Neural Volumetric Rendering

Many slides from ECCV 2022 Tutorial by Angjoo Kanazawa,
Ben Mildenhall, Pratul Srinivasan, Matt Tancik

Capturing Reality

Earliest cave painting (45,500 years old) in Sulawesi, Indonesia

Capturing Reality

Monet’s Cathedral series: study of light 1893-1894

Capturing Reality

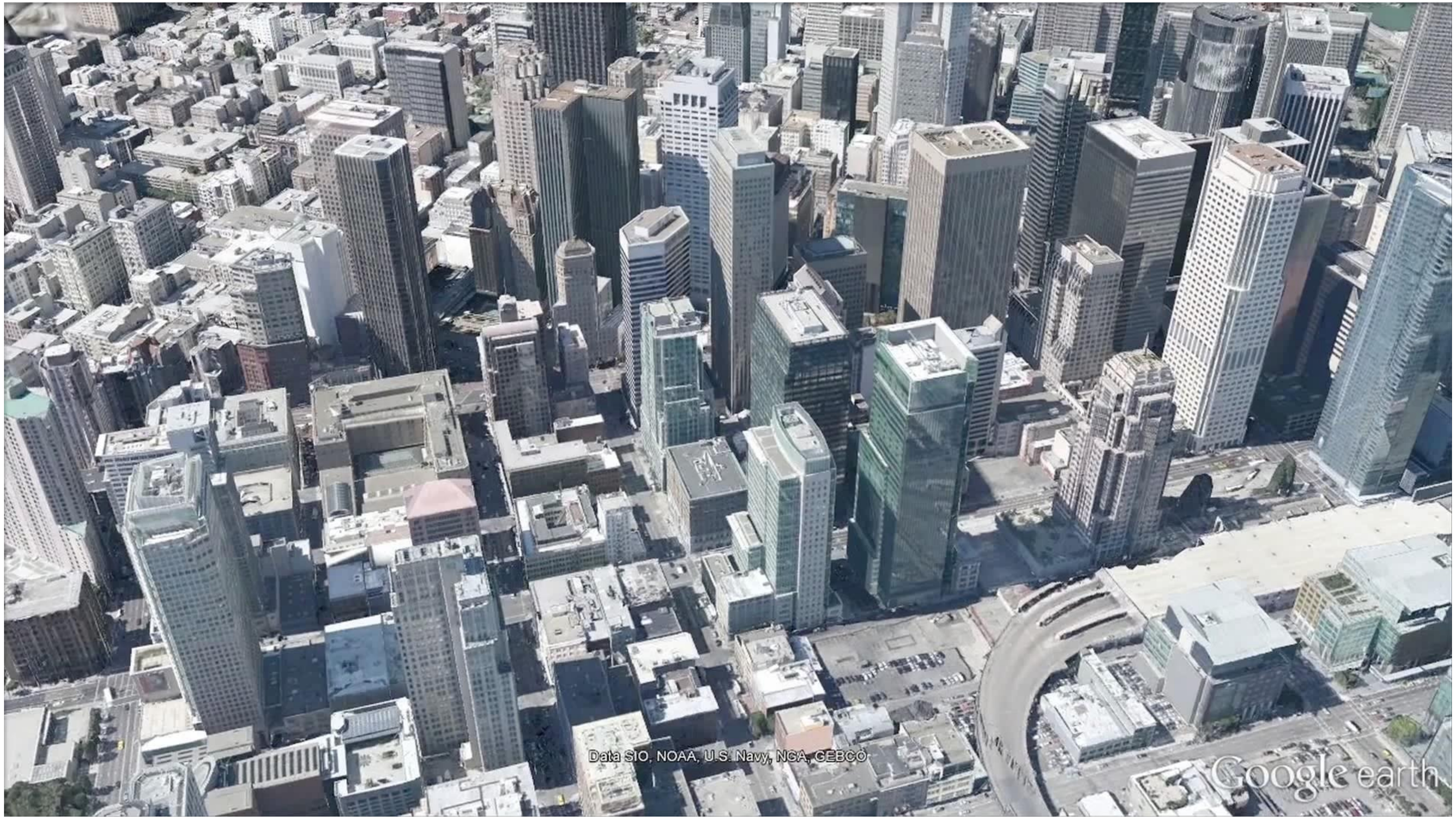
First self-portrait Cornelius 1839

First Movie - Muybridge 1878

Capturing Reality – in 3D

Building Rome in a Day, Agarwal et al. ICCV 2009

Capturing Reality – in 3D

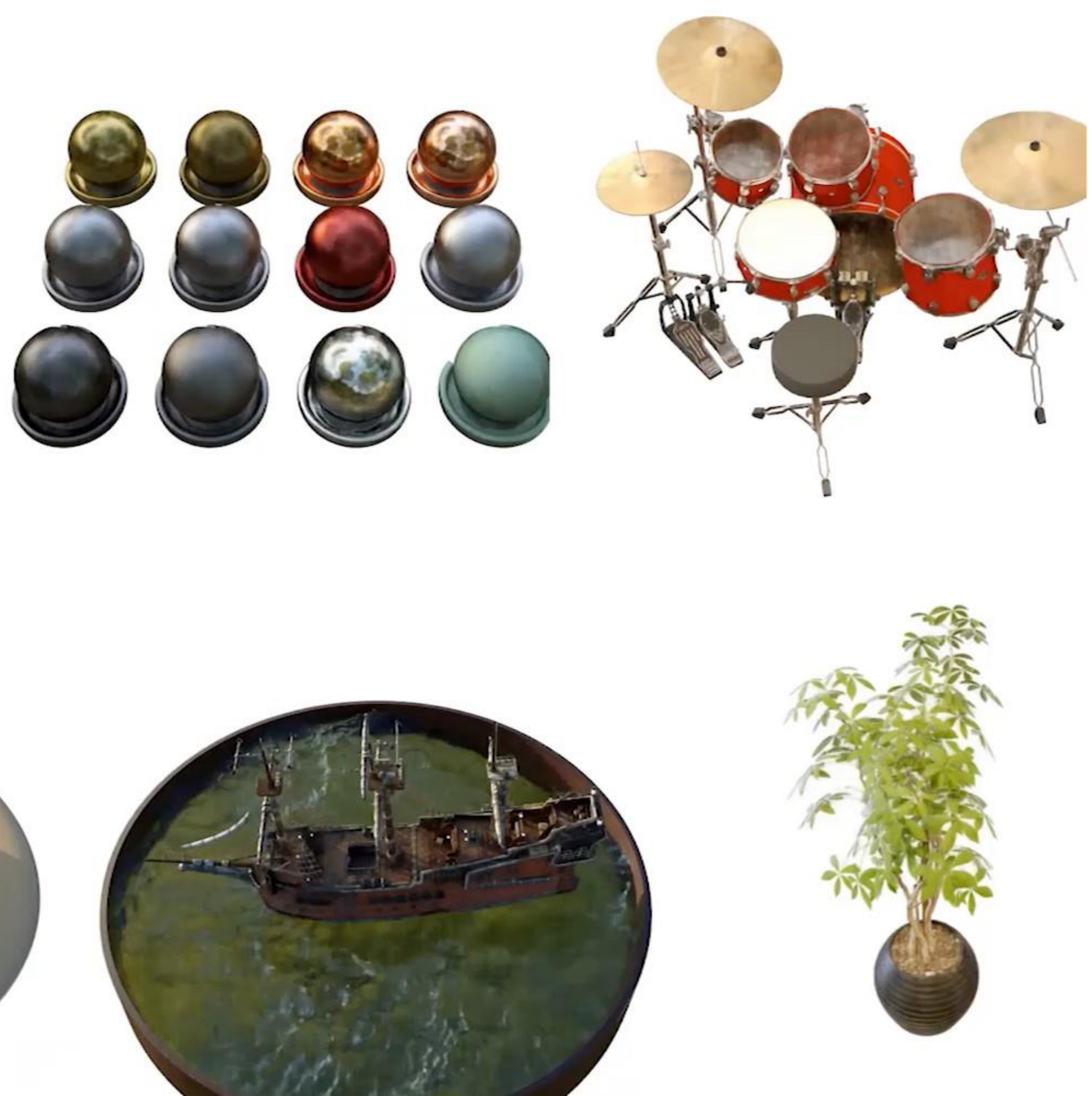


Google Earth 2016~

2020: Neural Radiance Field (NeRF)

Mildenhall*, Srinivasan*, Tancik*, Barron, Ramamoorthi, Ng, ECCV 2020

- Original NeRF paper: 11000+ citations in 4 years



Project 6 Notebook - Neural Radiance Fields (NeRF)

In this project, you will learn:

1. Basic usage of the PyTorch deep learning library
2. How to understand and build neural network models in PyTorch
3. How to build a Neural Radiance Field NeRF from a set of images
4. How to synthesize novel views from a NeRF

If this is your first time working with PyTorch, **please go through the "What is PyTorch" and "Neural Networks" tutorials in Deep Learning with PyTorch: A 60 Minute Blitz**. It won't take too long, but you will learn a lot and it will make this assignment much easier. You can use a new Colab notebook for the tutorials.

