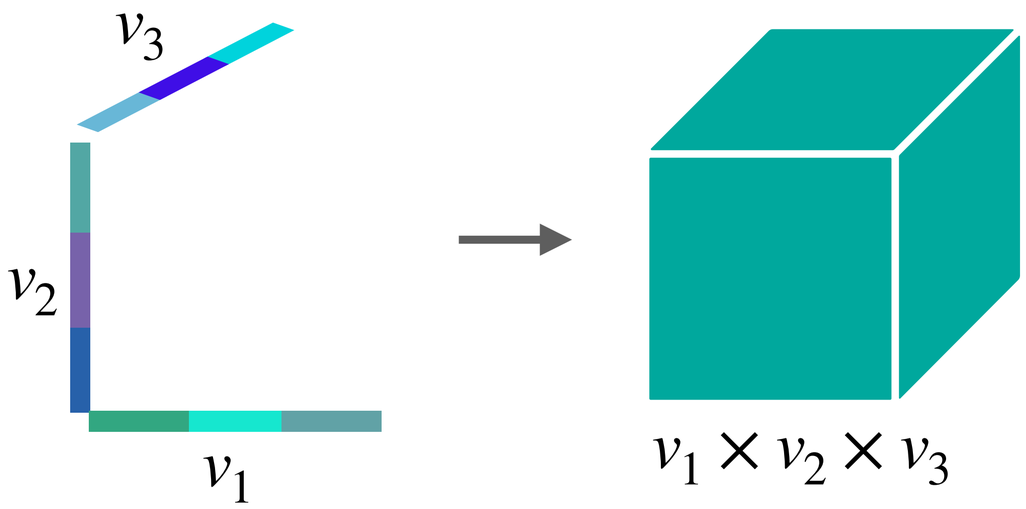
Dictionary

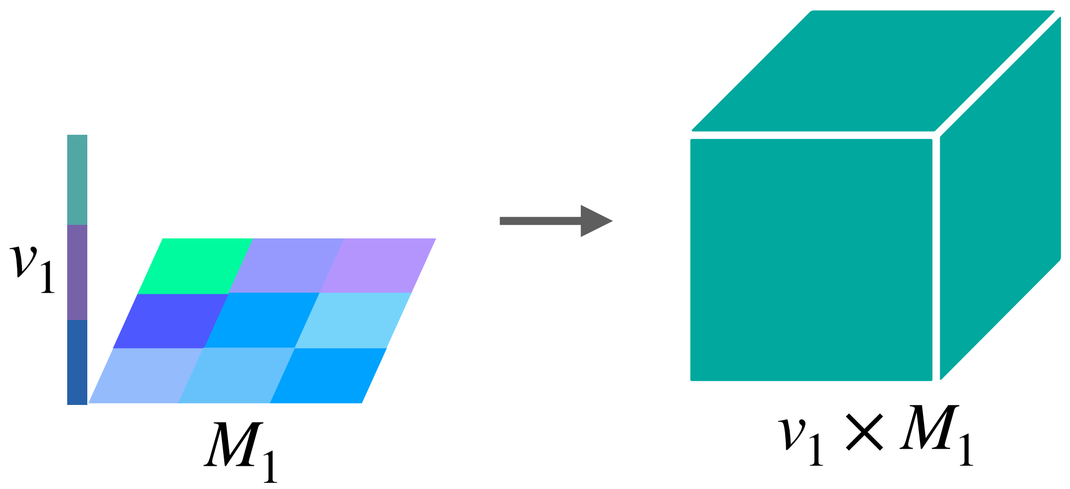
Sparsity



Lui et al. 2020, Neural Sparse Voxel Fields
Yu*, Friedovich-Keil* et al. 2021, Plenoxels: Radiance Fields without Neural Networks
Sun et al. 2021, Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction

