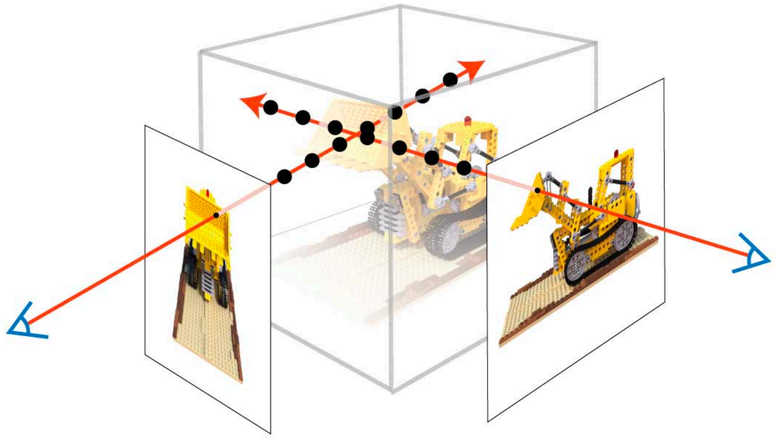
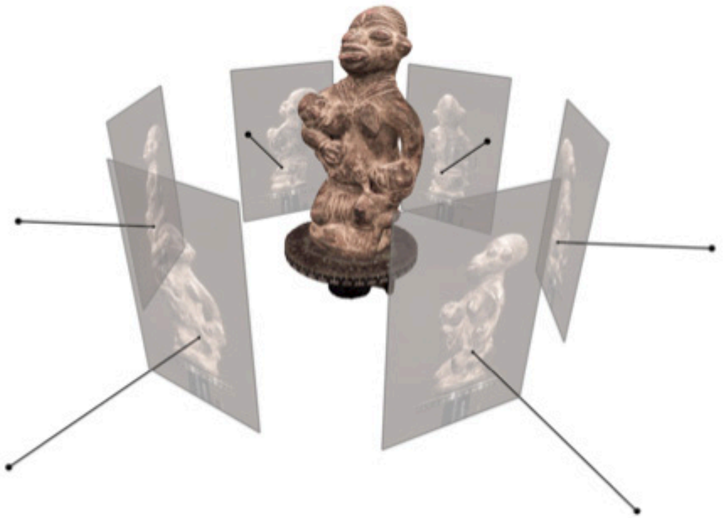
3D from Multi-view





Instructor: Shubham Tulsiani

A reminder to ask questions!

Image credits: Jorge Cham, “We have No Idea”

3D from Multi-view

Images + Cameras

3D Representation

Slide credits: Derek Hoiem & Jon Barron

3D from Multi-view

Image credits: *Noah Snavely*

3D from Multi-view

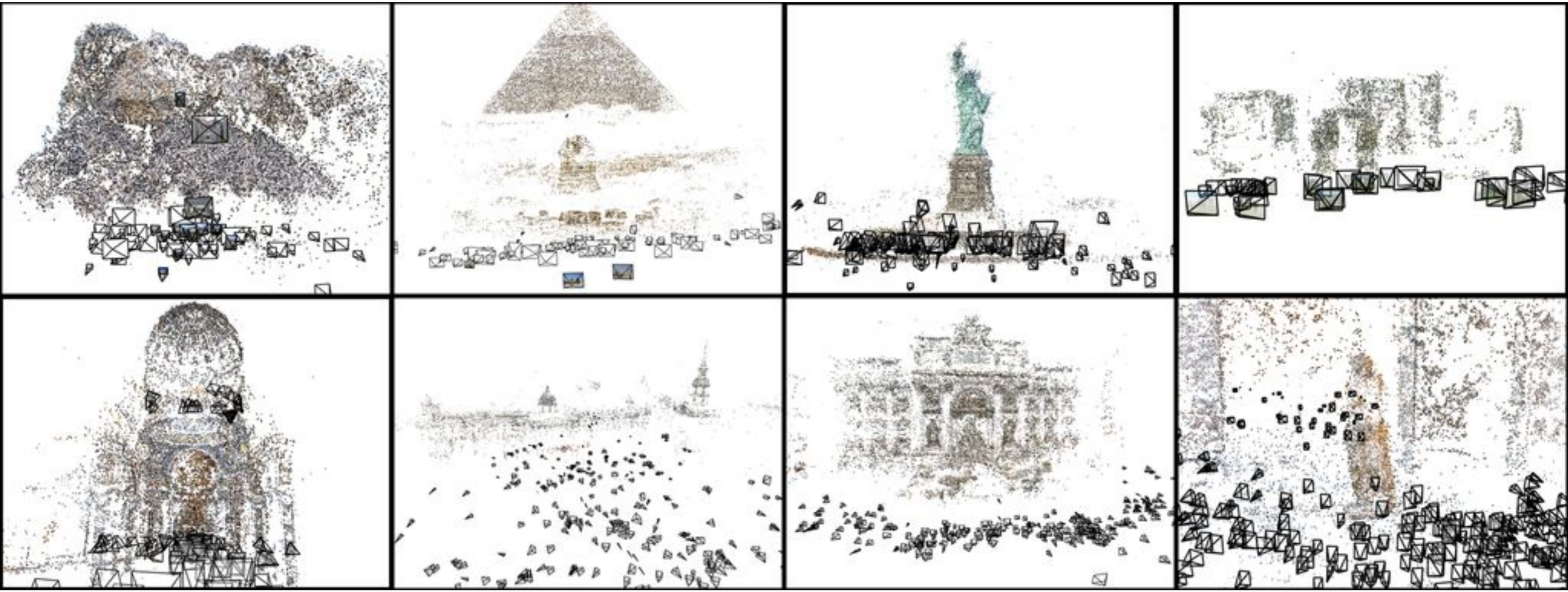


Image credits: *Noah Snavely*

3D from Multi-view

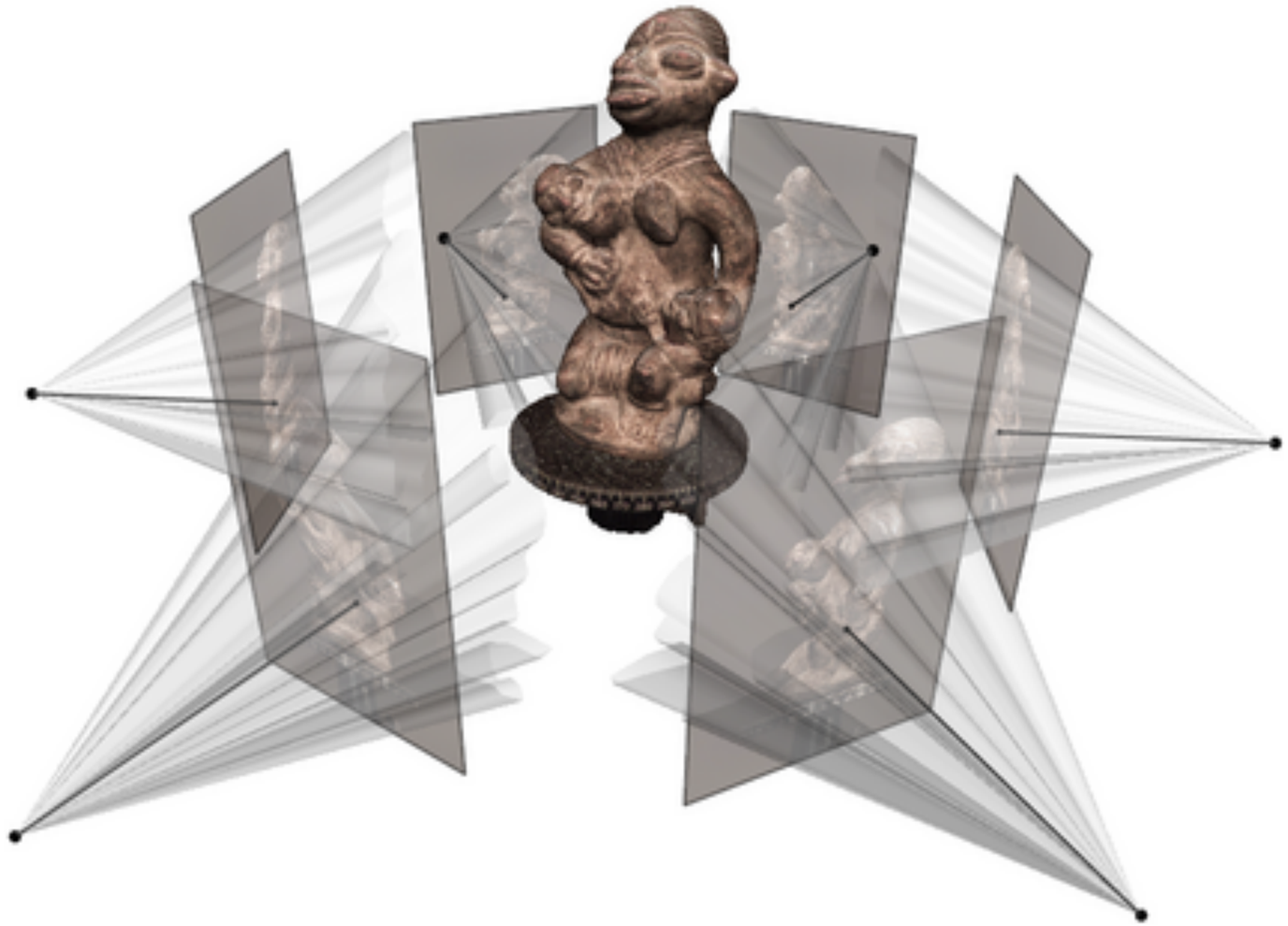
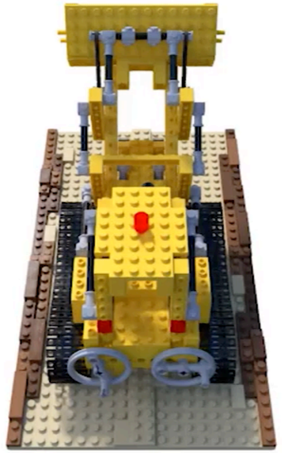


Image credits: *Carlos Hernandez*

3D from Multi-view



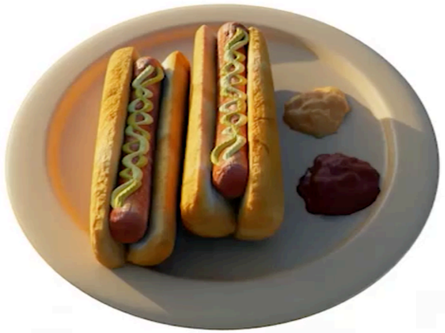


Image credits: *Mildenhall et. al.*

3D from Multi-view

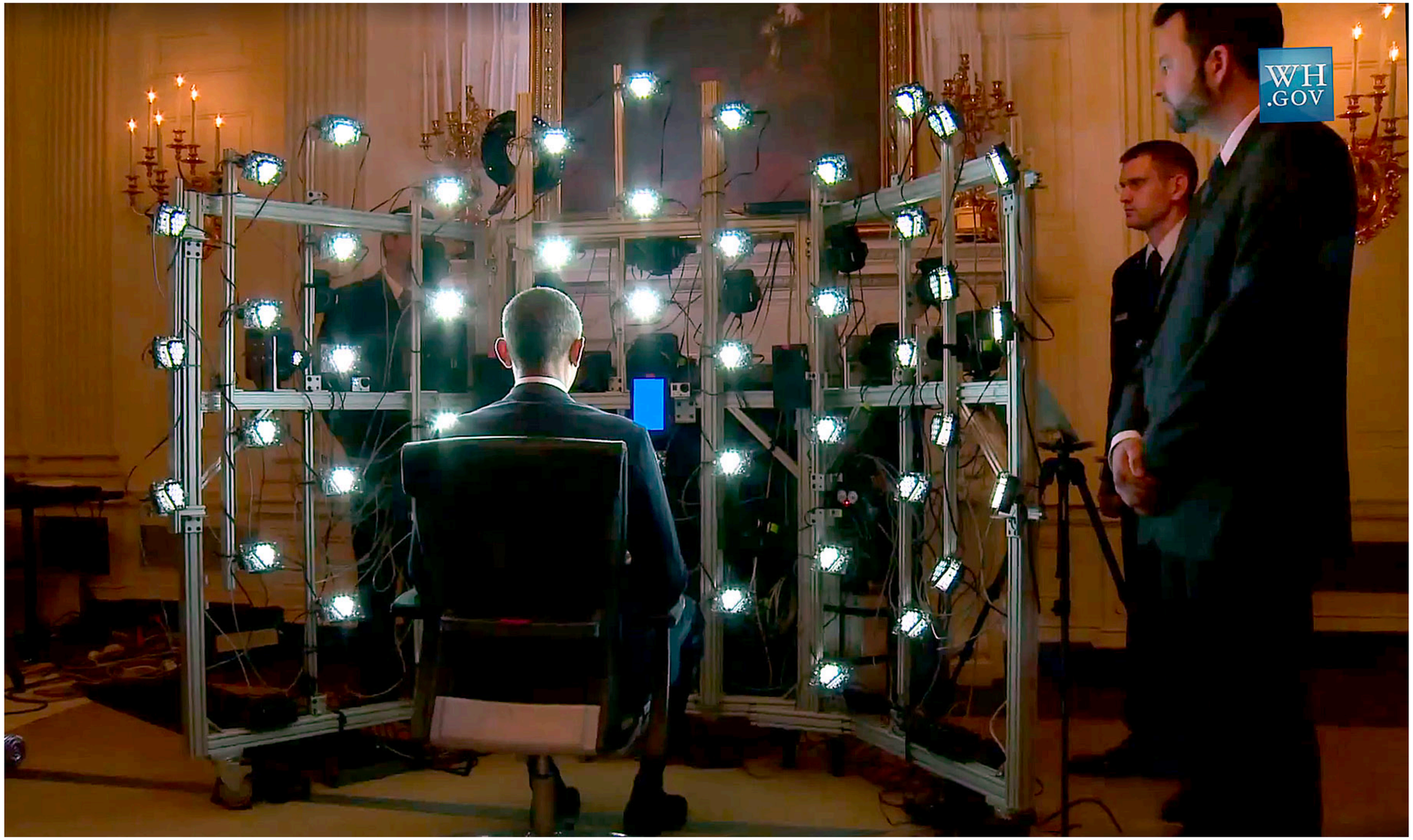


Image credits: *Paul Debevec*

3D from Multi-view

Image credits: *Paul Debevec*

3D from Multi-view

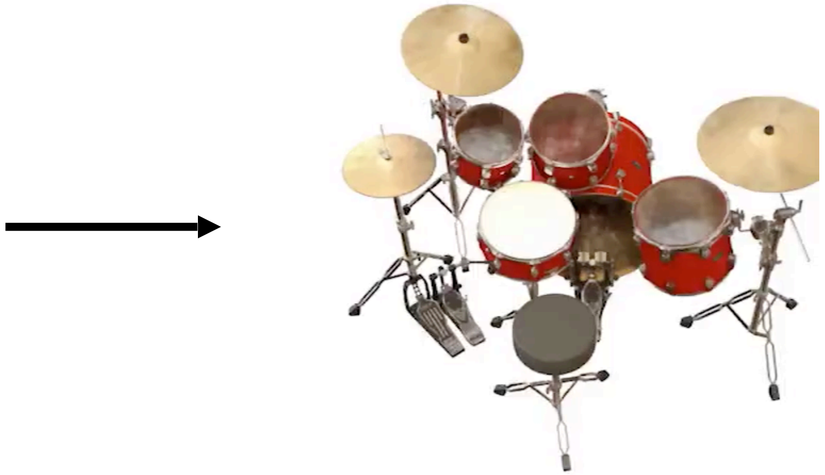
Images + Cameras

3D Representation

Slide credits: Derek Hoiem & Jon Barron

3D from Multi-view

Images + Cameras



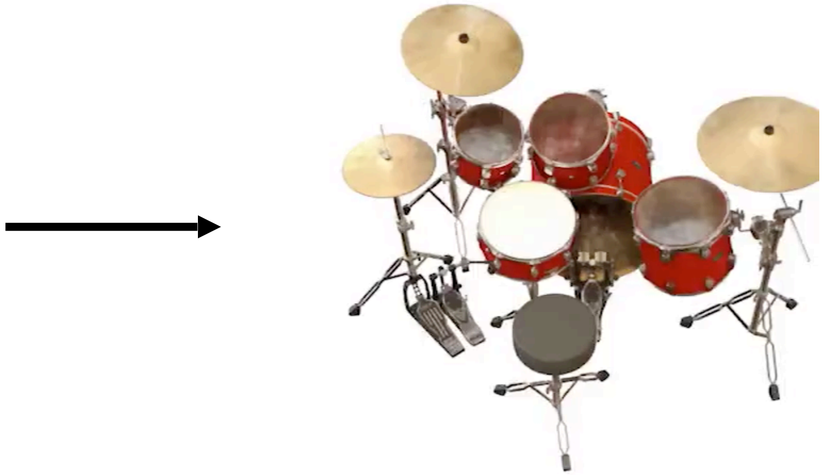
3D Representation

What representation?

How can we infer it from posed images?

3D from Multi-view

Images + Cameras



3D Representation

What representation?

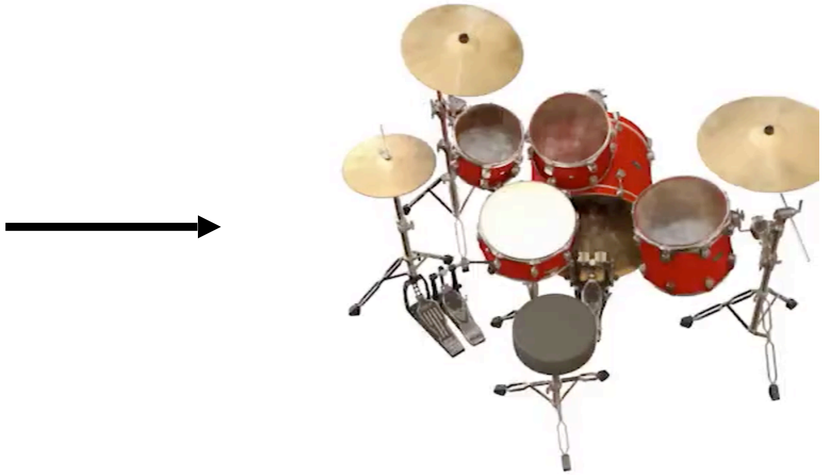
How can we infer it from (posed) images?