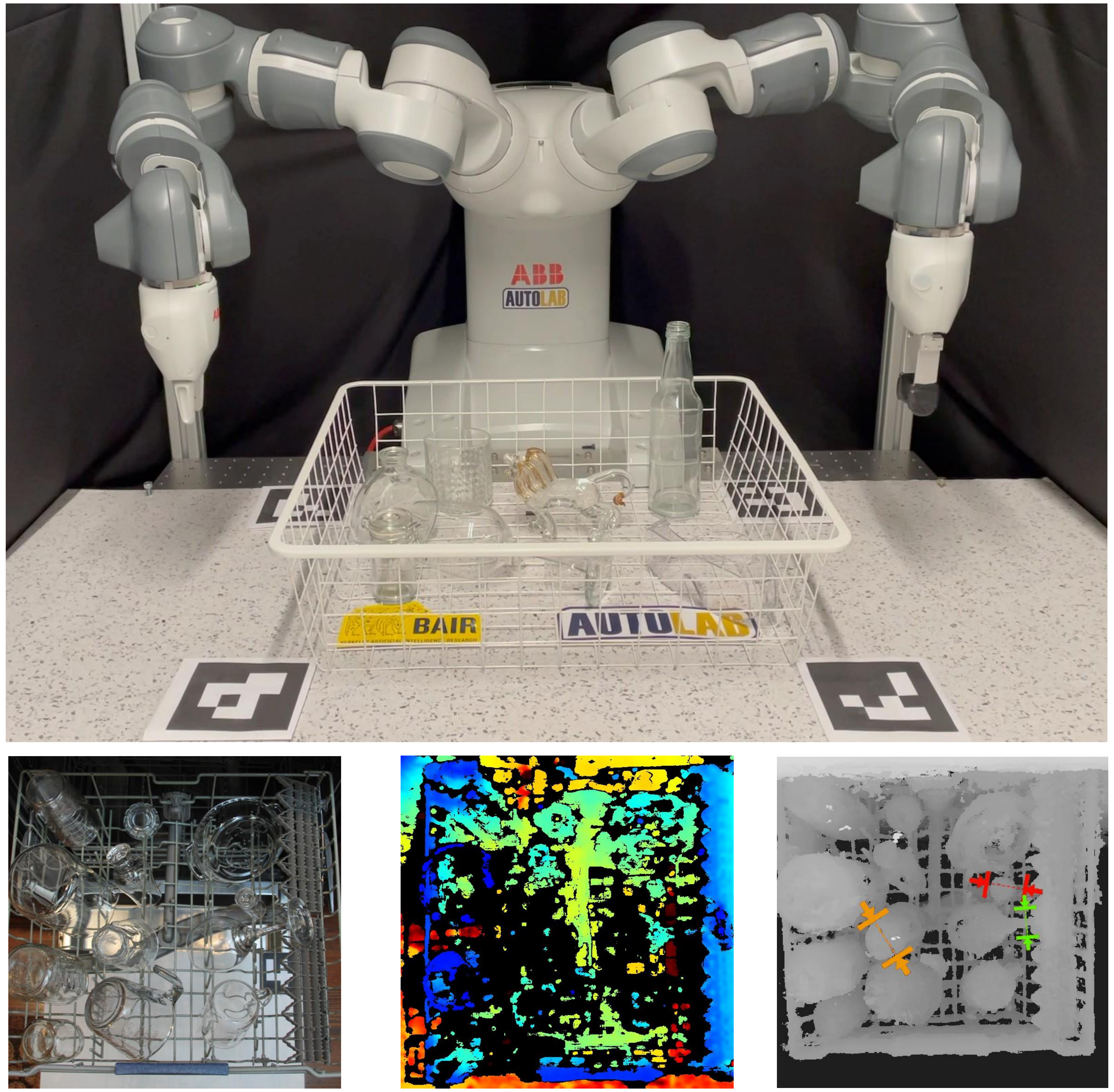
Initialization

Run the cell below to import the necessary libaries and print the device that the code will be run on (GPU vs.CPU). By default, you should get a GPU (i.e., the output is `cuda`).