Low Rank Approximations

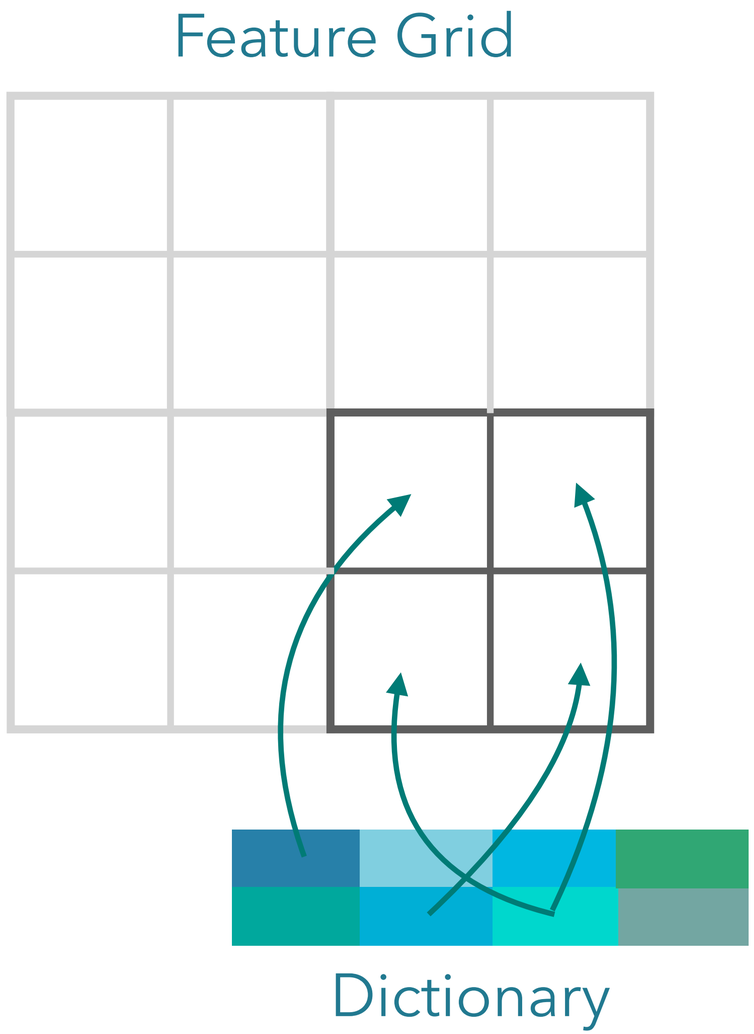




Chen*, Xu* et al. 2022, TensoRF Tensorial Radiance Fields

Chan*, Lin*, Chan*, Nagano* et al. 2022, EG3D: Efficient Geometry-aware 3D Generative Adversarial Networks