Key idea: Find a 3D representation (**x**) whose renderings are consistent with all observed views

$$\min_{\mathbf{x}} \sum_{i} \|Render(\mathbf{x}, \pi_i) - I_i\|^2$$

3D from Multi-view

Images + Cameras



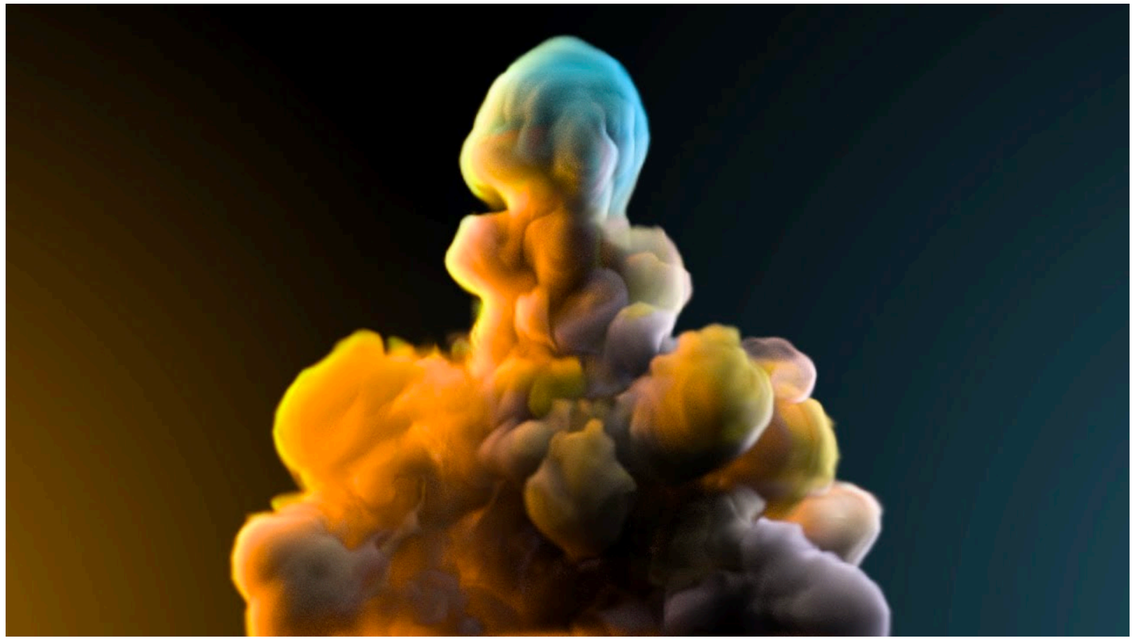
3D Representation

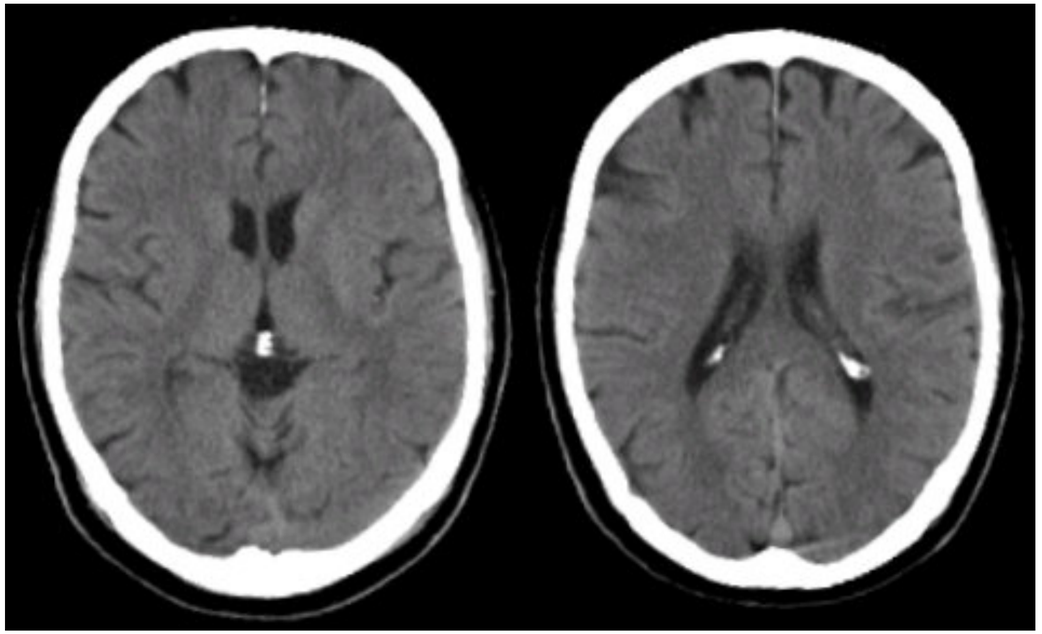
What representation?
(and how to render it)

How can we infer it from posed images?

Today's goal: Understand two popular choices and corresponding rendering mechanisms

(Neural/Grid-based) Density Fields

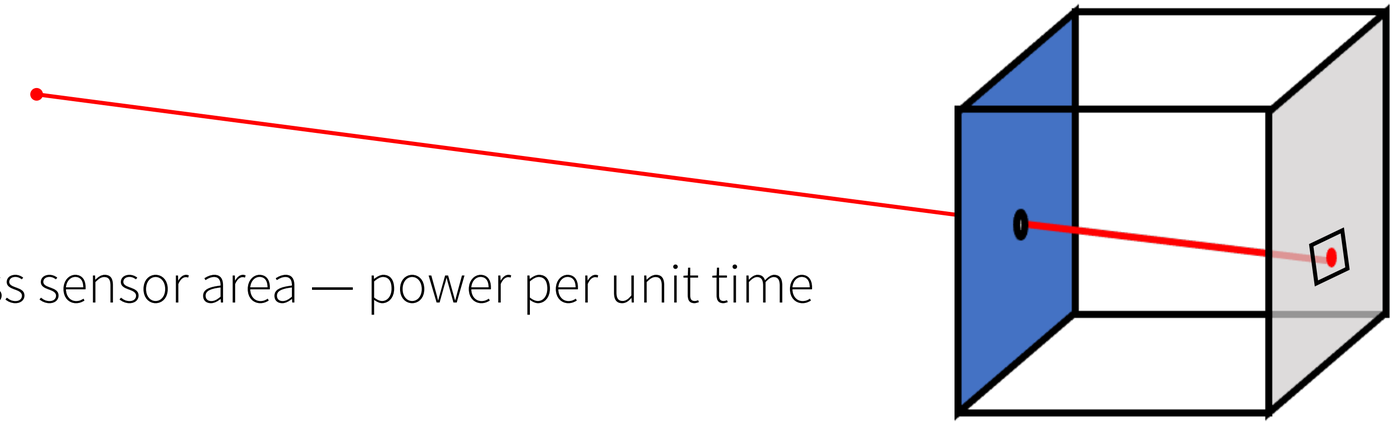




$$V[x, y, z] \in \mathbf{R}^+ \qquad f(\mathbf{p}) \in \mathbb{R}^+$$

Q: How to render such volumetric representations?

What does a pixel measure?



Flux across sensor area — power per unit time

<u>Definition</u>: The field *radiance* (*luminance*) at a point in space in a given direction is the power per unit solid angle per unit area perpendicular to the direction

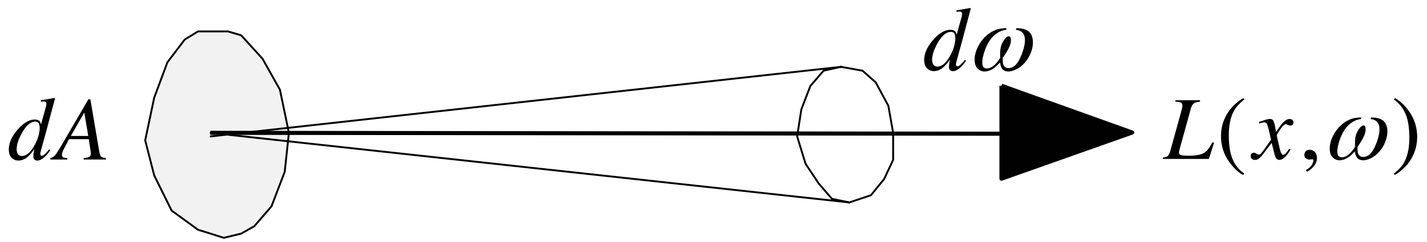
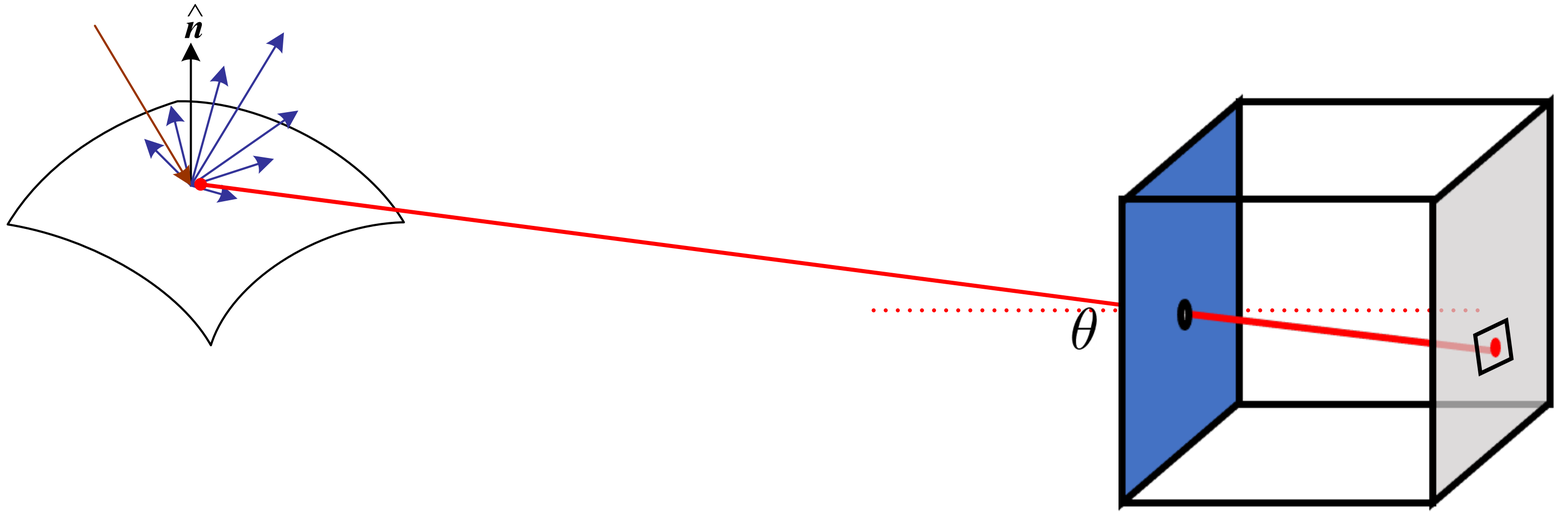


Image credits: Pat Hanrahan

Surface Rendering



$$\propto L(\mathbf{x}^*, \omega)$$

radiance for: ***x**** = optical centre, *w* = direction from x* to pixel sensor

$$= L(\mathbf{x}^* - \lambda\omega, \omega)$$

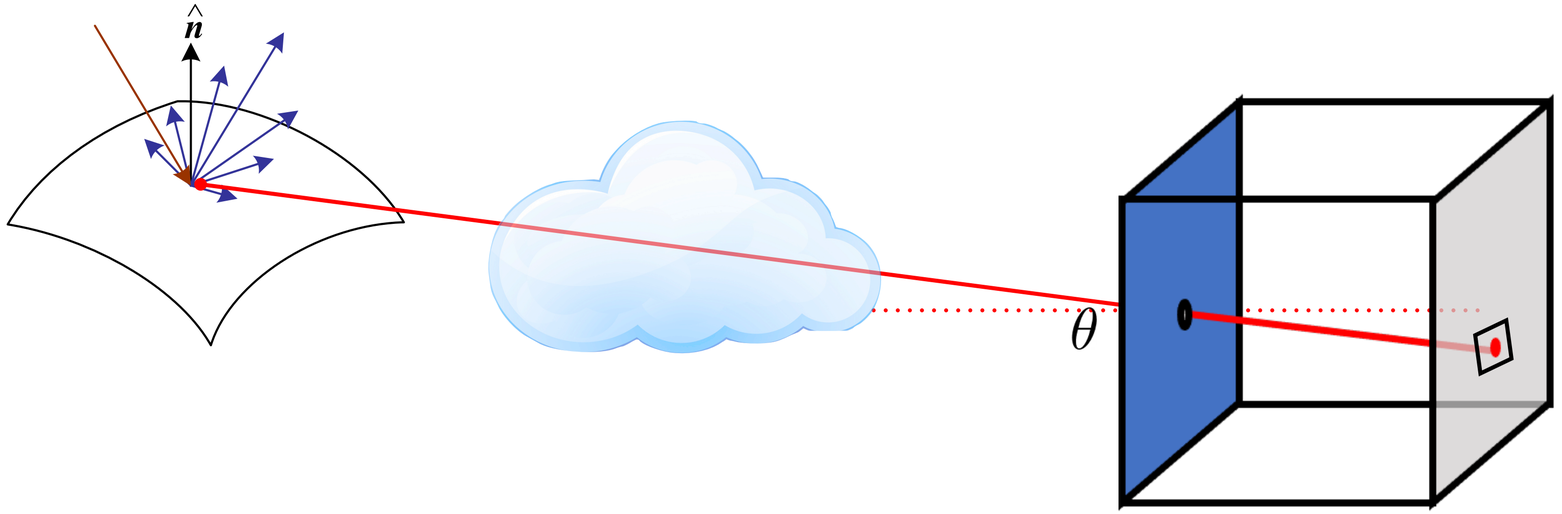
Pixel value → Emitted radiance at a **single** (surface) point

Beyond Surfaces

Not a valid assumption for fog, clouds, scattering, liquids etc.

source: Studio Lernert & Sander

Beyond Surfaces



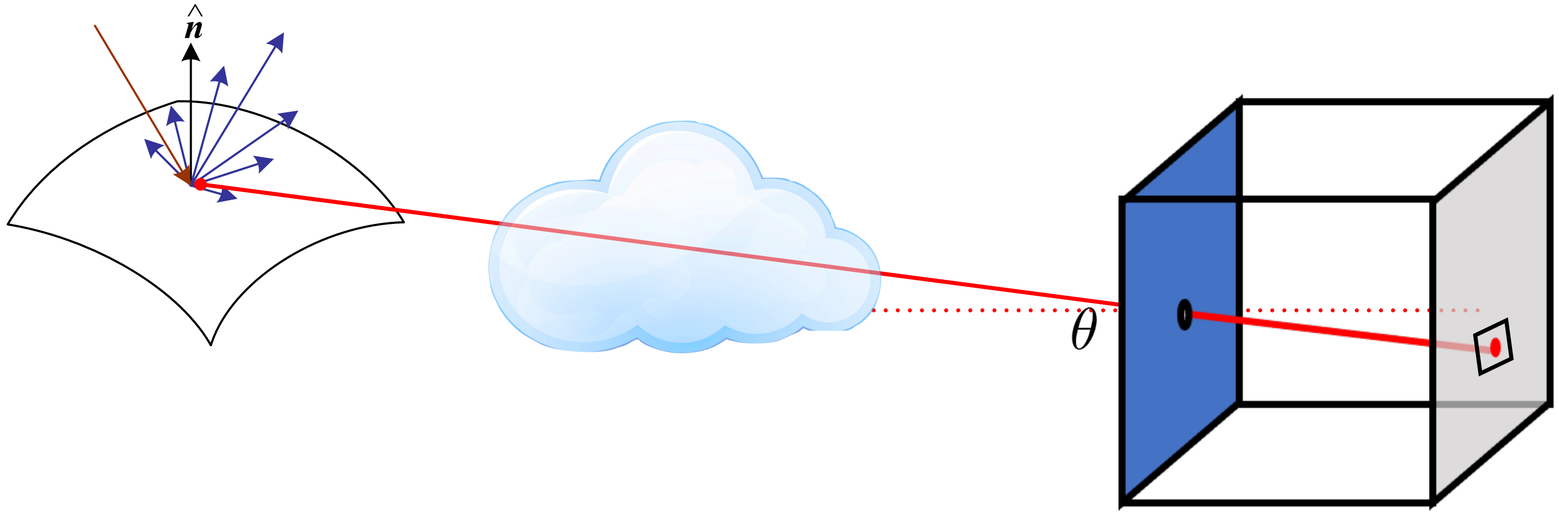
$$\propto L(\mathbf{x}^*, \omega)$$

radiance for: ***x**** = pixel sensor centre, *w* = direction from x to optical centre

$$\neq L(\mathbf{x}^* - \lambda\omega, \omega)$$

Pixel value **cannot** be reduced to emitted radiance by a single surface point

Volume Rendering



$$\propto L(\mathbf{x}^*, \omega)$$

radiance for: ***x**** = pixel sensor centre, *w* = direction from x to optical centre

How does L(x, w) vary along a ray?

Absorption

Light traveling in a medium can get absorbed by it

Slide adapted from: Ioannis (Yannis) Gkioulekas

Absorption

Light traveling in a medium can get absorbed by it

Slide adapted from: Ioannis (Yannis) Gkioulekas

Absorption

Thus, if one is to be five times as distant, make it five times bluer.

—Treatise on Painting, Leonardo Da Vinci, pp 295, circa 1480.