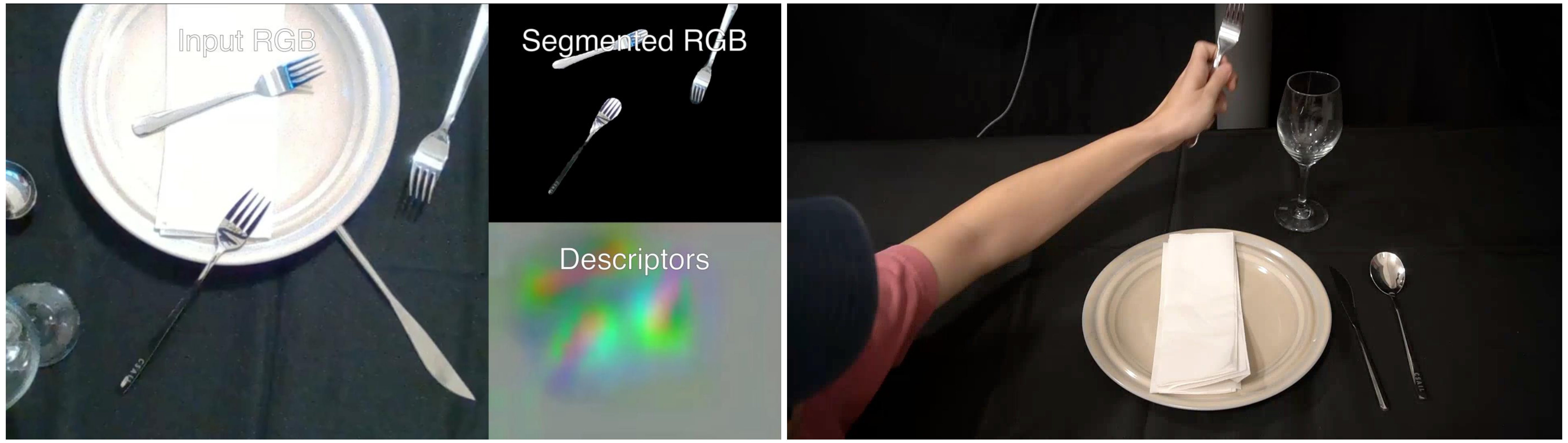
n [2]:

```
import os
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
import imageio
import time

device_type = (
    "cuda" if torch.cuda.is_available() else
    "mps" if torch.backends.mps.is_available() else
    "cpu"
)
device = torch.device(device_type)
print(device)

%load_ext autoreload
%autoreload 2
```