Dictionary Methods

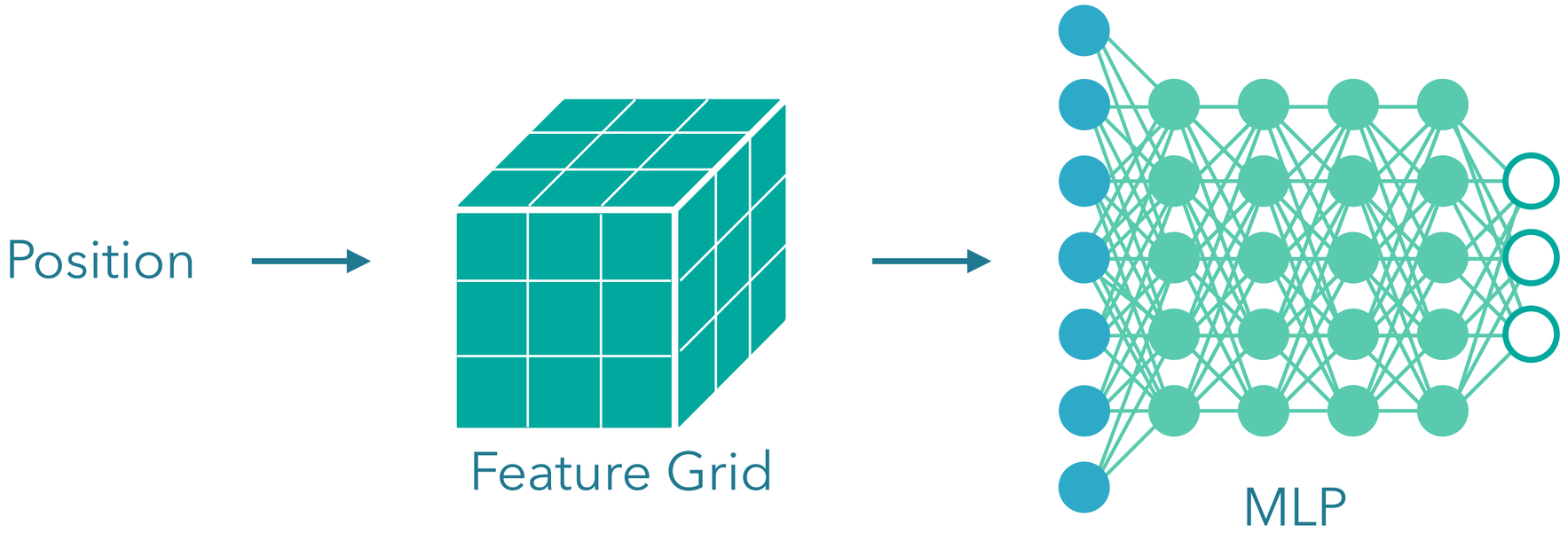


Feature Grid > Dictionary

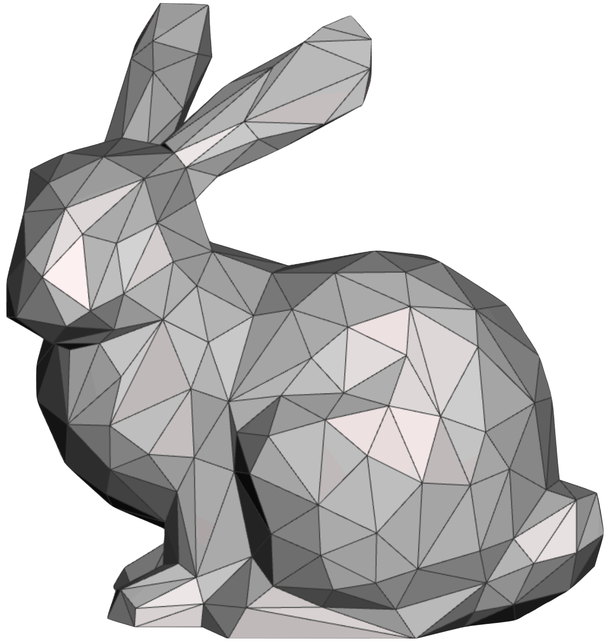
Mapping with collisions

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

Feature Grids as MLP Encodings



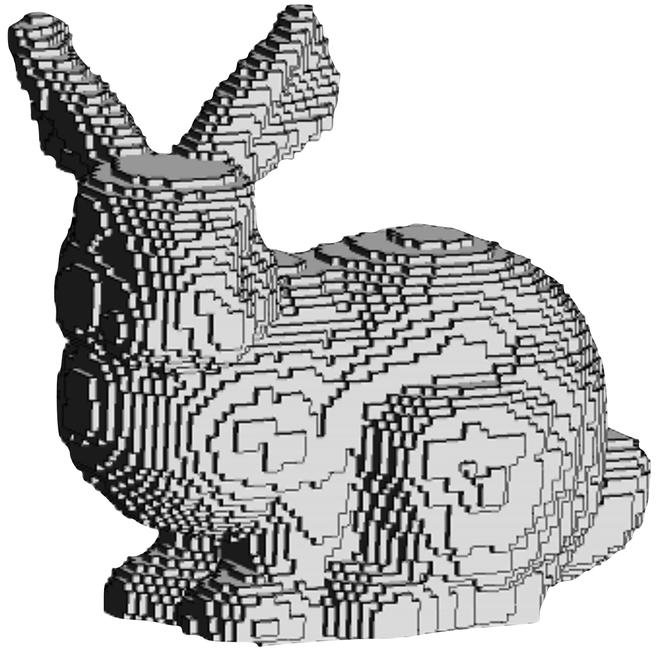