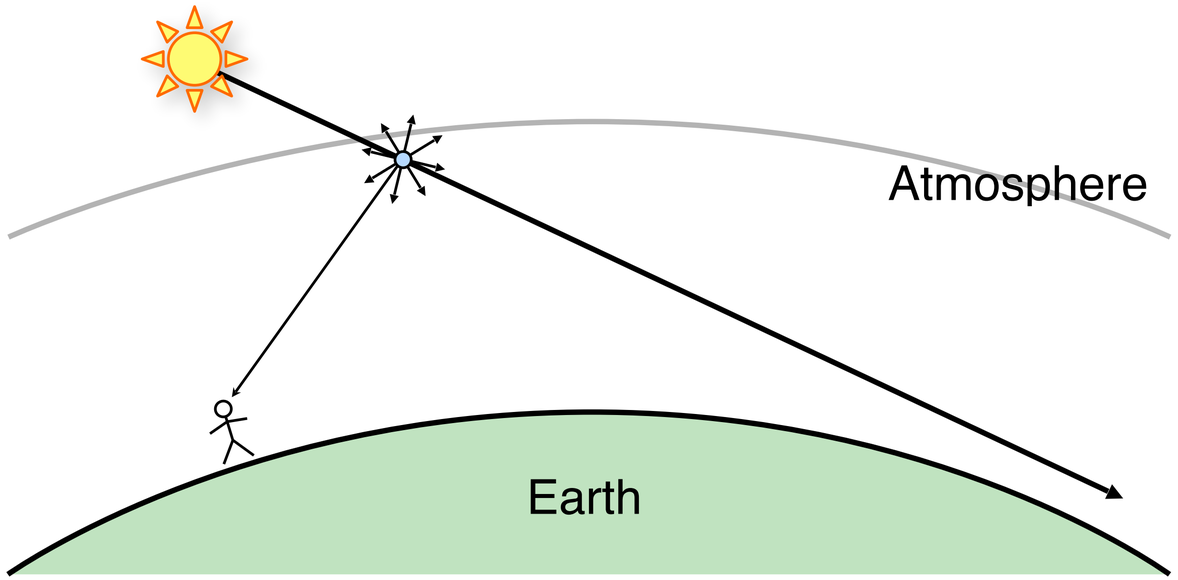
Slide credits: Ioannis (Yannis) Gkioulekas

Emission

The medium can emit light on its own

Slide adapted from: Ioannis (Yannis) Gkioulekas

Scattering

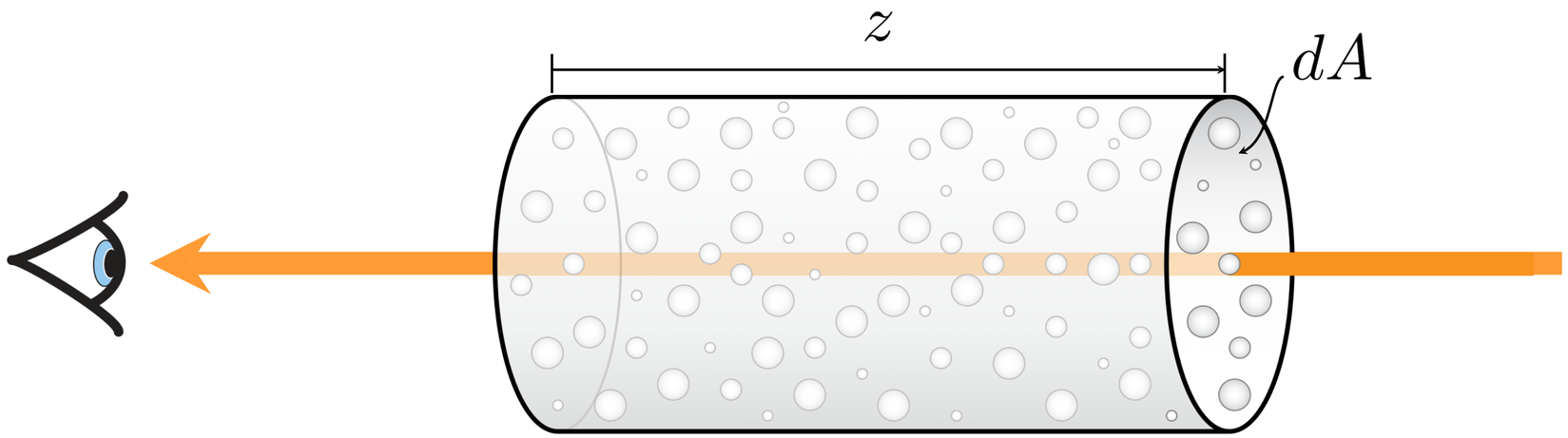


Light along a ray can depend on light in other directions

(we will **not** model scattering effects in today's lecture)

Slide adapted from: Ioannis (Yannis) Gkioulekas

Volume Rendering a Ray

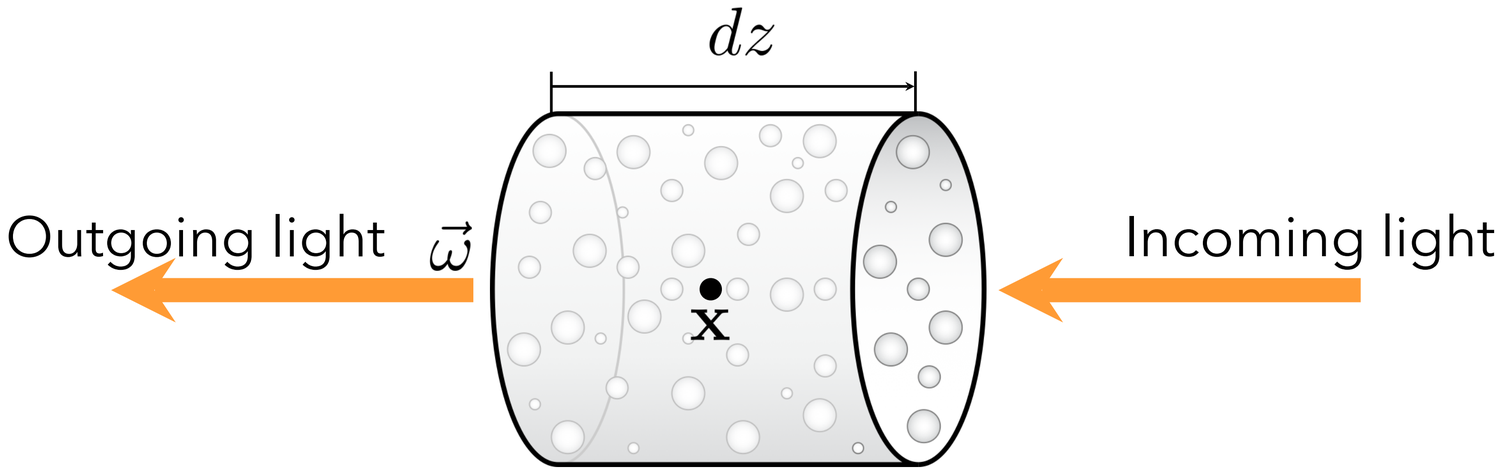


How much light is *lost/gained* along the differential beam due to interactions of light with the medium?

How does L(x, w) vary along a ray?

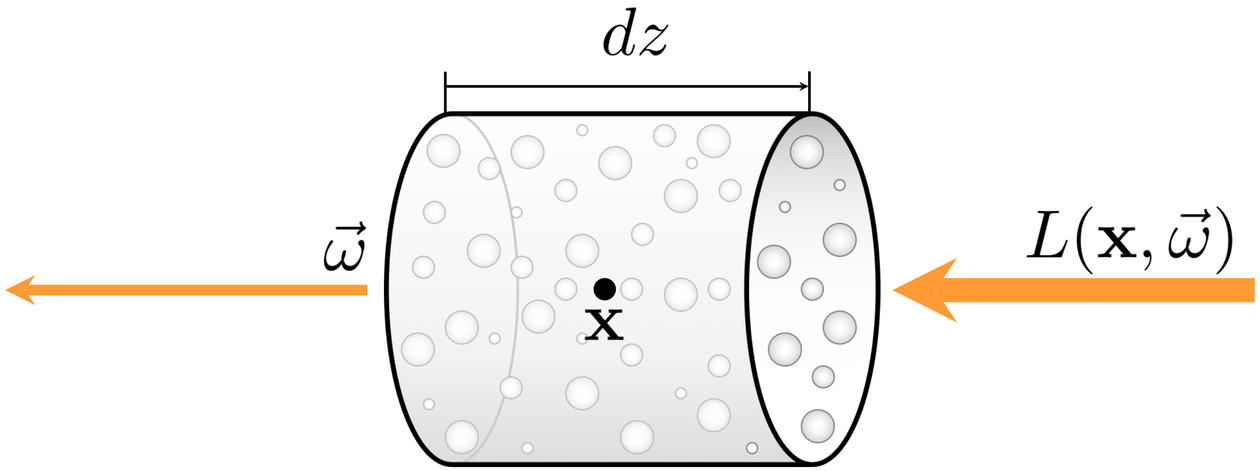
Slide credits: Ioannis (Yannis) Gkioulekas

Volume Rendering a Ray



Slide credits: Ioannis (Yannis) Gkioulekas

Volume Rendering a Ray: Absorption



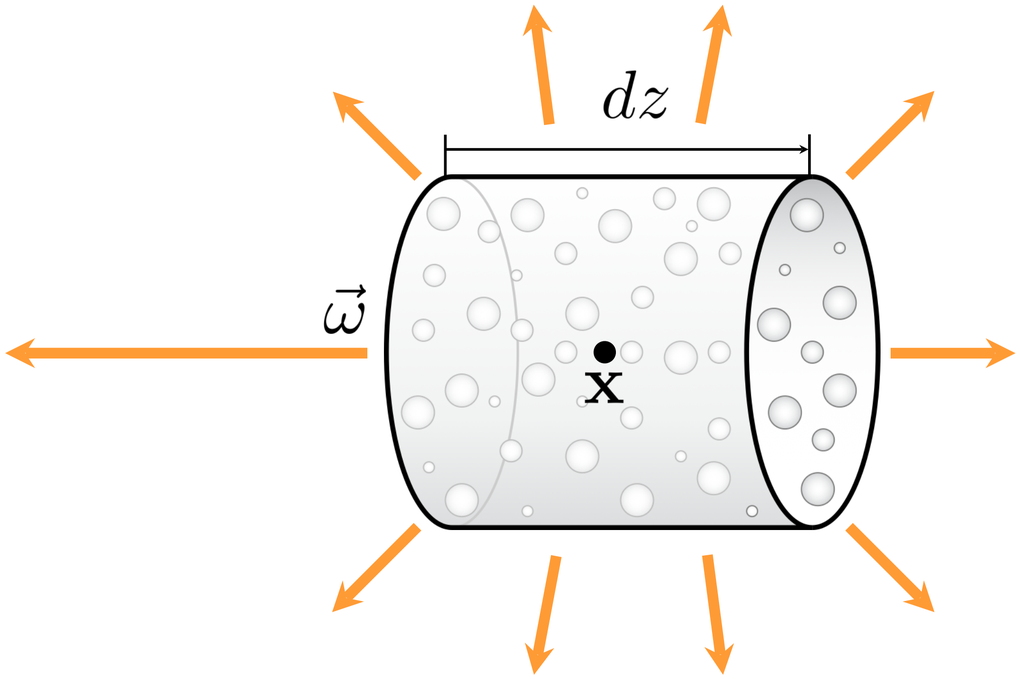
$$dL(\mathbf{x}, \vec{\omega}) = -\sigma_a(\mathbf{x}) L(\mathbf{x}, \vec{\omega}) dz$$

$\sigma_a(\mathbf{x})$: absorption coefficient

Q: Why should dL be **proportional** to L (incoming radiance)?

Slide credits: Ioannis (Yannis) Gkioulekas

Volume Rendering a Ray: Emission

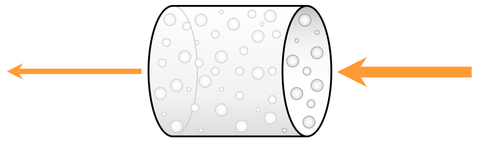


$$dL(\mathbf{x}, \vec{\omega}) = \sigma_a(\mathbf{x}) L_e(\mathbf{x}, \vec{\omega}) dz$$

$L_e(\mathbf{x}, \vec{\omega})$: emitted radiance

Slide credits: Ioannis (Yannis) Gkioulekas

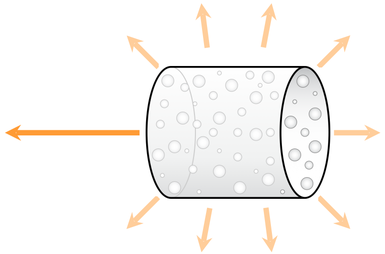
Volume Rendering: Emission-Absorption Model



Absorption

$$dL(\mathbf{x}, \vec{\omega}) = \boxed{-\sigma_a(\mathbf{x})L(\mathbf{x}, \vec{\omega})dz} \quad \text{Losses}$$

$$\boxed{+\sigma_a(\mathbf{x})L_e(\mathbf{x}, \vec{\omega})dz} \quad \text{Gains}$$



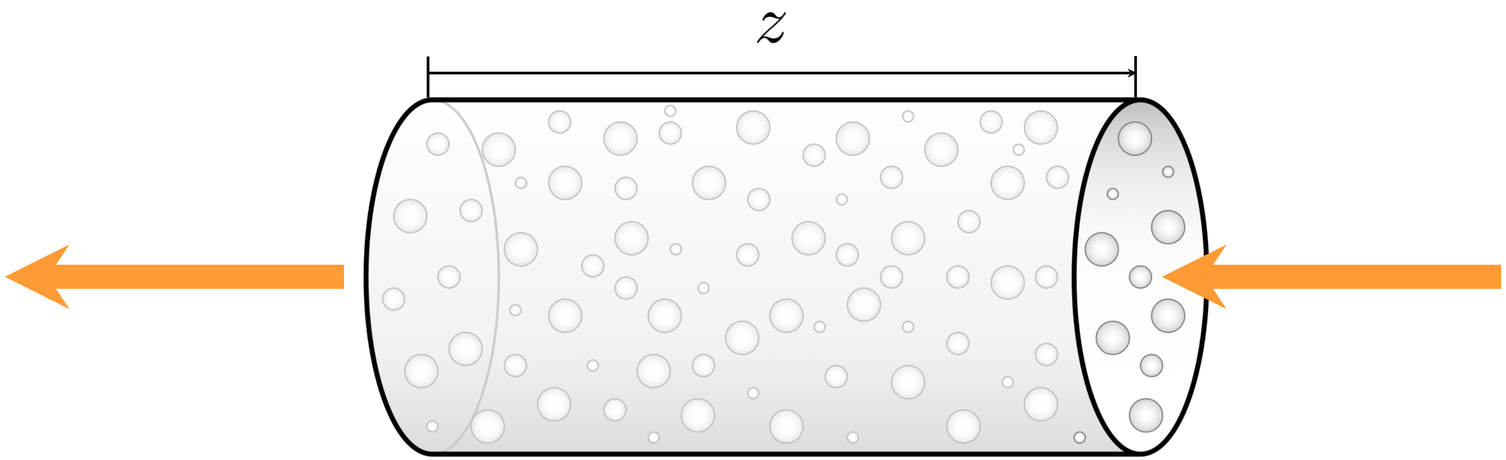
Emission

Will drop the subscript moving forward

$$\sigma_a \equiv \sigma$$

Slide credits: Ioannis (Yannis) Gkioulekas

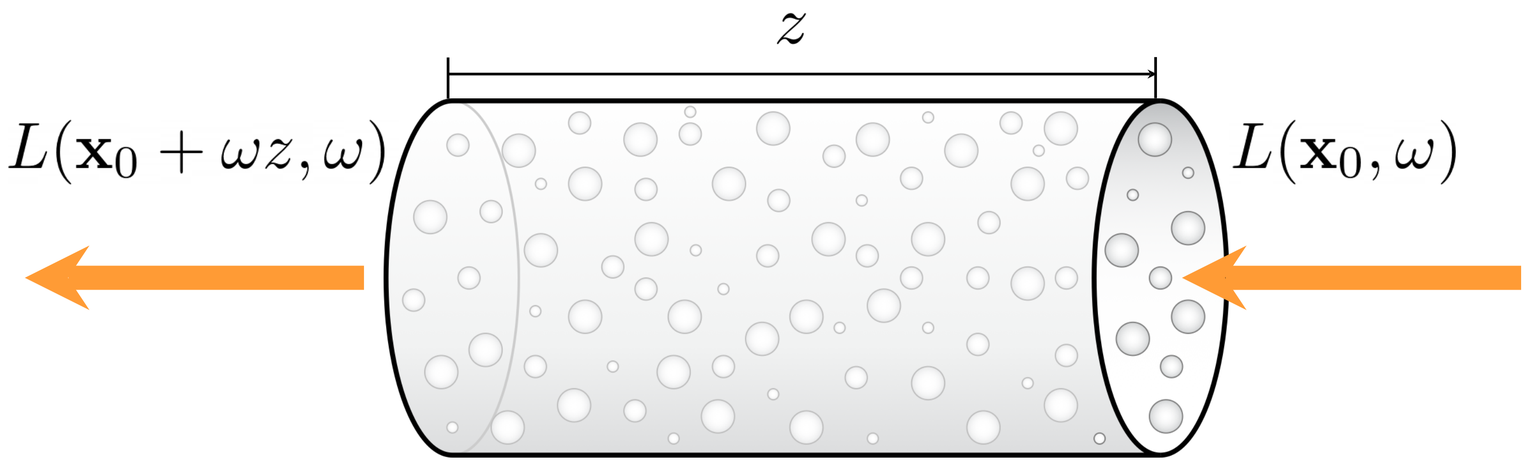
From infinitesimal to finite lengths



1. **Model absorption-only case**
2. **Allow emission**

Slide adapted from: Ioannis (Yannis) Gkioulekas

Absorption-only Volume Rendering



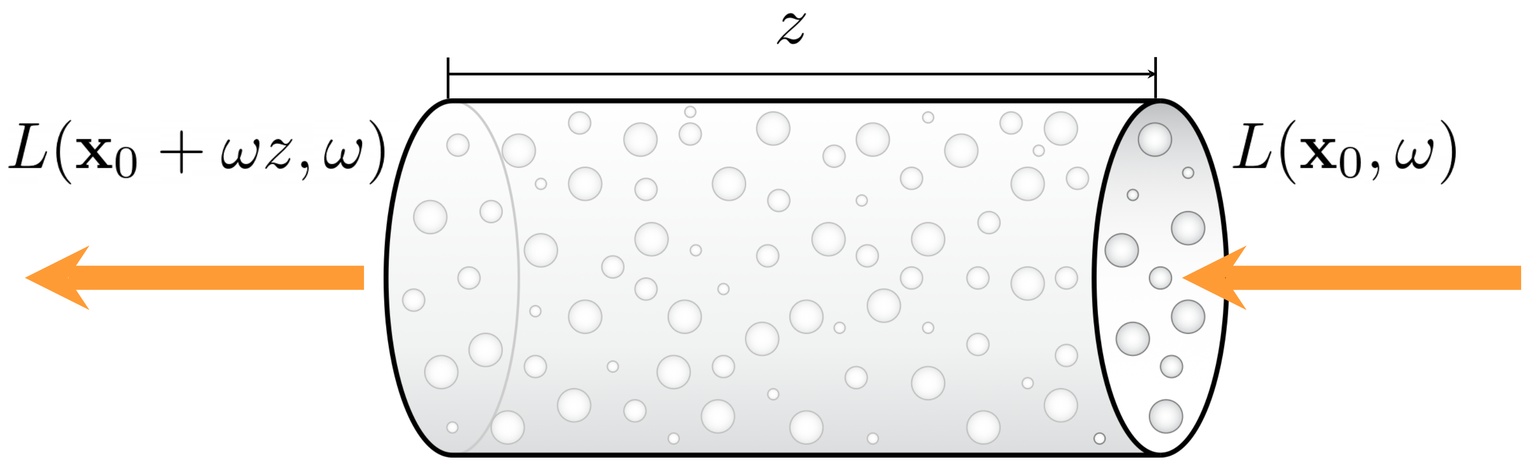
Q: What if we have a homogenous medium? (uniform coefficient)

$$dL(\mathbf{x}, \omega) = -\sigma L(\mathbf{x}, \omega) dz$$

$$L(\mathbf{x}_0 + \omega z, \omega) = e^{-\sigma z} L(\mathbf{x}_0, \omega)$$

Slide adapted from: Ioannis (Yannis) Gkioulekas

Absorption-only Volume Rendering



Q: What if we have a non-homogenous medium?

$$dL(\mathbf{x}, \omega) = -\sigma(\mathbf{x})L(\mathbf{x}, \omega)dz$$

$$L(\mathbf{x}_0 + \omega z, \omega) = e^{-\int_{t=0}^{z} \sigma(\mathbf{x}+\omega\mathbf{t})dt} L(\mathbf{x}_0, \omega)$$

Transmittance

Slide adapted from: Ioannis (Yannis) Gkioulekas

Transmittance

$T(\mathbf{x}, \mathbf{y})$

What fraction of radiance at **x** in direction of **y**, reaches **y** ?
(along a straight line under absorption-only model)

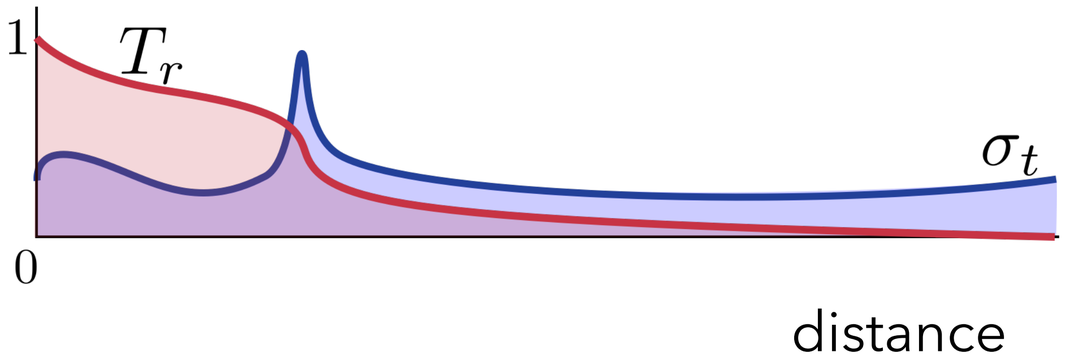
Q: should this be symmetric?