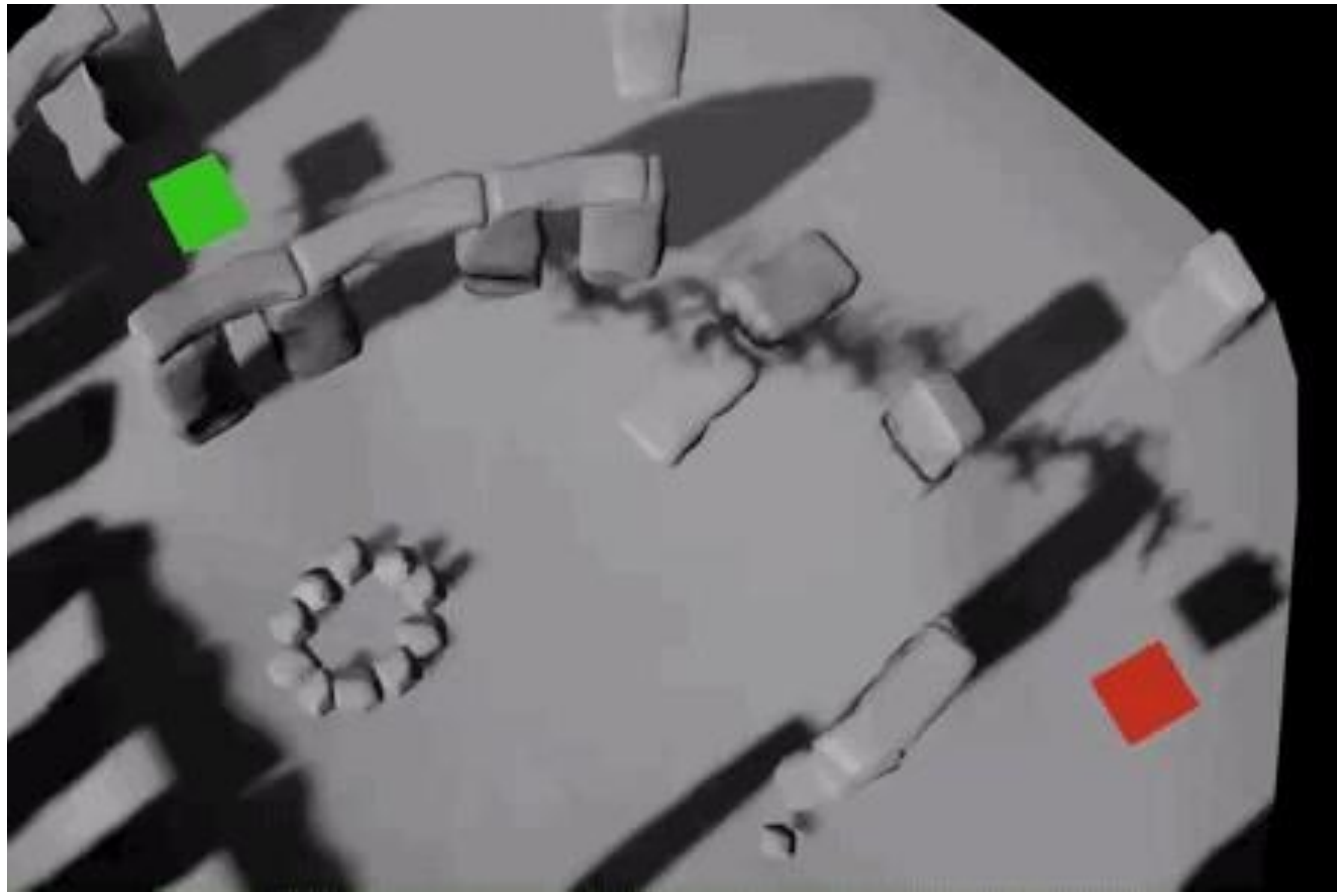
Robotics



Dex-NeRF: Using a Neural Radiance field to Grasp Transparent Objects, [Ichnowski and Avigal et al. CoRL 2021]



NeRF-Supervision: Learning Dense Object Descriptors from Neural Radiance Fields, [Yen-Chen et al. ICRA 2022]



Vision-Only Robot Navigation in a Neural Radiance World [Adamkiewicz and Chen et al. ICRA 2022]