Revisiting Geometry Representations



Mesh Representation

Small memory footprint

Hard to optimize



Voxel Representation

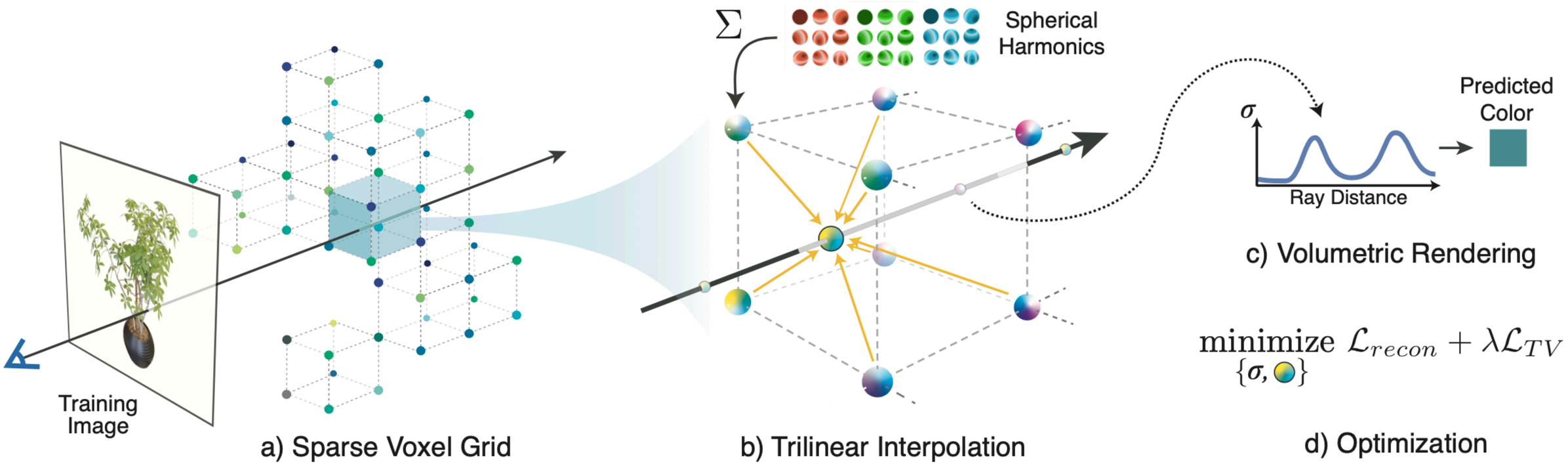
?

Easy to optimize

Large memory footprint

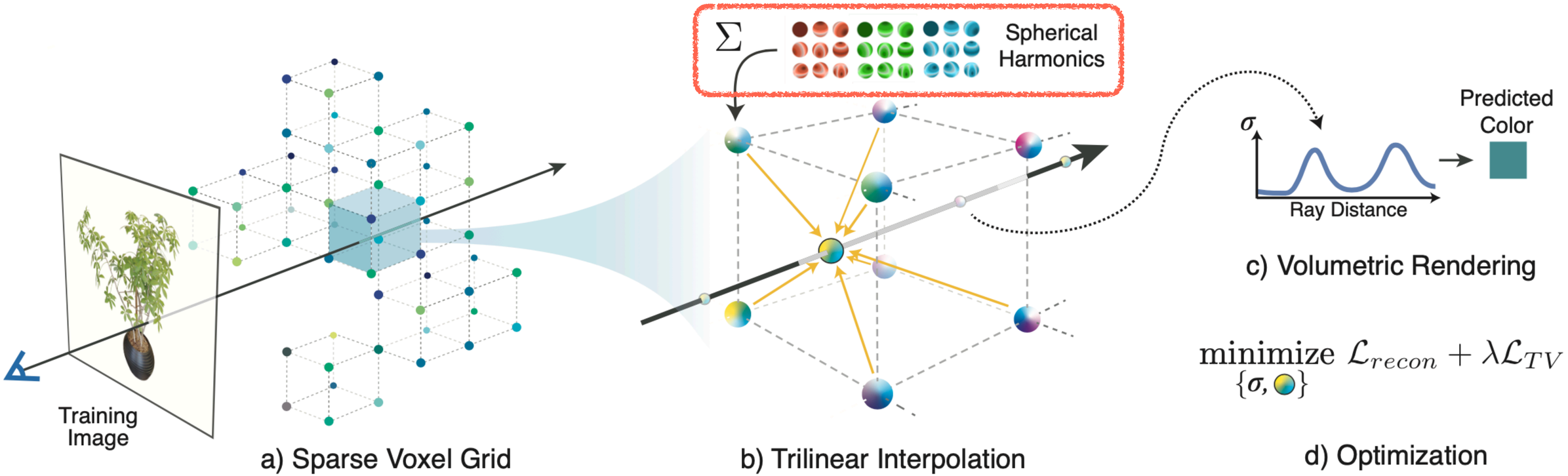
?