Homogenous Medium:

$$e^{-\sigma\|\mathbf{x}-\mathbf{y}\|}$$

Non-Homogenous Medium:

$$e^{-\int_{t=0}^{\|\mathbf{x}-\mathbf{y}\|}\sigma(\mathbf{x}+\omega\mathbf{t})}$$



Slide adapted from: Ioannis (Yannis) Gkioulekas

Transmittance

$T(\mathbf{x}, \mathbf{y})$

What fraction of radiance at **x** in direction of **y**, reaches **y** ?
(along a straight line under absorption-only model)

Homogenous Medium:

$$e^{-\sigma\|\mathbf{x}-\mathbf{y}\|}$$

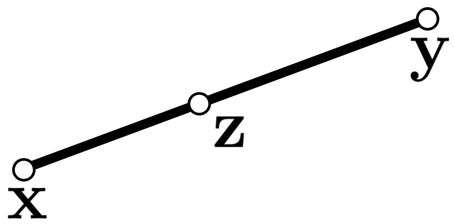
Non-Homogenous Medium:

$$e^{-\int_{t=0}^{\|\mathbf{x}-\mathbf{y}\|}\sigma(\mathbf{x}+\omega\mathbf{t})}$$

Multiplicativity:

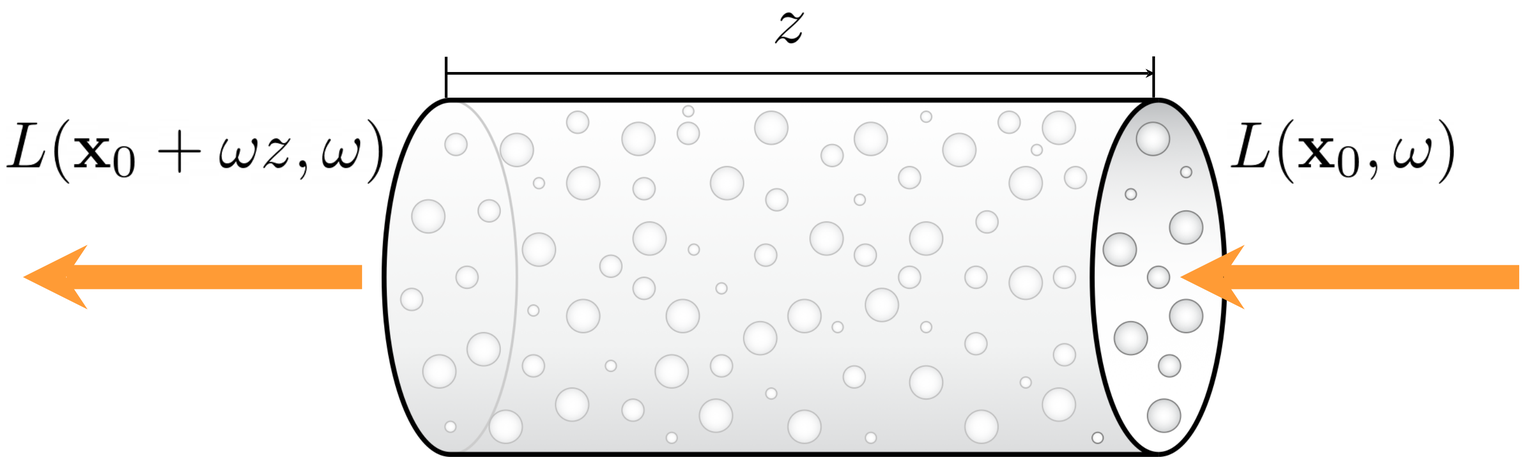
$$T(\mathbf{x}, \mathbf{y}) = T(\mathbf{x}, \mathbf{z})T(\mathbf{z}, \mathbf{y})$$

Why?



Slide adapted from: Ioannis (Yannis) Gkioulekas

Absorption-only Volume Rendering



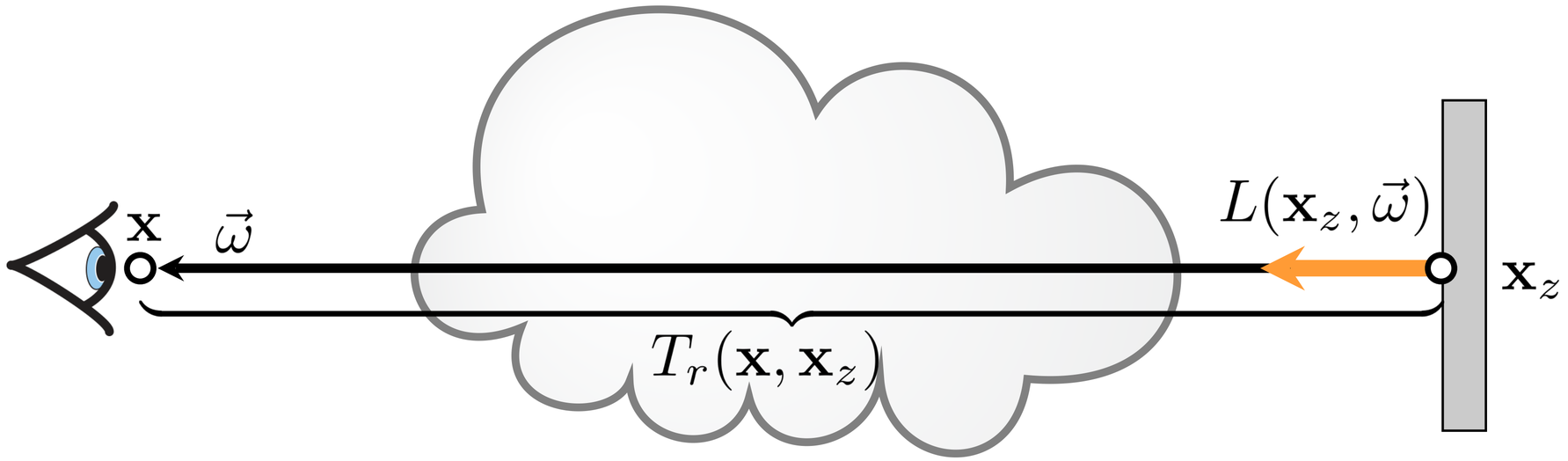
$$L(\mathbf{x}_0 + \omega z, \omega) = T(\mathbf{x}_0, \mathbf{x}_0 + \omega z) L(\mathbf{x}_0, \omega)$$

Slide adapted from: Ioannis (Yannis) Gkioulekas

Absorption-only Volume Rendering

Slide adapted from: Ioannis (Yannis) Gkioulekas

Absorption-only Volume Rendering

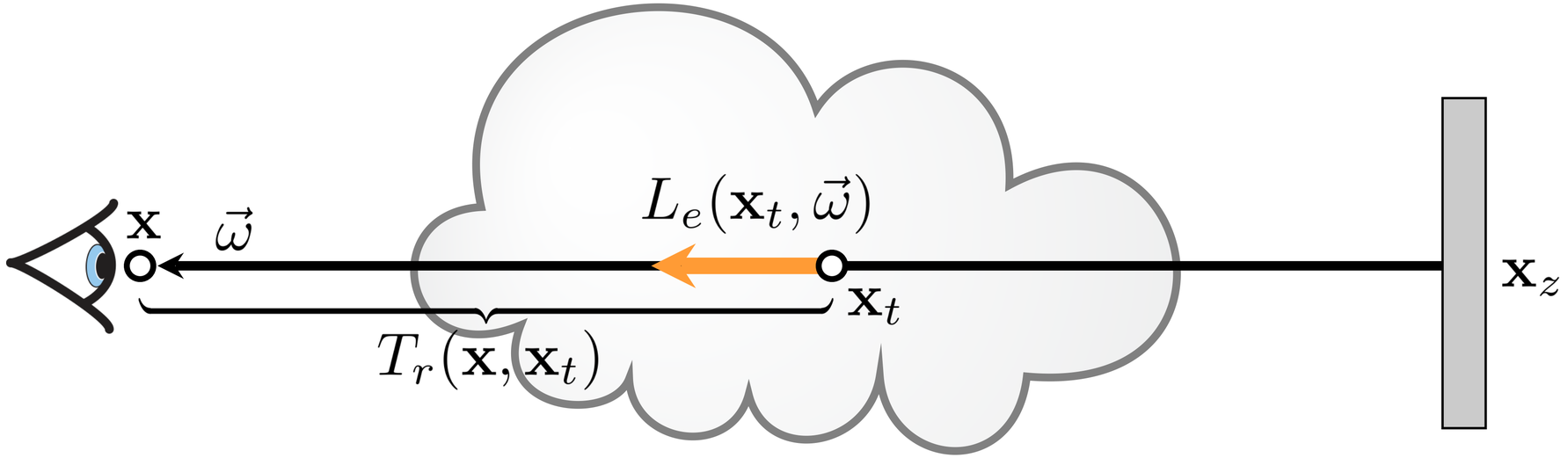


$$L(\mathbf{x}, \omega) = T(\mathbf{x}, \mathbf{x}_z) L(\mathbf{x}_z, \omega)$$

Radiance from ‘outside’ the medium

Slide adapted from: Ioannis (Yannis) Gkioulekas

Emission-Absorption Volume Rendering



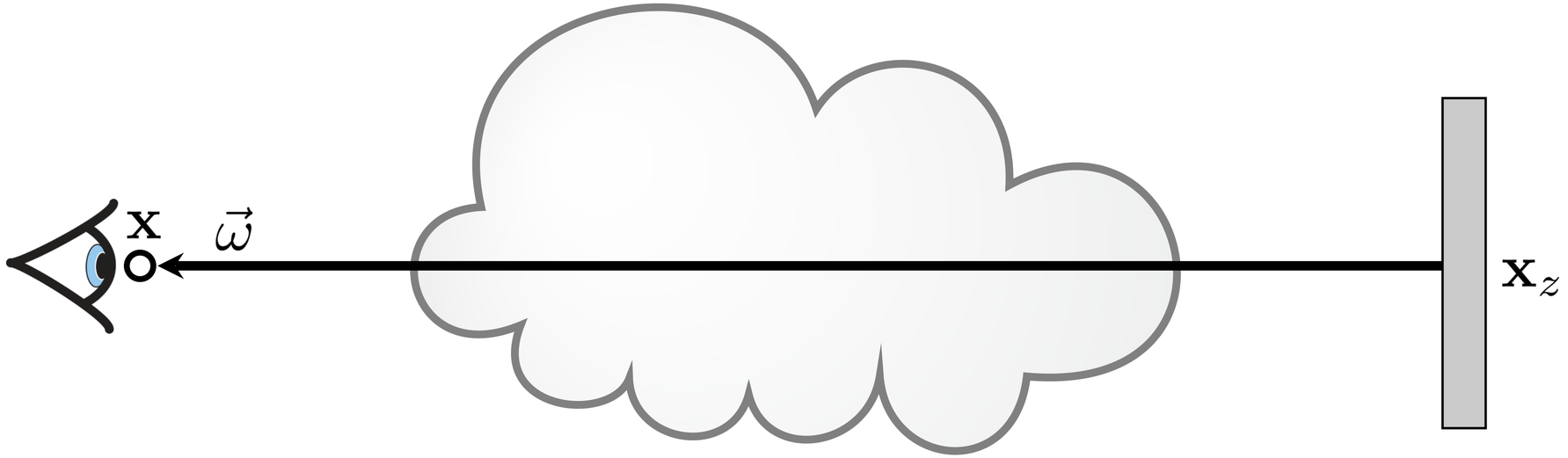
$$L(\mathbf{x}, \omega) = T(\mathbf{x}, \mathbf{x}_z) L(\mathbf{x}_z, \omega)$$

$$+ \int_0^z T(\mathbf{x}, \mathbf{x}_t) \sigma(\mathbf{x}_t) L_e(\mathbf{x}_t, \omega) dt$$

Accumulated Emitted Radiance from inside

Slide adapted from: Ioannis (Yannis) Gkioulekas

Emission-Absorption Volume Rendering

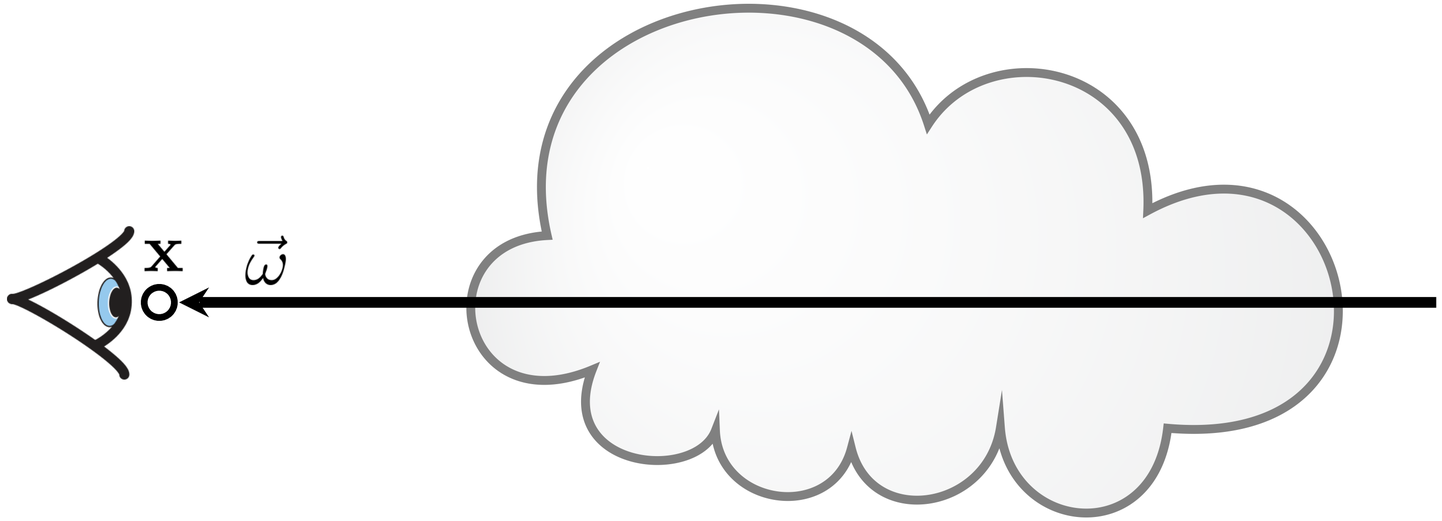


$$L(\mathbf{x}, \omega) = T(\mathbf{x}, \mathbf{x}_z) L(\mathbf{x}_z, \omega) + \int_0^z T(\mathbf{x}, \mathbf{x}_t) \sigma(\mathbf{x}_t) L_e(\mathbf{x}_t, \omega) dt$$

Can we compute this analytically? Approximately?

Slide adapted from: Ioannis (Yannis) Gkioulekas

Computational Volume Rendering

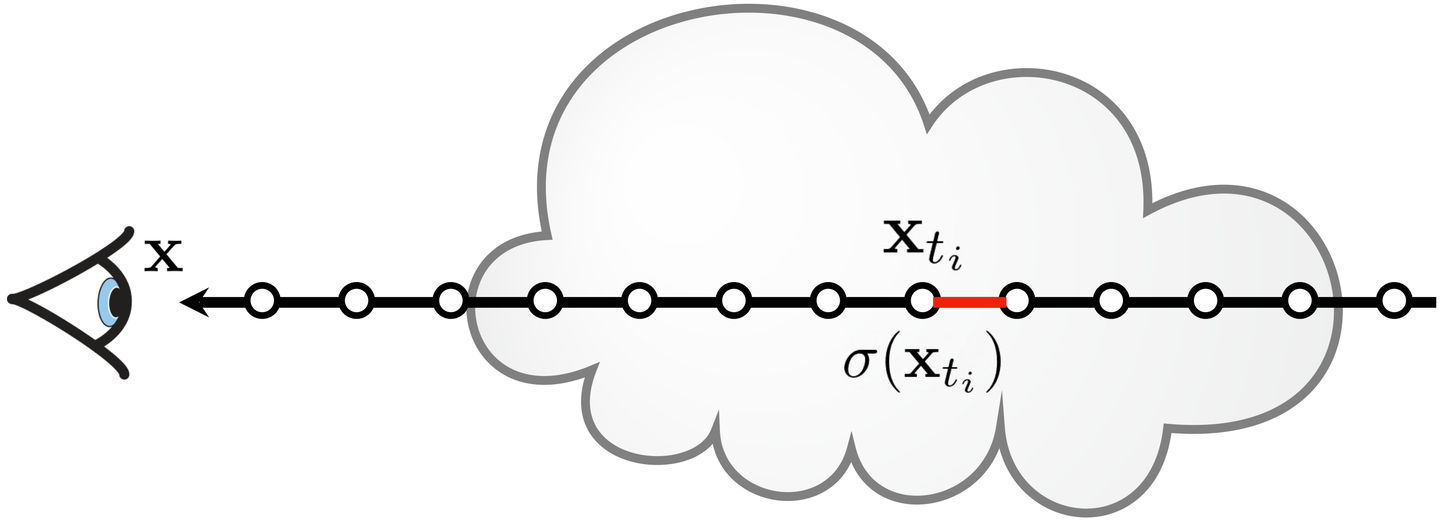


$$L(\mathbf{x}, \omega) = \int_0^z T(\mathbf{x}, \mathbf{x}_t)\sigma(\mathbf{x}_t)L_e(\mathbf{x}_t, \omega)dt$$

Let's ignore light from outside medium (for now)

Slide adapted from: Ioannis (Yannis) Gkioulekas

Computational Volume Rendering: Ray Marching



$$L(\mathbf{x}, \omega) = \sum_{i=1}^{N} \text{(contribution from } i^{th} \text{ segment)}$$

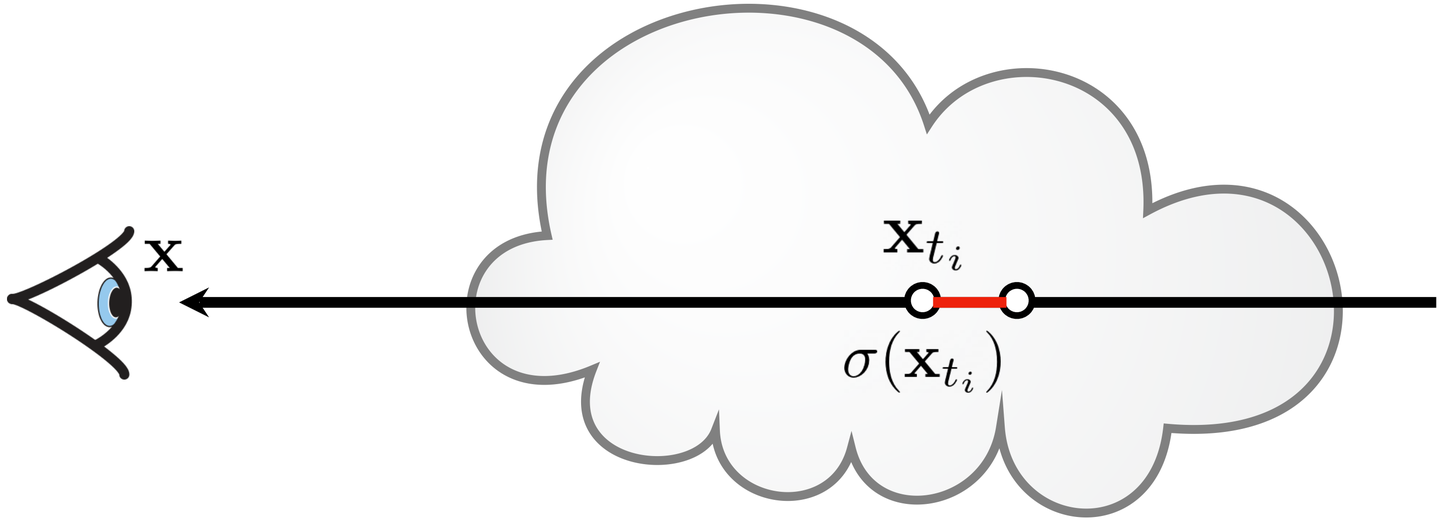
Approximate with a discrete sum

$\mathbf{x}_{t_i}$: i^{th} sample along ray at depth t_i

Δt : distance between successive samples

Slide adapted from: Ioannis (Yannis) Gkioulekas

Computational Volume Rendering: Ray Marching

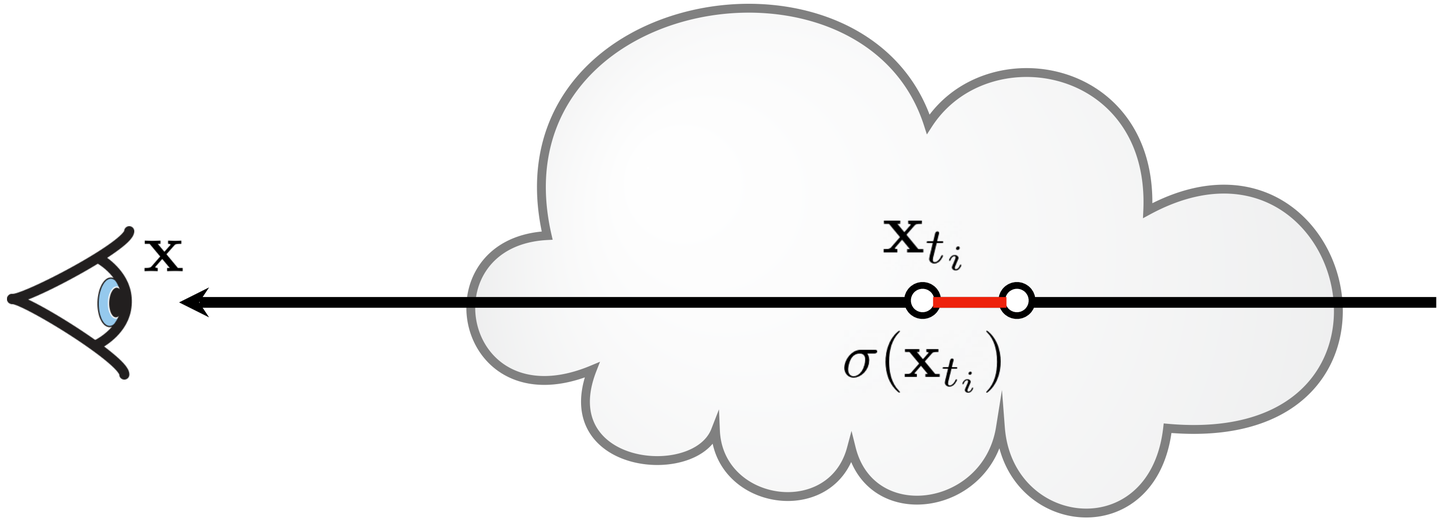


$$L(\mathbf{x}, \omega) = \sum_{i=1}^{N} \text{(contribution from } i^{th} \text{ segment)}$$

$\mathbf{x}_{t_i}$: i^{th} sample along ray at depth t_i

Δt : distance between successive samples

Computational Volume Rendering: Ray Marching

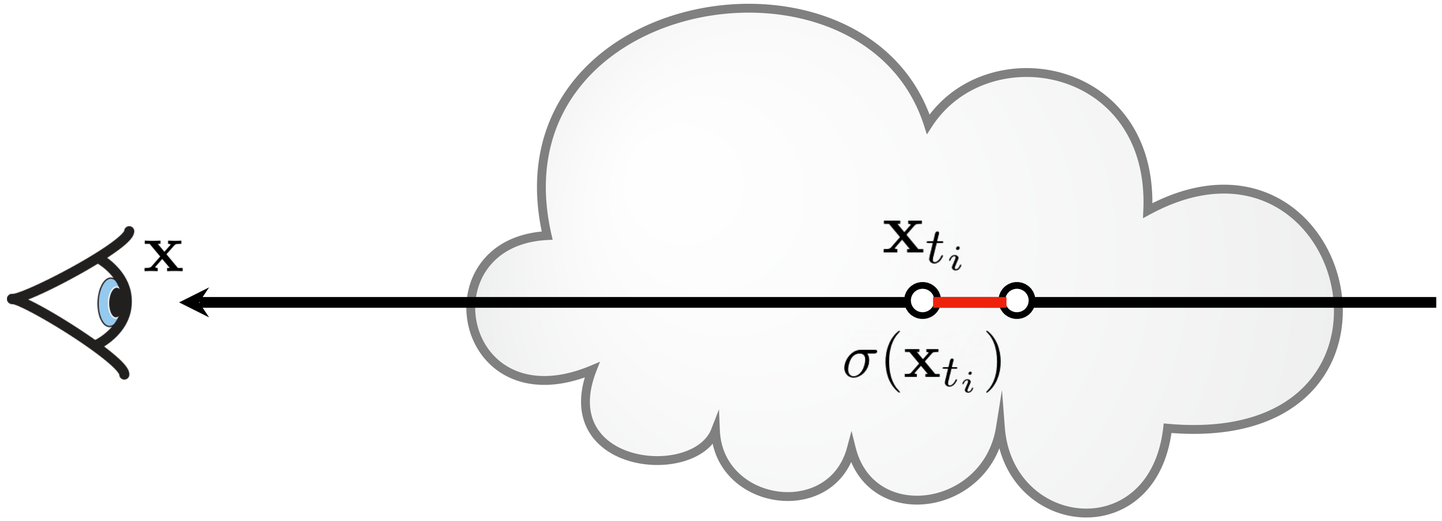


$$L(\mathbf{x}, \omega) = \sum_{i=1}^{N} T(\mathbf{x}, \mathbf{x}_{t_i}) \text{ x (emission from i}^{\text{th}} \text{ segment)}$$

$\mathbf{x}_{t_i}$: i$^{\text{th}}$ sample along ray at depth t_i

Δt : distance between successive samples

Computational Volume Rendering: Ray Marching



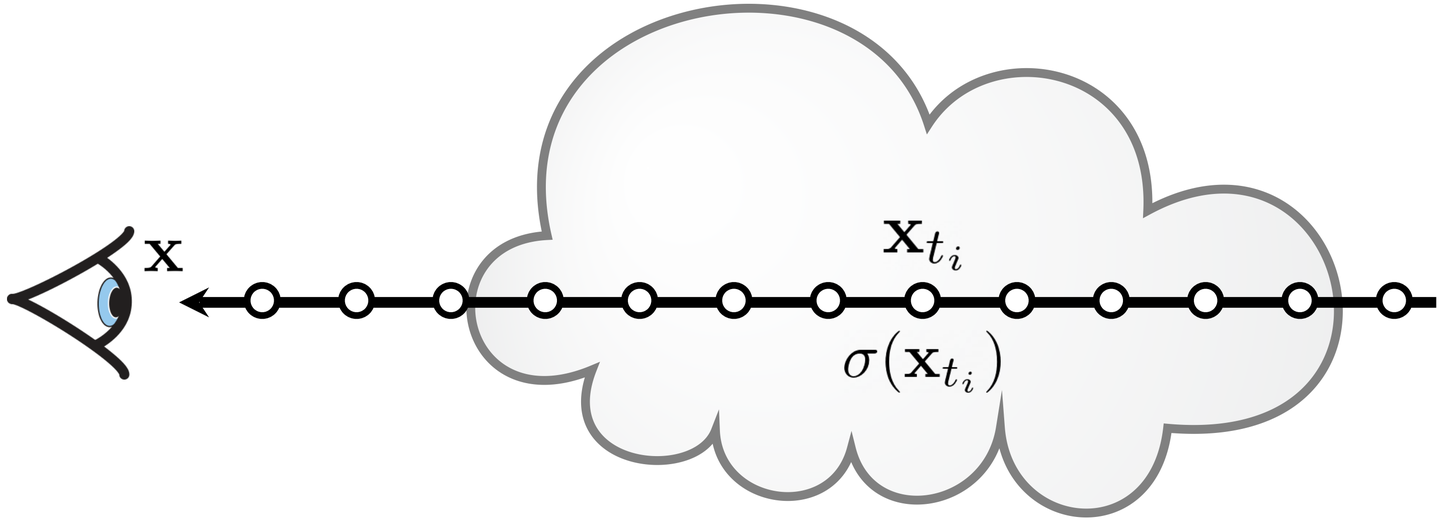
$$L(\mathbf{x},\omega) = \sum_{i=1}^{N} T(\mathbf{x},\mathbf{x}_{t_i})(1-e^{-\sigma_{t_i}\Delta t})L_e(\mathbf{x}_{t_i},\omega)$$

Requires a longer derivation which we'll skip

$\mathbf{x}_{t_i}$: i[th] sample along ray at depth t_i

Δt : distance between successive samples

Computational Volume Rendering: Ray Marching



$$L(\mathbf{x},\omega) = \sum_{i=1}^{N} T(\mathbf{x},\mathbf{x}_{t_i})(1-e^{-\sigma_{t_i}\Delta t})L_e(\mathbf{x}_{t_i},\omega)$$

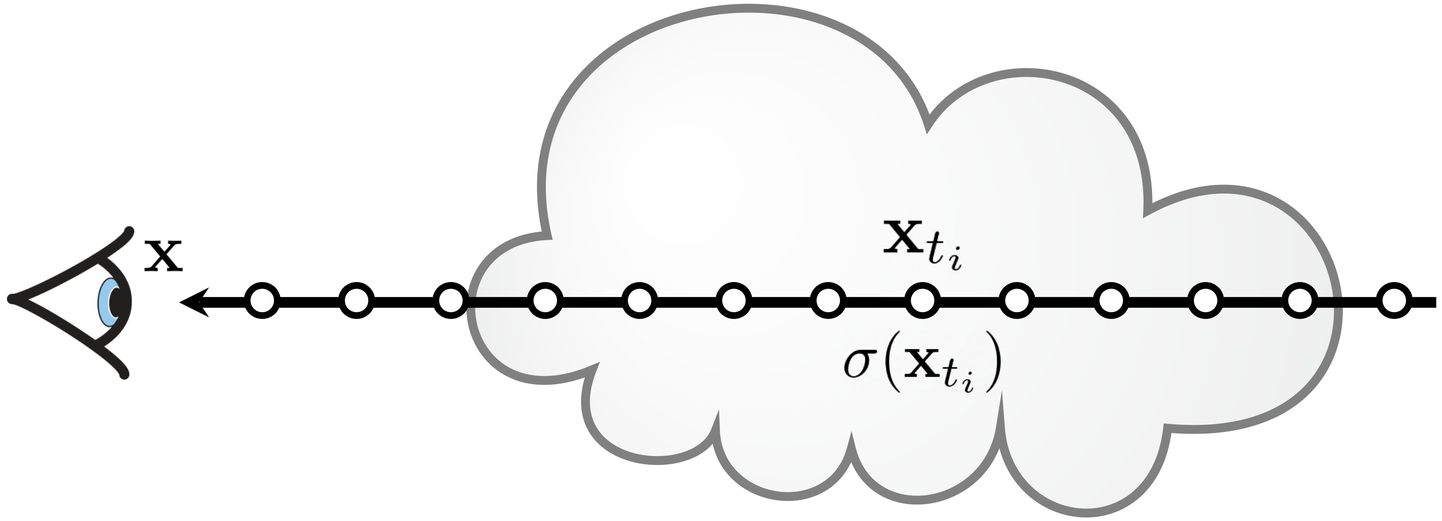
Multiplicativity

Base case: $T(\mathbf{x},\mathbf{x}_{t_1}) = 1$

$$T(\mathbf{x},\mathbf{x}_{t_i}) = T(\mathbf{x},\mathbf{x}_{t_{i-1}})e^{-\sigma_{t_{i-1}}\Delta t}$$

Assume constant coefficient between samples

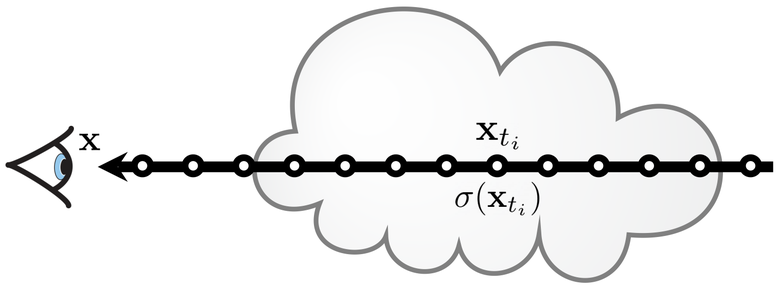
Computational Volume Rendering: Ray Marching



1. Draw uniform samples along a ray (N segments, or N+1 points)
2. Compute transmittance between camera and each sample
3. Aggregate contributions across segments to get overall radiance (color)

(minor modifications can allow non-uniform samples)

Computational Volume Rendering: A summary



$$\sigma_{t_i} \equiv \sigma(\mathbf{x}_{t_i})$$

$$L_e(\mathbf{x}_{t_i}, \omega)$$

$$\longrightarrow \quad L(\mathbf{x}, \omega)$$

If we can compute:

a) (per-point) density

b) (per-point, direction) emitted light,

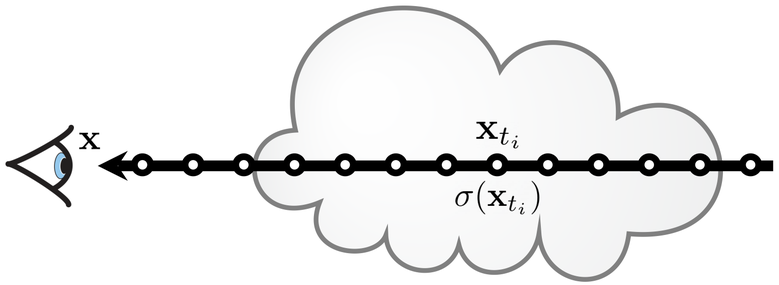
we can render **any** ray through the medium

Equivalently, we can render an image from any camera viewpoint (using H*W rays)

Note: **Differentiable** process w.r.t. the **density**, **emitted light**

and also camera parameters if density, emission are differentiable functions of position, direction

Volumes: Rendering and Representation

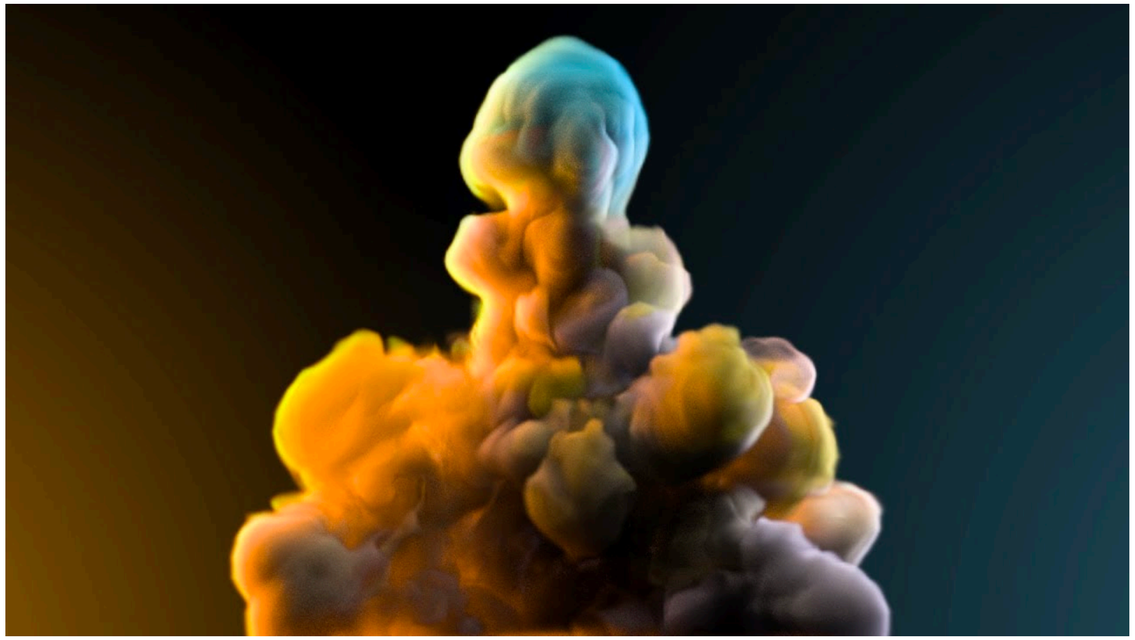


$$\sigma_{t_i} \equiv \sigma(\mathbf{x}_{t_i})$$

$$L_e(\mathbf{x}_{t_i}, \omega)$$

$$\longrightarrow \quad L(\mathbf{x}, \omega)$$

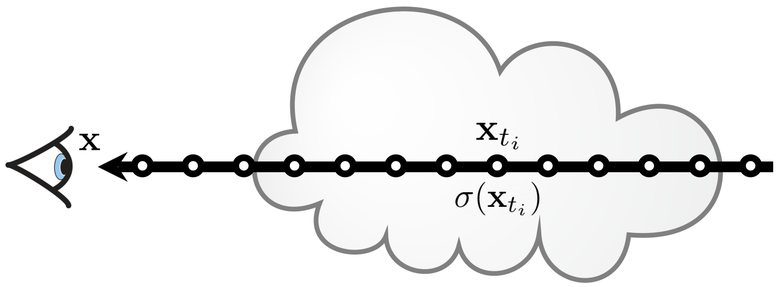
Rendering Algorithm



How to represent volumes?