MLPs are not required…



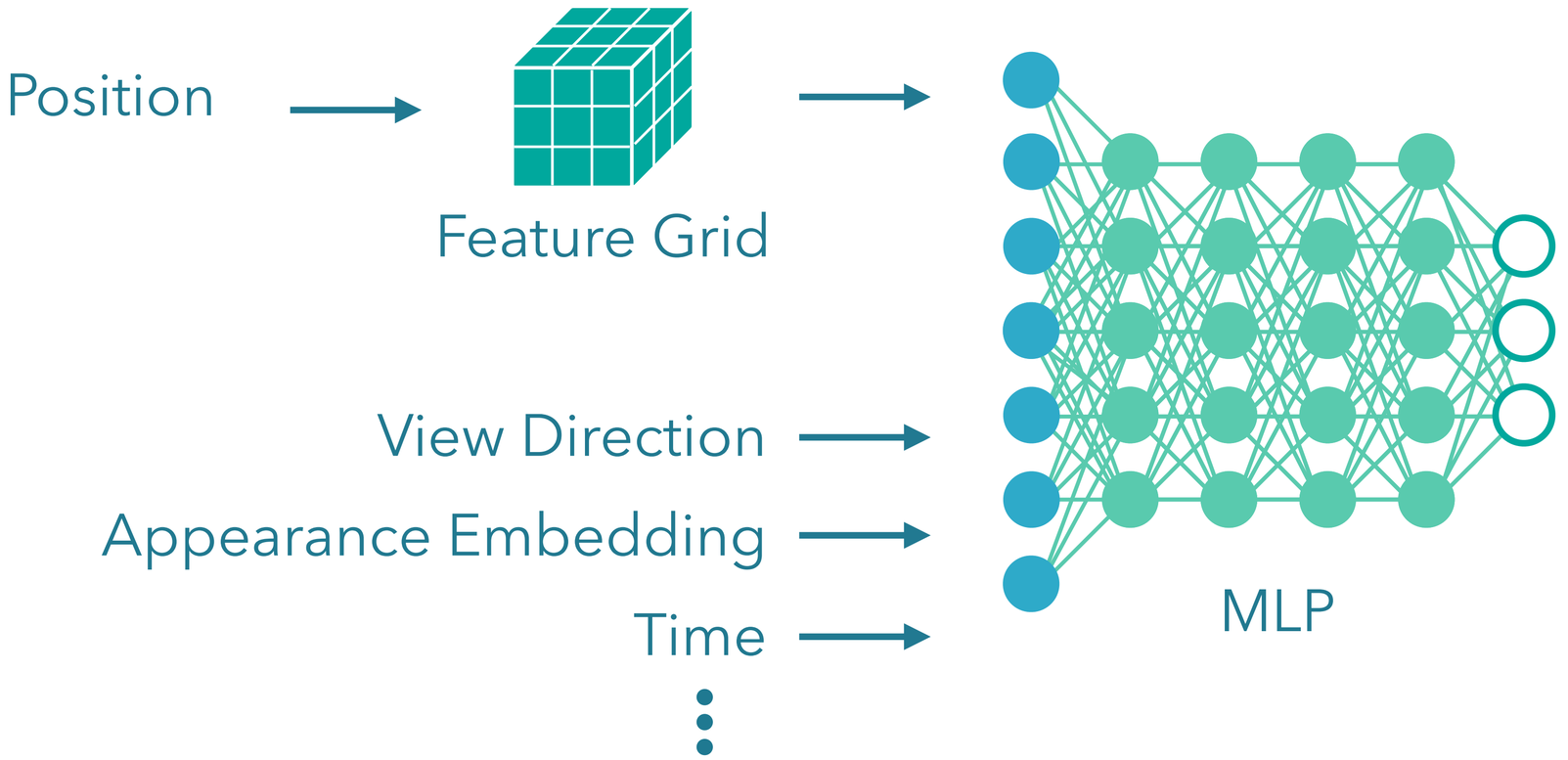
Yu*, Friedovich-Keil* et al. 2021, Plenoxels: Radiance Fields without Neural Networks

MLPs are not required…



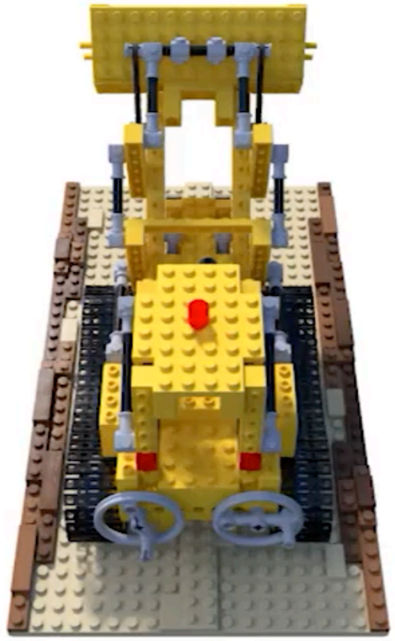
Yu*, Friedovich-Keil* et al. 2021, Plenoxels: Radiance Fields without Neural Networks