(such that we can compute pointwise density and emitted light)

Volumes: Rendering and Representation

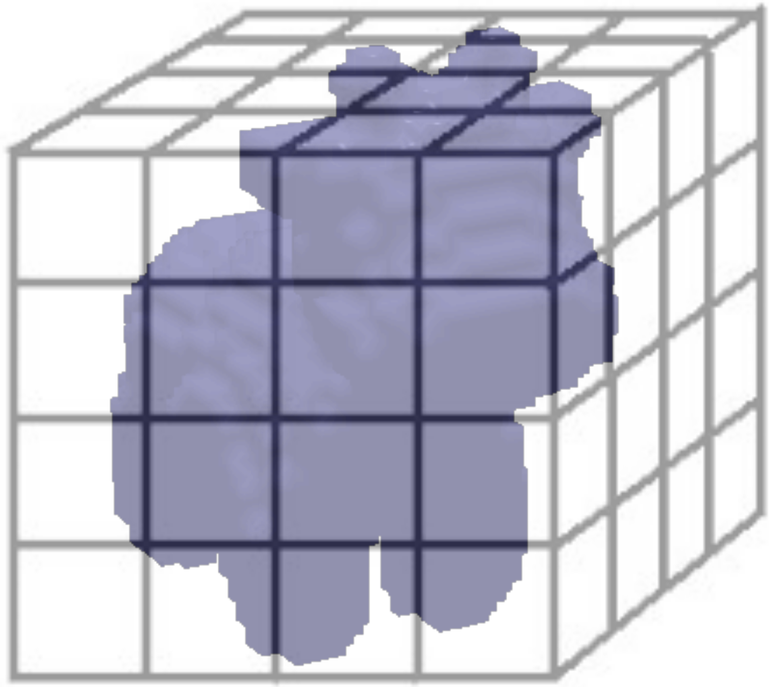


$$\sigma_{t_i} \equiv \sigma(\mathbf{x}_{t_i})$$

$$L_e(\mathbf{x}_{t_i}, \omega)$$

$$\longrightarrow \quad L(\mathbf{x}, \omega)$$

Rendering Algorithm



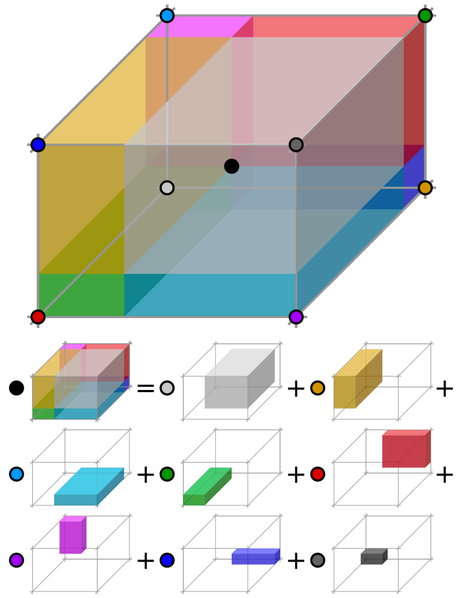
Option 1: A grid

Grid Representation

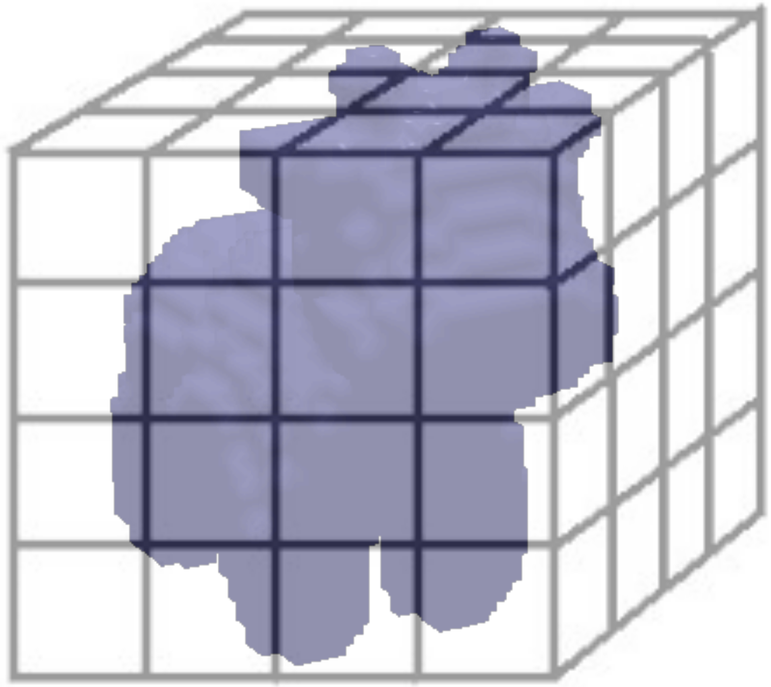
$$\sigma[x, y, z] \in \mathbb{R}^+$$

Store a positive scalar in each voxel (representing density)

Q: How to obtain $\sigma(\mathbf{x})$ for arbitrary **x** ?



Trilinear Interpolation



Grid Representation

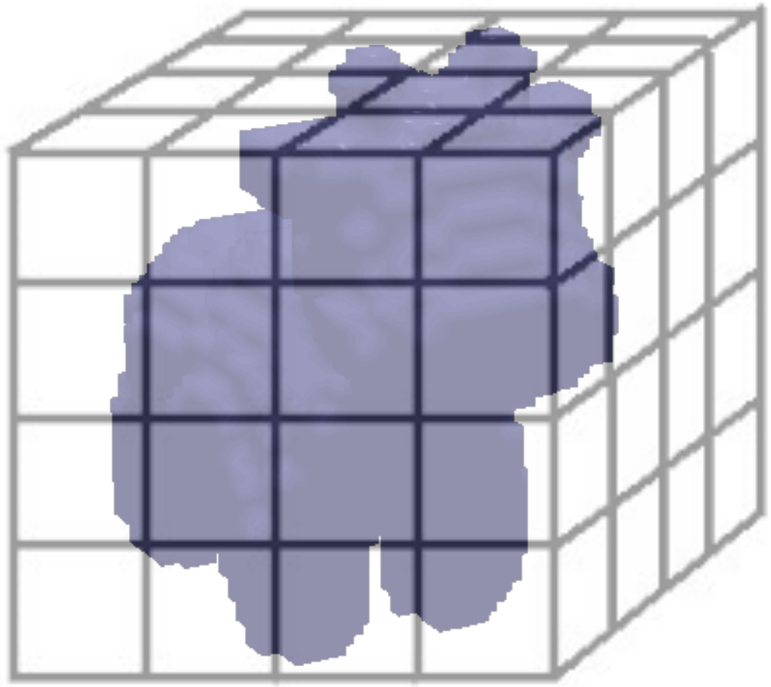
$$\sigma[x, y, z] \in \mathbb{R}^+$$

Store a positive scalar in each voxel (representing density)

Q: How to model view-dependent emission? $L_e(\mathbf{x}, \omega)$

A: Ignore view-dependence in emission

$$L_e(\mathbf{x}, \omega) \equiv c(\mathbf{x})$$



Grid Representation

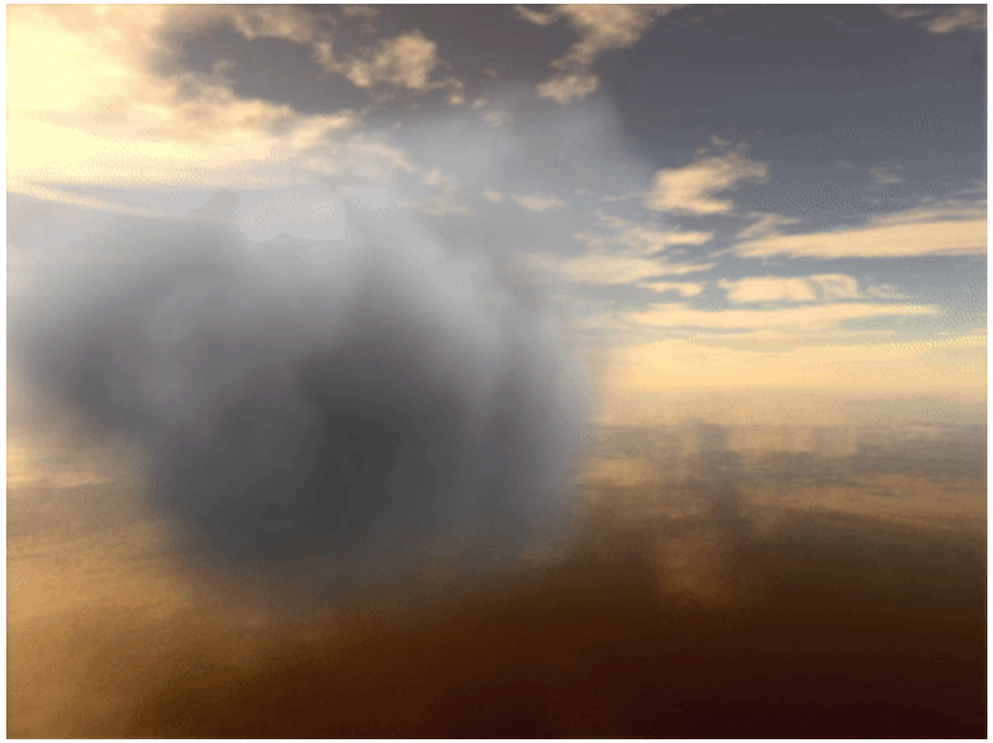
$$\sigma[x, y, z] \in \mathbb{R}^+$$

Store a positive scalar in each voxel (representing density)

$$c[x, y, z] \in [0, 1]^3$$

Store a color in each voxel (representing emitted light)

Grid Representation



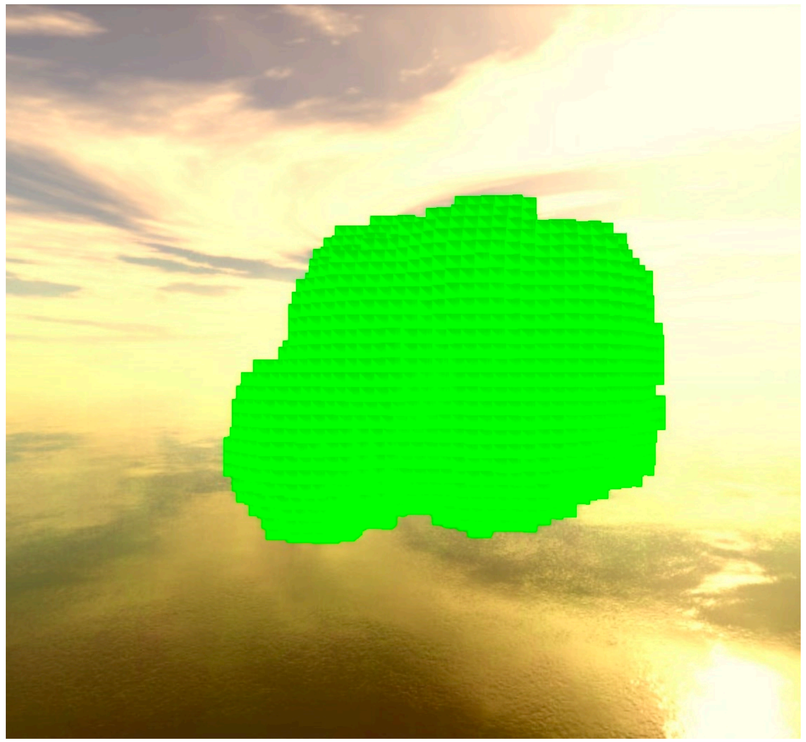


Image credits: https://geigersgit.github.io/clouds.html

Grid Representation

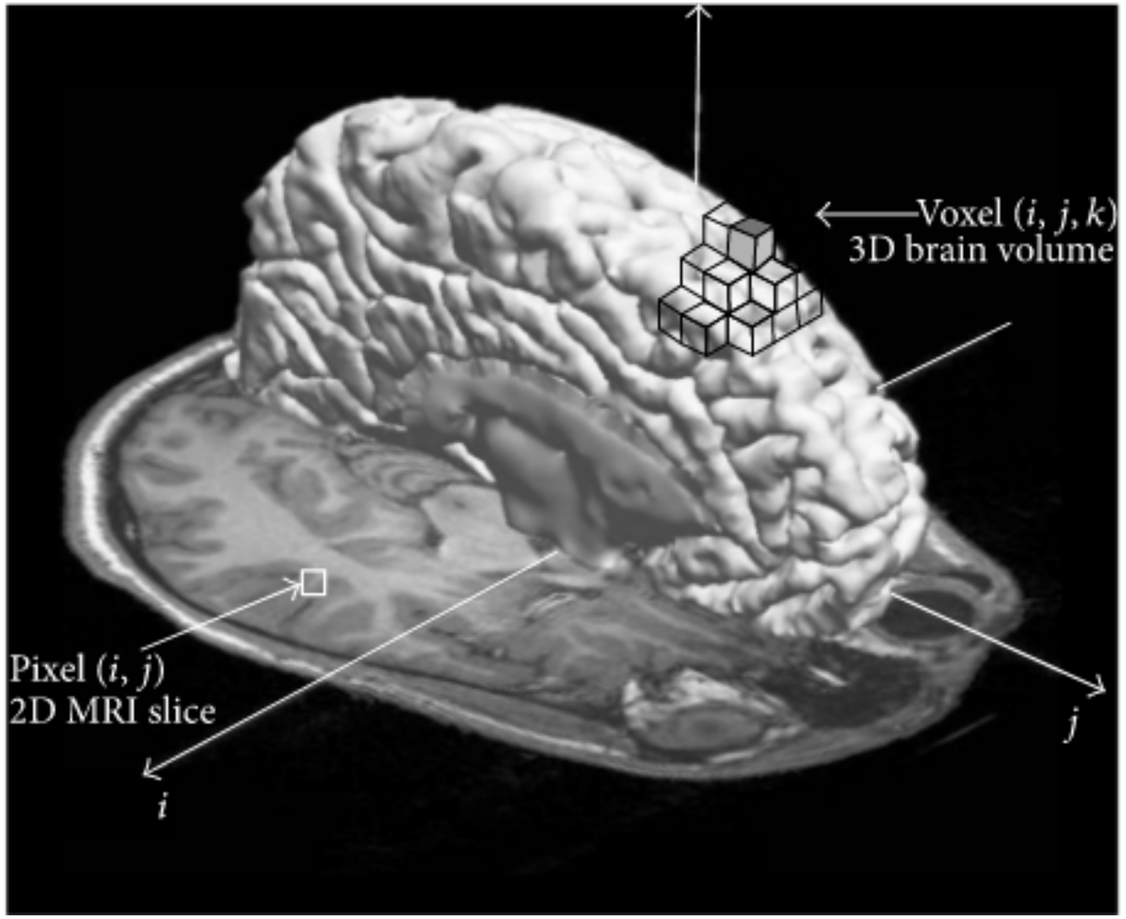


Image credits: https://www.hindawi.com/journals/cmmm/2015/450341/

Grid Representation

$$\sigma[x, y, z] \in \mathbb{R}^+$$

Store a positive scalar in each voxel (representing density)

$$c[x, y, z] \in [0, 1]^3$$

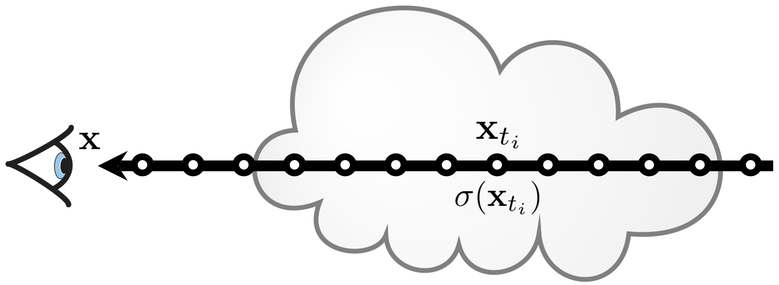
Store a color in each voxel (representing emitted light)

Cannot model view-dependent effects

Cubic growth of grid size with scene dimension

Video credits: NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.

Volumes: Rendering and Representation



$$\sigma_{t_i} \equiv \sigma(\mathbf{x}_{t_i})$$

$$L_e(\mathbf{x}_{t_i}, \omega)$$

$$\longrightarrow \quad L(\mathbf{x}, \omega)$$

Rendering Algorithm

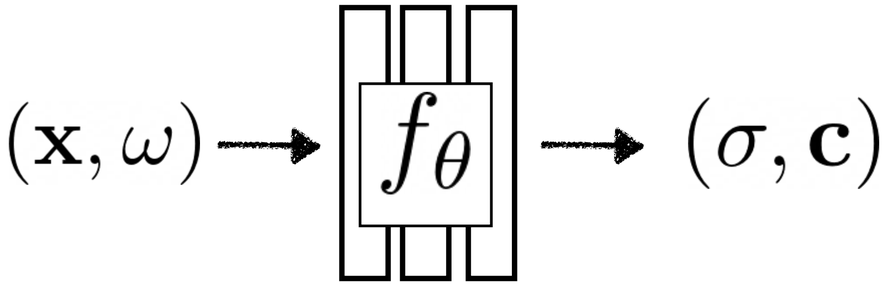
$$(\mathbf{x}, \omega) \longrightarrow \boxed{f_\theta} \longrightarrow (\sigma, \mathbf{c})$$

$$\sigma \in \mathbb{R}^+ \qquad \mathbf{c} \in [0,1]^3$$

Ensure with designing MLP that density only depends on **x**

Option 2: An MLP

Neural Radiance Fields



$$\sigma \in \mathbb{R}^+$$

$$\mathbf{c} \in [0,1]^3$$

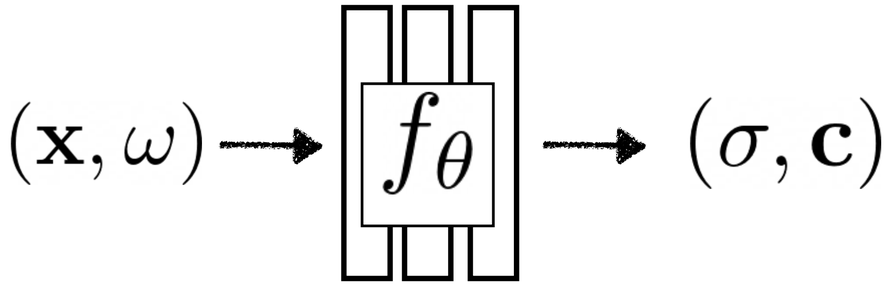
A scene is represented as an MLP that:

1) Given an input point, predicts density

2) Given a point and viewing direction, predicts radiance (color)

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.

Neural Radiance Fields



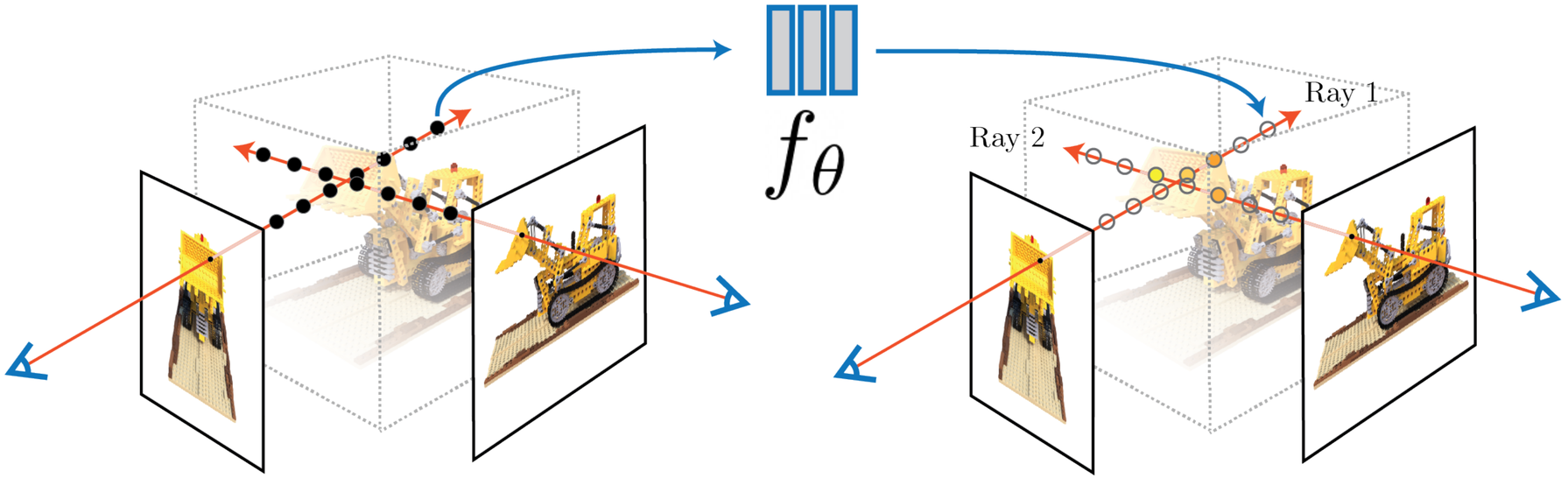
How to train this MLP?

Key idea: Learn a 3D representation consistent with observed views

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.

Learning Neural Radiance Fields

$$(\mathbf{x}, \omega) \rightarrow f_\theta \rightarrow (\sigma, \mathbf{c})$$



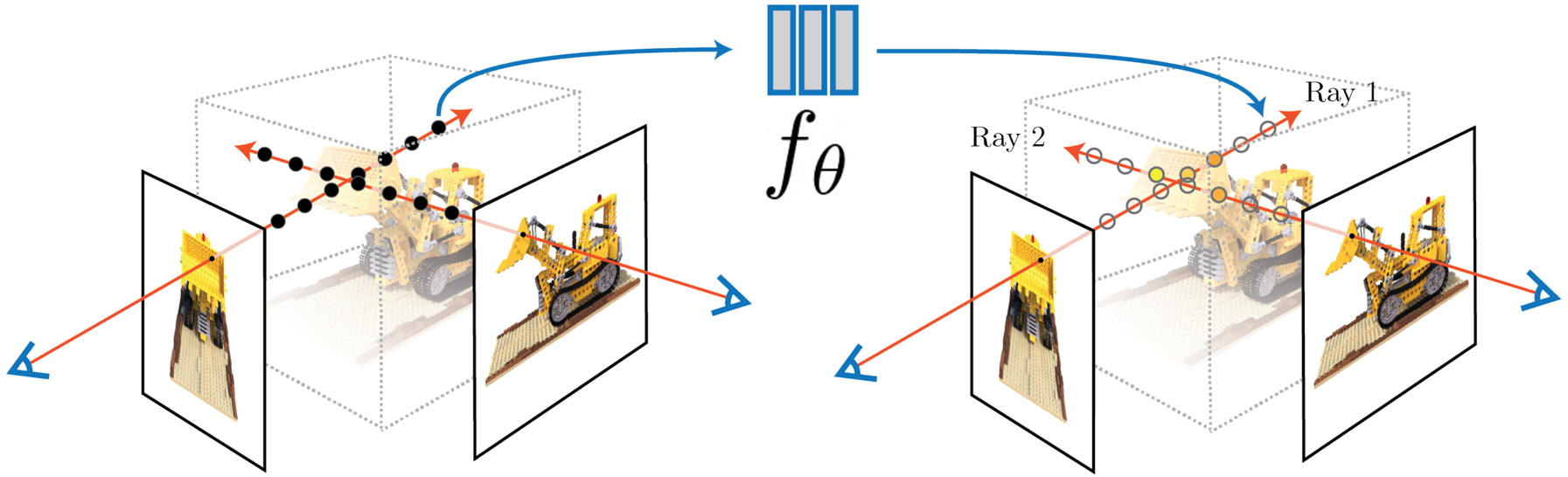
NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.

Learning Neural Radiance Fields

(pixel, camera) -> ray

$$\min_{\theta} \sum_{I} \sum_{\mathbf{p}} \left\| \text{render}(\mathbf{p}, \pi; f_\theta) - I[\mathbf{p}] \right\|^2$$

volume rendering



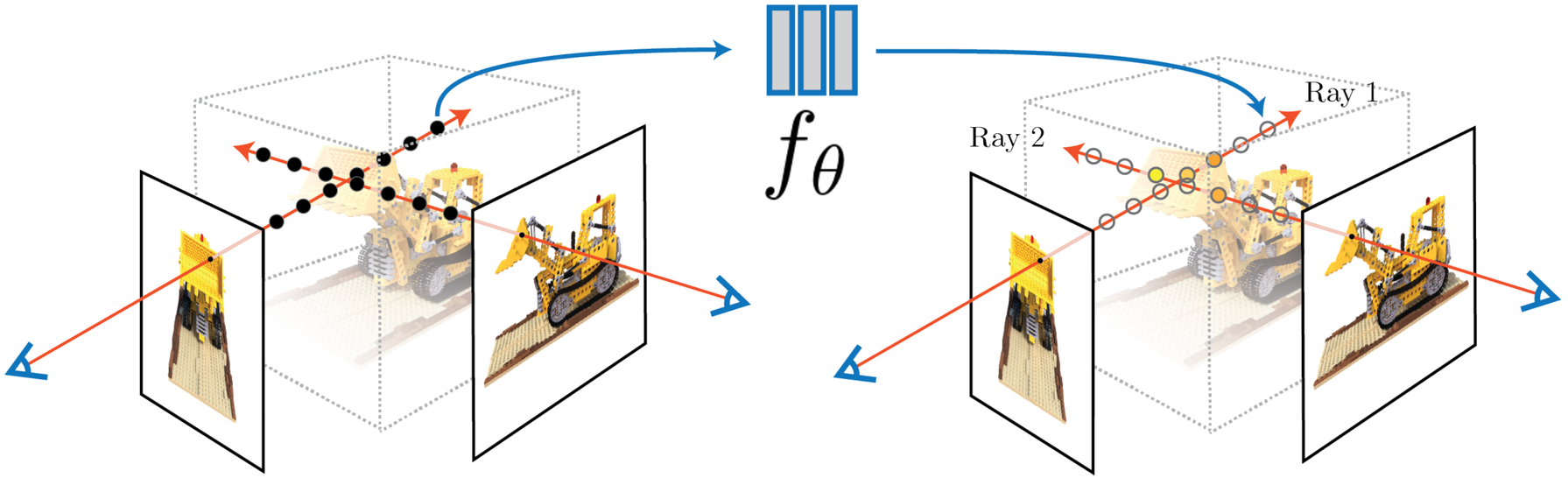
NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.

Learning Neural Radiance Fields

$$\min_{\theta} \sum_{I} \sum_{\mathbf{p}} \|\text{render}(\mathbf{p}, \pi; f_\theta) - I[\mathbf{p}]\|^2$$

Typically require multiple images (> 50) with known camera parameters

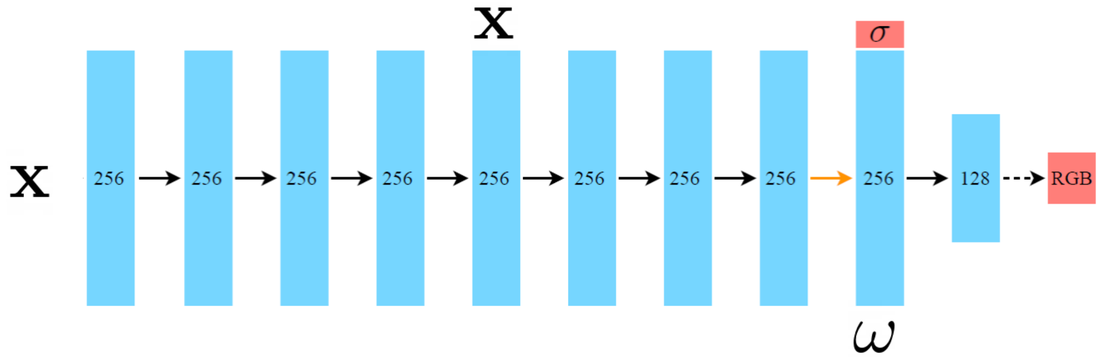
In practice, randomly sample a few pixels (rays) per optimization iteration



NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.

Learning Neural Radiance Fields

1) Acquire multiple images of a scene with associated camera viewpoints



2) Design a neural network $(\sigma, \mathbf{c}) = f_\theta(\mathbf{x}, \omega)$

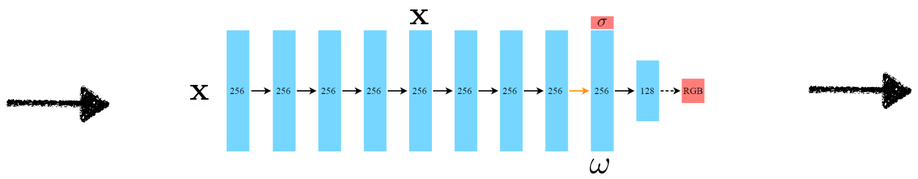
$$\min_\theta \sum_I \sum_{\mathbf{p}} \|\text{render}(\mathbf{p}, \pi; f_\theta) - I[\mathbf{p}]\|^2$$

3) Train with a view-synthesis loss using volume rendering

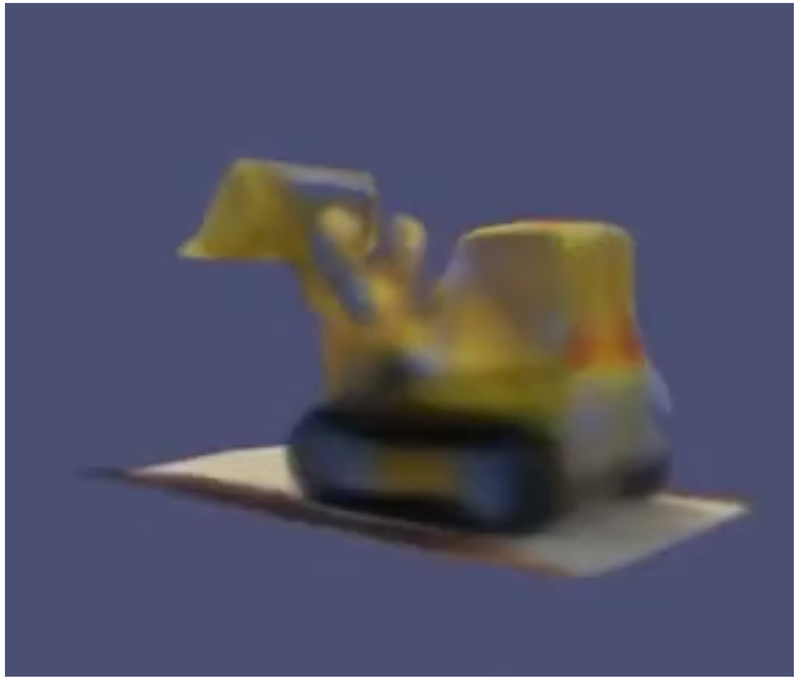
NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.

Learning Neural Radiance Fields: An Initial Attempt

100 training images



Optimized Neural Net



Novel-view Renderings

Cannot capture high-frequency details

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.

Modeling High Frequencies: A Toy Problem

$$(x, y) \rightarrow \boxed{f_\theta} \rightarrow (r, g, b)$$

Input Image

Using a ‘standard’ MLP

$$\mathbf{v}$$

$$\sin(\mathbf{v}), \cos(\mathbf{v})$$
$$\sin(2\mathbf{v}), \cos(2\mathbf{v})$$
$$\sin(4\mathbf{v}), \cos(4\mathbf{v})$$
$$\dots$$
$$\sin(2^{L-1}\mathbf{v}), \cos(2^{L-1}\mathbf{v})$$

$$\gamma(\mathbf{v})$$

Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. Tancik et. al.

Modeling High Frequencies: A Toy Problem

$$\gamma(x, y) \rightarrow \boxed{f_\theta} \rightarrow (r, g, b)$$

Input Image

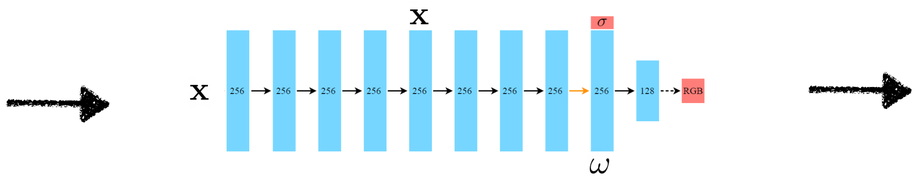
Using a ‘standard’ MLP

Using position encoding

Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. Tancik et. al.

Learning Neural Radiance Fields

100 training images



(using position encodings in input)

Optimized Neural Net

Novel-view Renderings

A great example that ‘execution matters’

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.

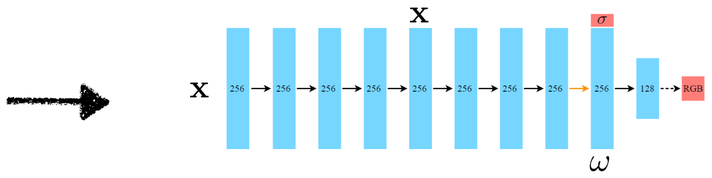
Learning Neural Radiance Fields

NeRF (Naive)

NeRF (with positional encoding)

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.

Learning Neural Radiance Fields



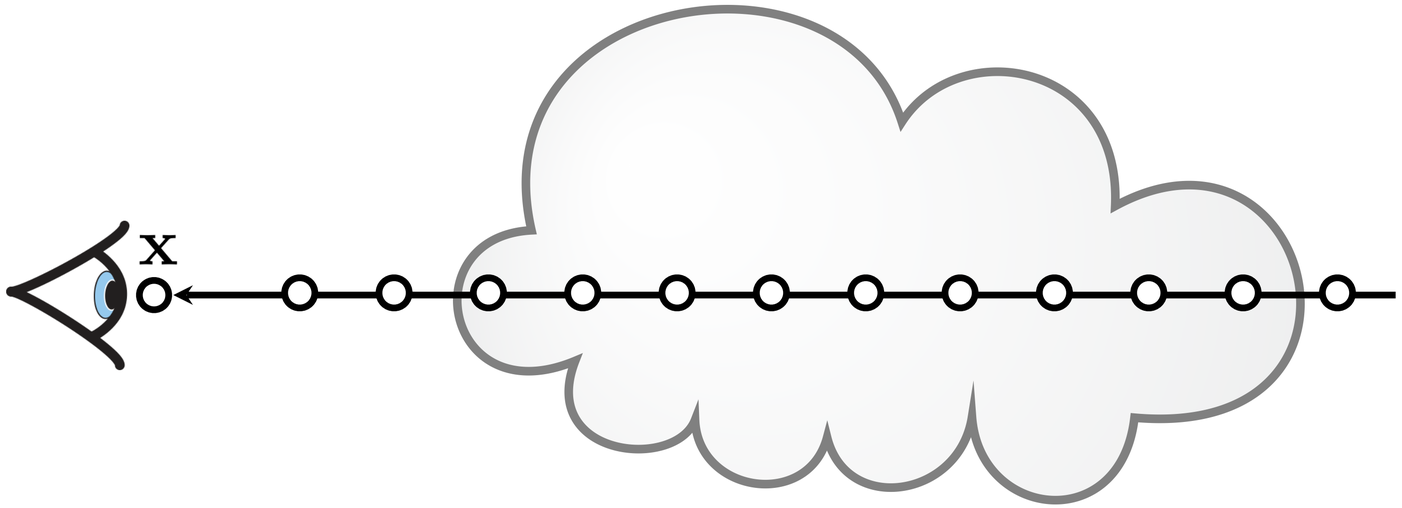
(using position encodings in input)

A very ‘simple’ representation and learning approach

Has inspired **many, many** extensions and followups

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.