But MLPs are convenient



Results (again)

























Takeaways

1. MLP for 3D shape
2. MLP for view-dependent appearance
3. Single scene 3D reconstruction
4. Differentiable volume rendering

(None of these proposed by NeRF)

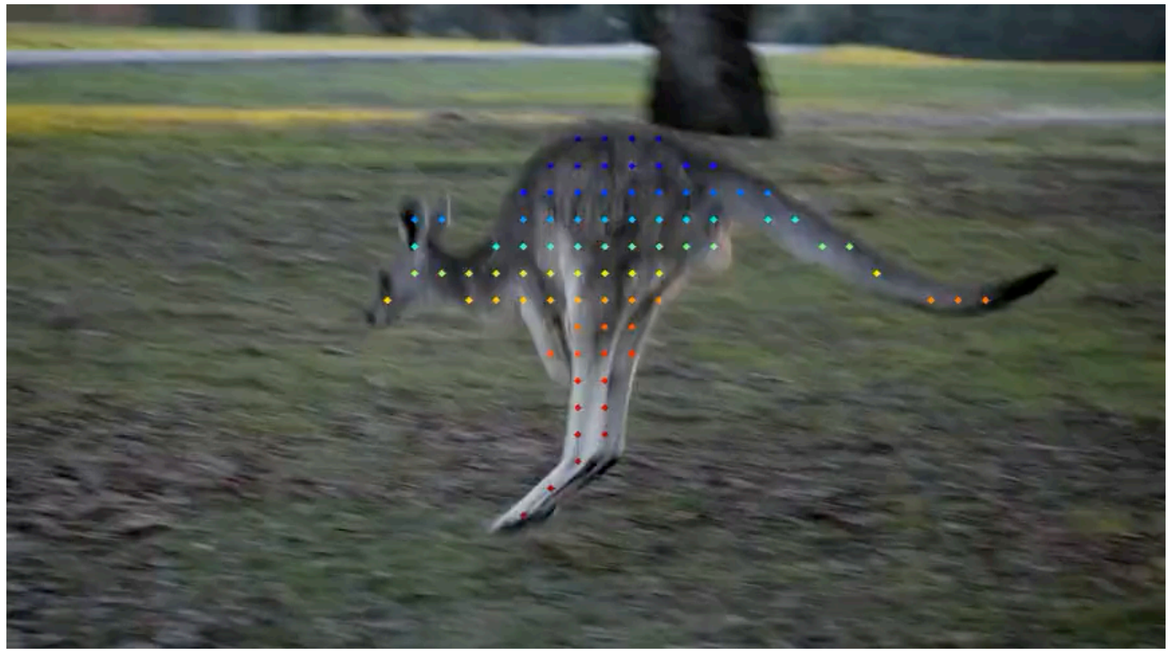
Takeaways

1. MLP for 3D shape = not great* (but useful step to better compression)
2. MLP for view-dependent appearance
3. Single scene 3D reconstruction
4. Differentiable volume rendering