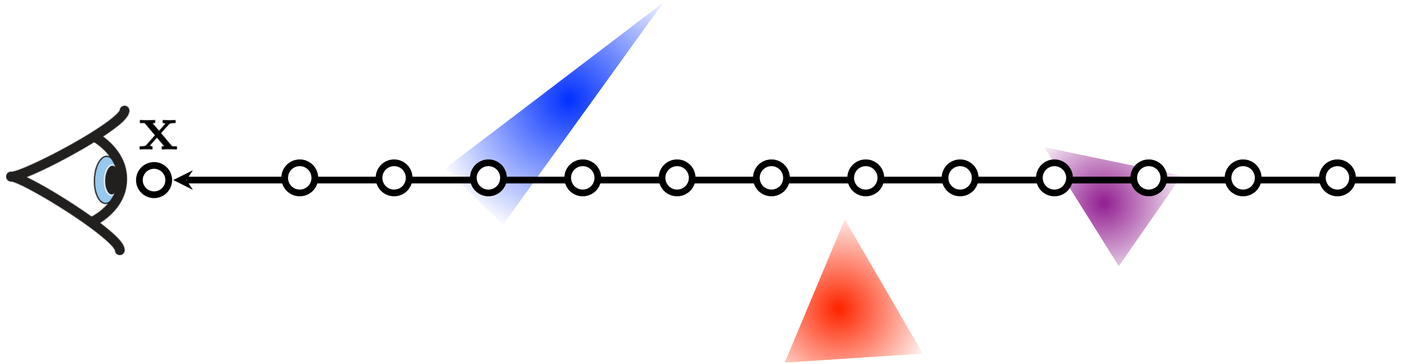
Recap: Rendering Volumes



1. Draw samples along the ray

2. Aggregate their contributions to render

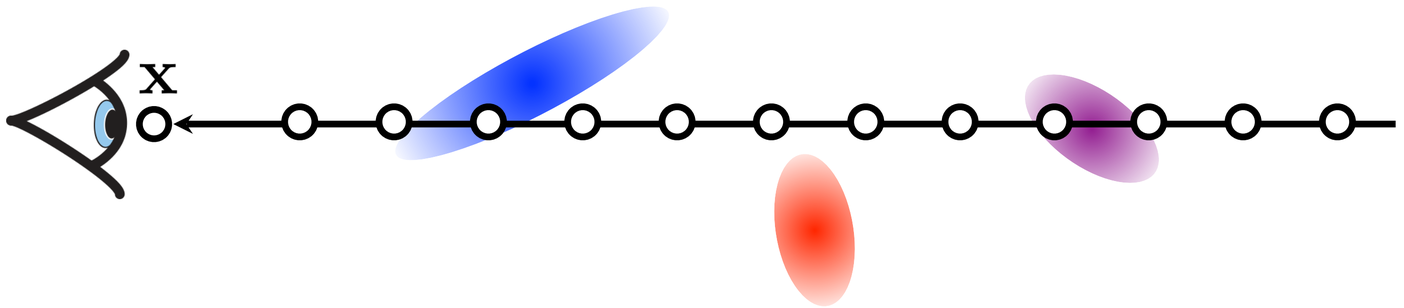
Rendering Primitives (e.g. Triangles/Meshes)



1. Draw samples along the ray

2. Aggregate their contributions to render

Rendering Primitives (e.g. Gaussians)

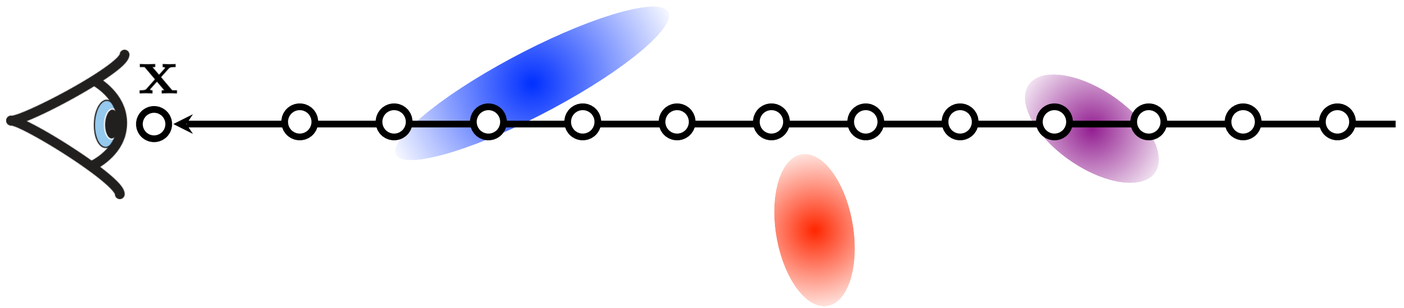


~~**1.** Draw samples along the ray~~

(wasteful — we know where the primitives are!)

2. Aggregate their contributions to render

Rendering Primitives (e.g. Gaussians)

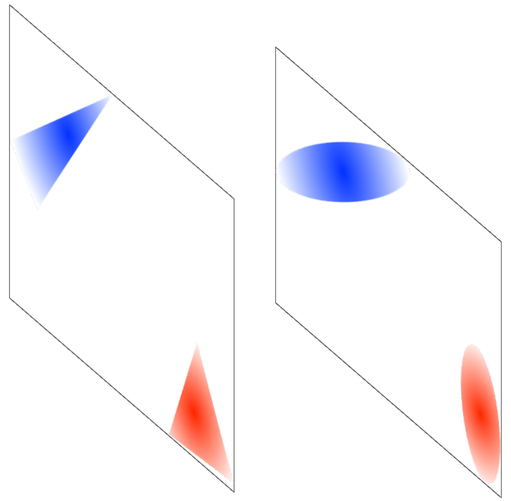


1. Find primitives that affect the ray/pixel

(can lead to significant speedups in rendering)

2. Aggregate their contributions to render

Differentiable Primitive Rendering



```
def render(primitives, camera):
    ###
    # Initialize a rasterization data structure
    # (records influencing primitives for each pixel)
    ###

    for primitive in primitives:
        prim2d = project(primitive, camera)
        ###
        # Update rasterization data structure
        ###

    for pixel in camera.grid:
        ###
        # Aggregate appearance from influencing primitives
        ###
```

Rasterization

Blending

Differentiable Gaussian Rendering

What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?

```
def render(gaussians, camera):
    ###
    # Initialize a rasterization data structure
    # (records influencing primitives for each pixel)
    ###

    for gaussian in gaussians:
        gauss2d = project(gaussian, camera)
        ###
        # Update rasterization data structure
        ###

    for pixel in camera.grid:
        ###
        # Aggregate appearance from influencing gaussians
        ###
```

Rasterization

Blending

Differentiable Gaussian Rendering

What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?

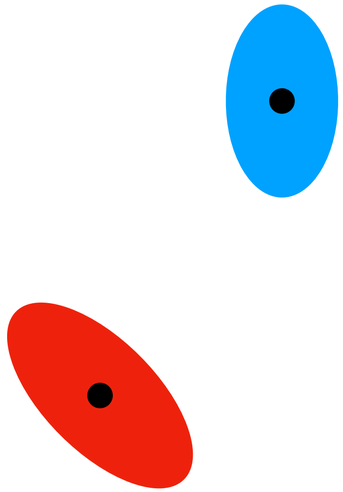
Differentiable Gaussian Rendering

What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?

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



Factorize as scale and rotation: $\mathbf{V} = RSS^TR^T$

Each gaussian also has an opacity and view-dependent color (via SH coefficients): $\alpha, \mathbf{c}$

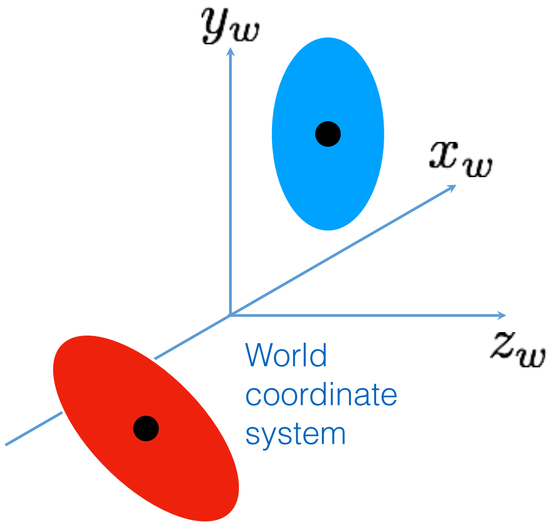
Slide adapted from Vincent Sitzmann.

Differentiable Gaussian Rendering

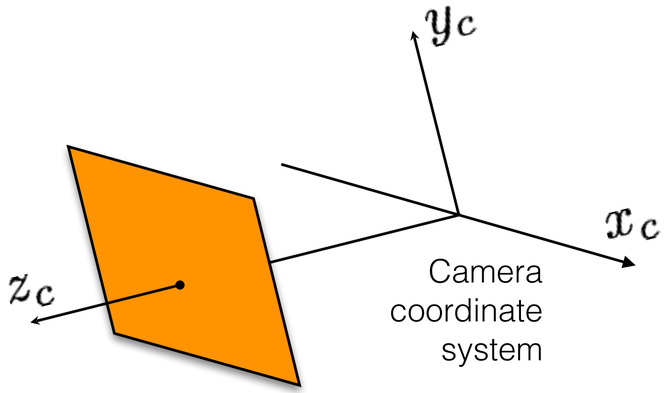
What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?



$\mathbf{p}, R, S$



$\mathbf{p}', R', S$

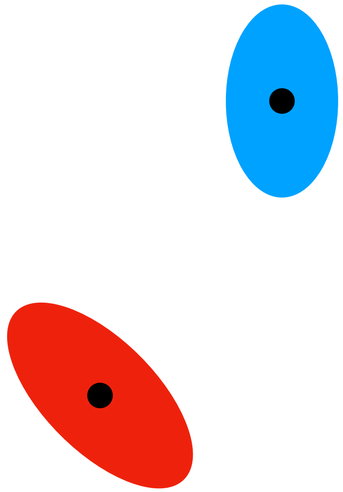
We can use the camera extrinsics to transform each 3D gaussian to the camera frame

Differentiable Gaussian Rendering

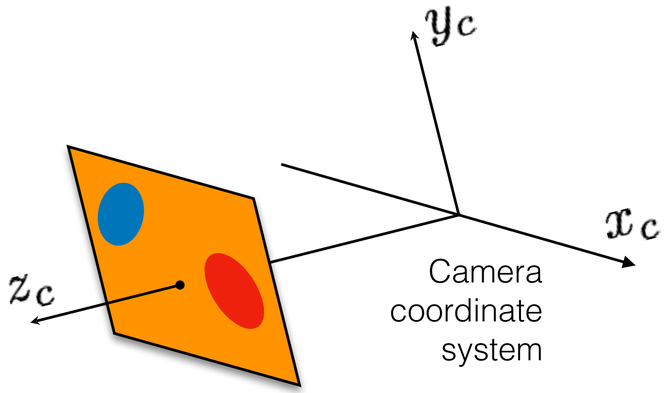
What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?



$\mathbf{p}', R', S$



Q: What is the image-space projection of a 3D gaussian?

A: Can approximate as a 2D gaussian!

(EWA Volume Splatting. *Zwicker et. al., 2001)*

$$\pi(\mathbf{x}) = \mathbf{u}$$

$$z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

π: Projection function for mapping 3D points to pixels

2D mean: $\mu_{2D} = \pi(\mu_{3D})$

2D covariance:

$$J = \frac{\partial \pi}{\partial \mathbf{x}}(\mu_{3D})$$

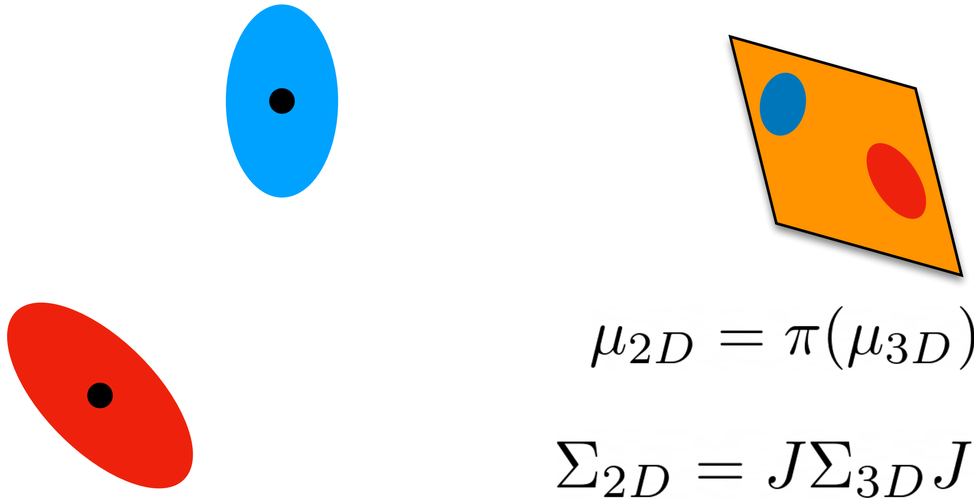
$$\Sigma_{2D} = J \Sigma_{3D} J^T$$

Differentiable Gaussian Rendering

What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?



$$\mu_{2D} = \pi(\mu_{3D})$$

$$\Sigma_{2D} = J\Sigma_{3D}J^T$$

1. Sort gaussians from closest to furthest from the camera

2. For each pixel **u**, compute opacity for each gaussian $\mathcal{G}_k$:

$$\bar{\alpha}_k = \alpha_k \frac{e^{-(\mathbf{u}-\mu_{2D}^k)^T(\Sigma_{2D}^k)^{-1}(\mathbf{u}-\mu_{2D}^k)}}{2\pi|\Sigma_{2D}^k|^{0.5}}$$

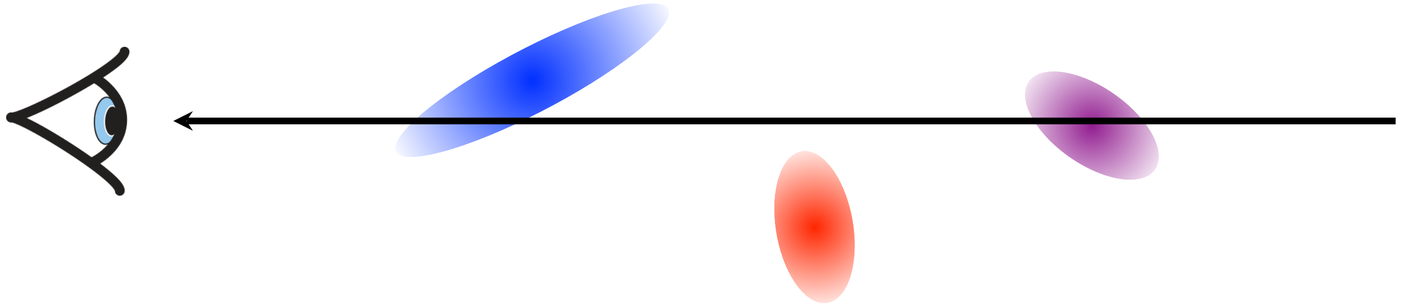
(In practice, can rasterize 'blocks' instead of entire image as not all gaussians influence all blocks)

Differentiable Gaussian Rendering

What is the representation of a 3D gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?



Compute per-gaussian weights based on opacities of current and previous gaussians:

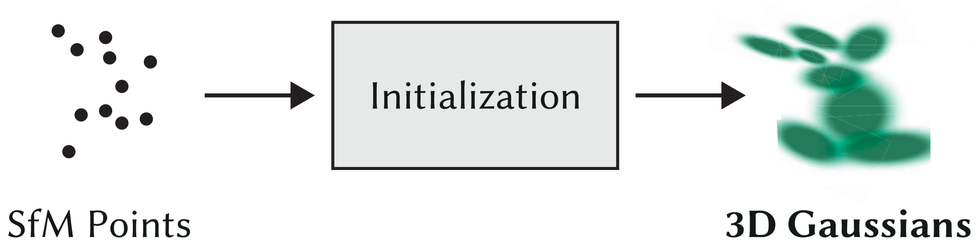
$$w_k = \bar{\alpha}_k \prod_{j=1}^{k-1}(1 - \bar{\alpha}_j)$$

Use per-gaussian SH coefficients and ray direction to get view-dependent color $\mathbf{c}_k$

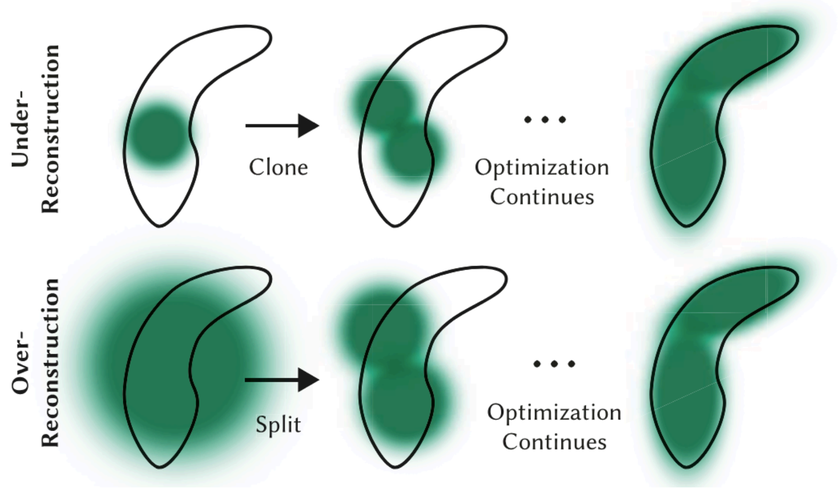
Aggregate to obtain pixel color:

$$\mathbf{c} = \sum_k w_k \mathbf{c}_k$$

Gaussian Splatting: Bells and Whistles



Initialize with sparse point cloud from SfM



Split/clone gaussians based on heuristics

3D Gaussian Splatting for Real-Time Radiance Field Rendering. *Kerbl et. al.*


Menu
Views
Capture
3D Gaussians
Camera Point view
Metrics
57.56 (17.37 ms)
VSync On


Menu Views Capture
▶ 3D Gaussians
▶ Camera Point view
Musical Instruments
▼ Metrics
89.37 (11.19 ms)

This video contains a voice-over

3D Gaussian Splatting for Real-Time Radiance Field Rendering

SIGGRAPH 2023
(ACM Transactions on Graphics)

Bernhard Kerbl*
Inria UNIVERSITÉ CÔTE D'AZUR

Georgios Kopanas*
Inria UNIVERSITÉ CÔTE D'AZUR

Thomas Leimkühler
mpii max planck institut informatik

George Drettakis
Inria UNIVERSITÉ CÔTE D'AZUR

* Denotes equal contribution

GraphDeco
GRAPHics and Design with hEterogeneous COntent

Bonus: NeRF Song

Questions?