MLPs are surprisingly good regularizers

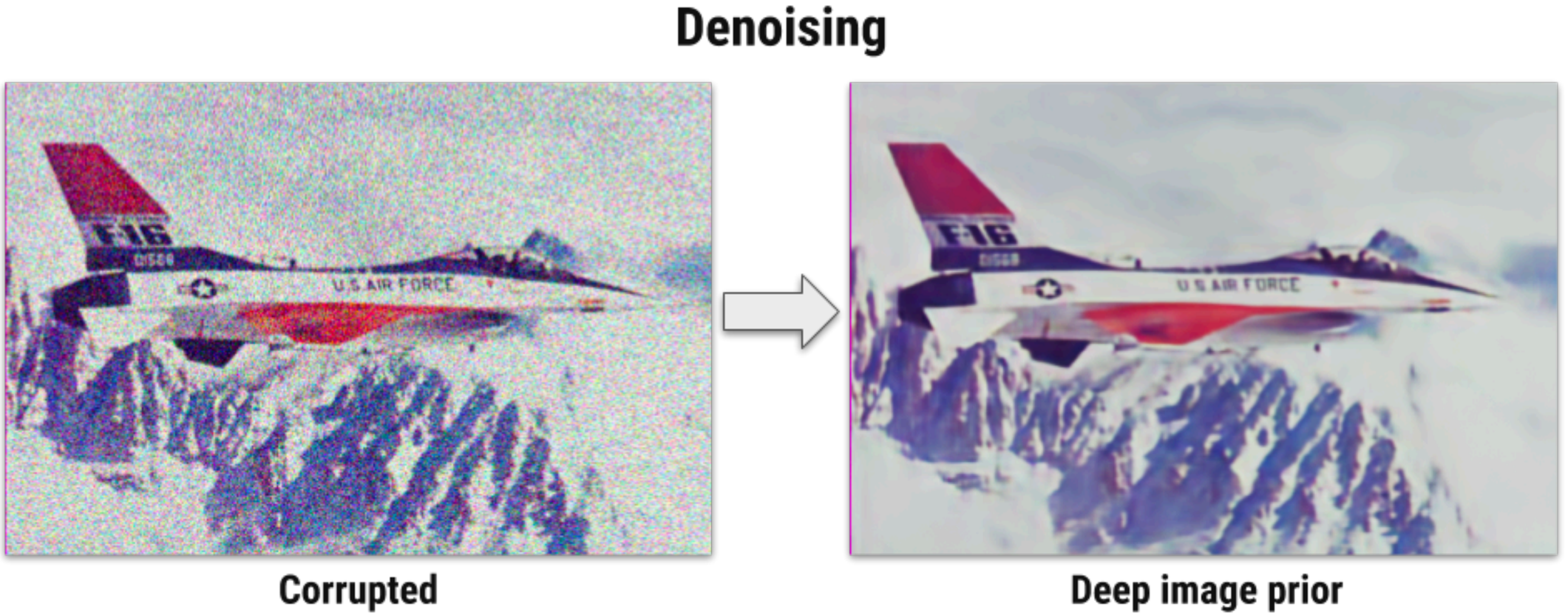


Li et al 2023, *DynIBaR*



Wang et al 2023, *Tracking Everything Everywhere All At Once*

But we already knew that



Pratul Srinivasan it's crazy how we're not seeing terrible overfitting artifacts. Like the regularization of the volume provided by the network seems to really help the overfitting from doing gradient descent on MPIs

Ulyanov et al 2018, *Deep Image Prior*

MLPs as compression

- Hard to make network weights small enough to compete with JPEG or MP4 on image or video data
- However, these codecs are not designed to be natively differentiable or optimization-friendly
- For naturally **sparse** signals being recovered in an inverse problem setup, MLPs have a competitive advantage
- Once the setup is promising enough, people develop more legitimate compression schemes

Takeaways

1. MLP for 3D shape = not great* (but necessary stepping stone?)
2. MLP for view-dependent appearance = pretty good
3. Single scene 3D reconstruction
4. Differentiable volume rendering

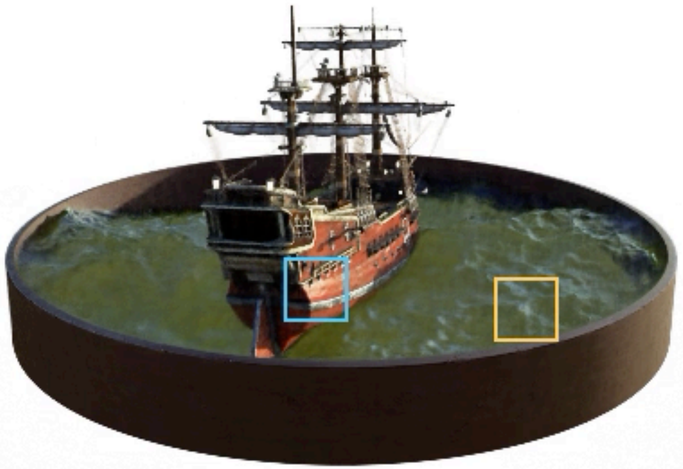
View dependent appearance representations

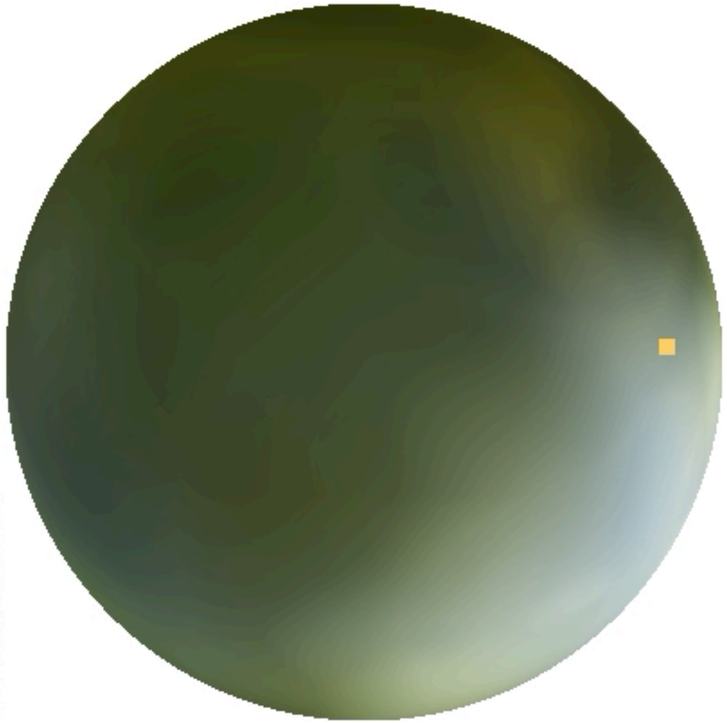
- Small MLP
- Spherical harmonics
- Spherical Gaussians

View dependence in original NeRF



Outgoing radiance distribution
for point on side of ship





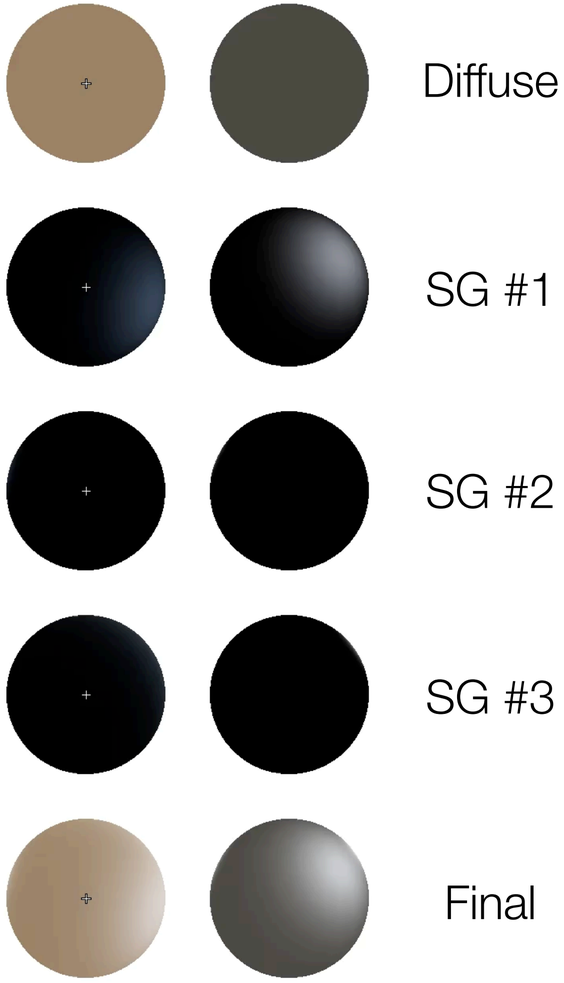
Outgoing radiance distribution
for point on water's surface

Mildenhall*, Srinivasan*, Tancik* et al. 2020, *NeRF*

One example alternative: Spherical Gaussians

View-dependent color: $\mathbf{C} = \mathbf{c}_d + \sum_i \mathbf{c}_i \exp\left(\lambda_i \left(\mu_i \cdot \mathbf{d} - 1\right)\right)$





Yariv*, Hedman* et al 2023, *BakedSDF*

Takeaways

1. MLP for 3D shape = not great* (but necessary stepping stone?)
2. MLP for view-dependent appearance = pretty good
3. Single scene 3D reconstruction = good
4. Differentiable volume rendering



Takeaways

1. MLP for 3D shape = not great* (but necessary stepping stone?)
2. MLP for view-dependent appearance = pretty good
3. Single scene 3D reconstruction = good
4. Differentiable volume rendering = very good

Convergence of volume rendering optimization



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