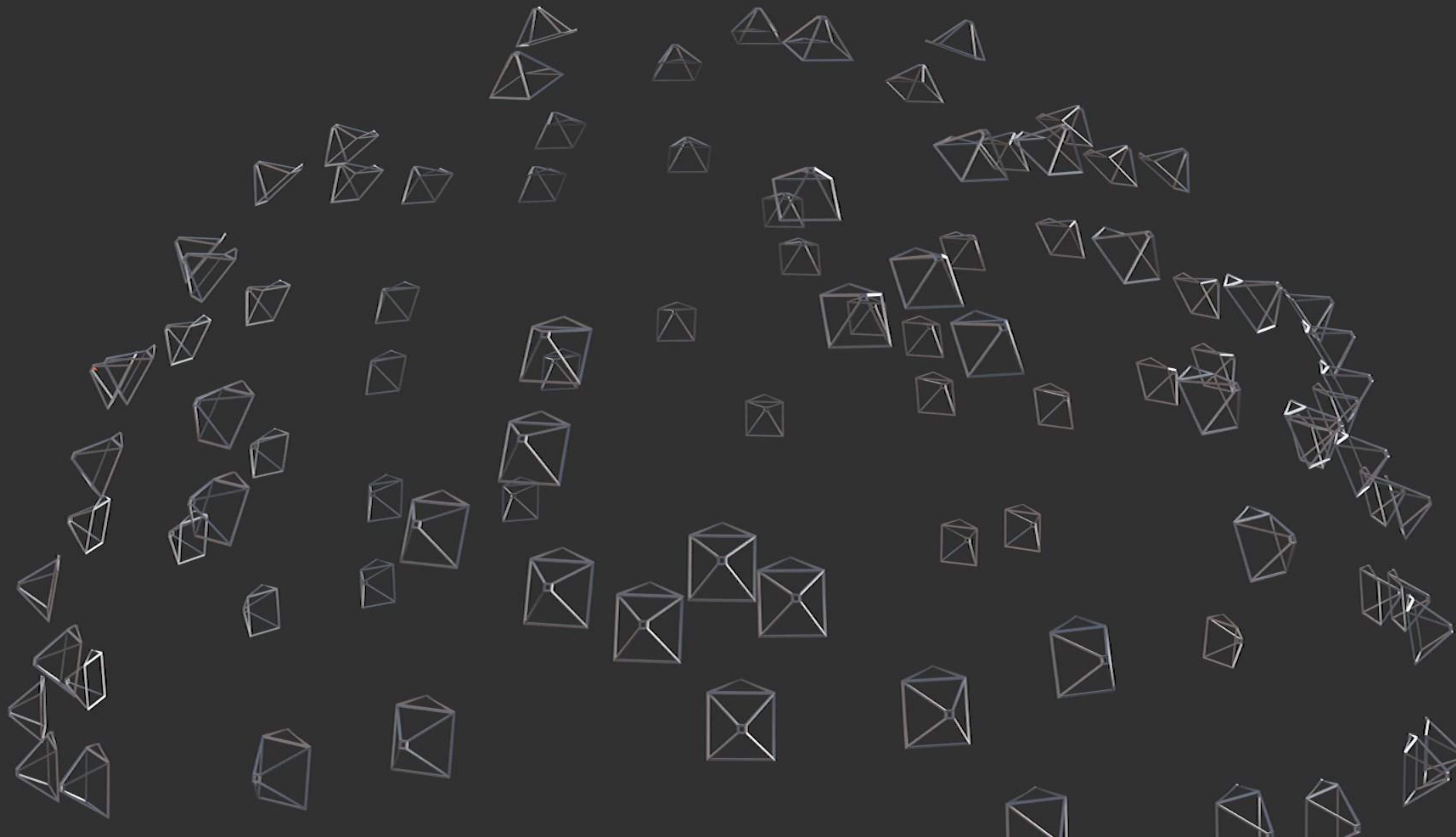
Birds Eye View

- What is NeRF?
- How is it different or similar to existing approaches?
- What is its historical context?

Problem Statement

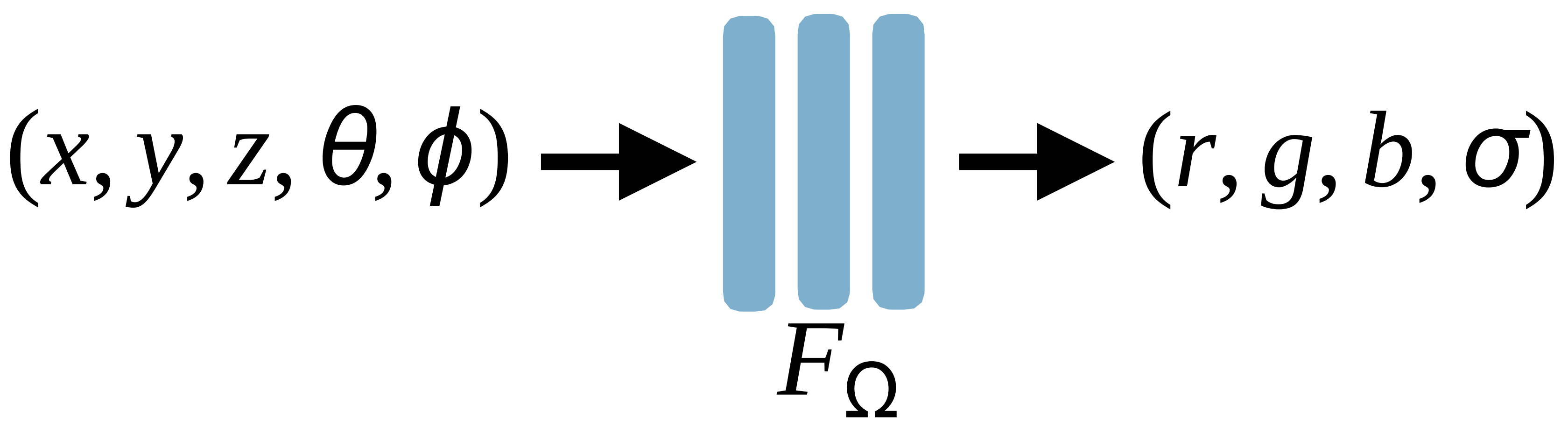
Input:
A set of calibrated Images

Output:
A 3D scene representation that renders novel views

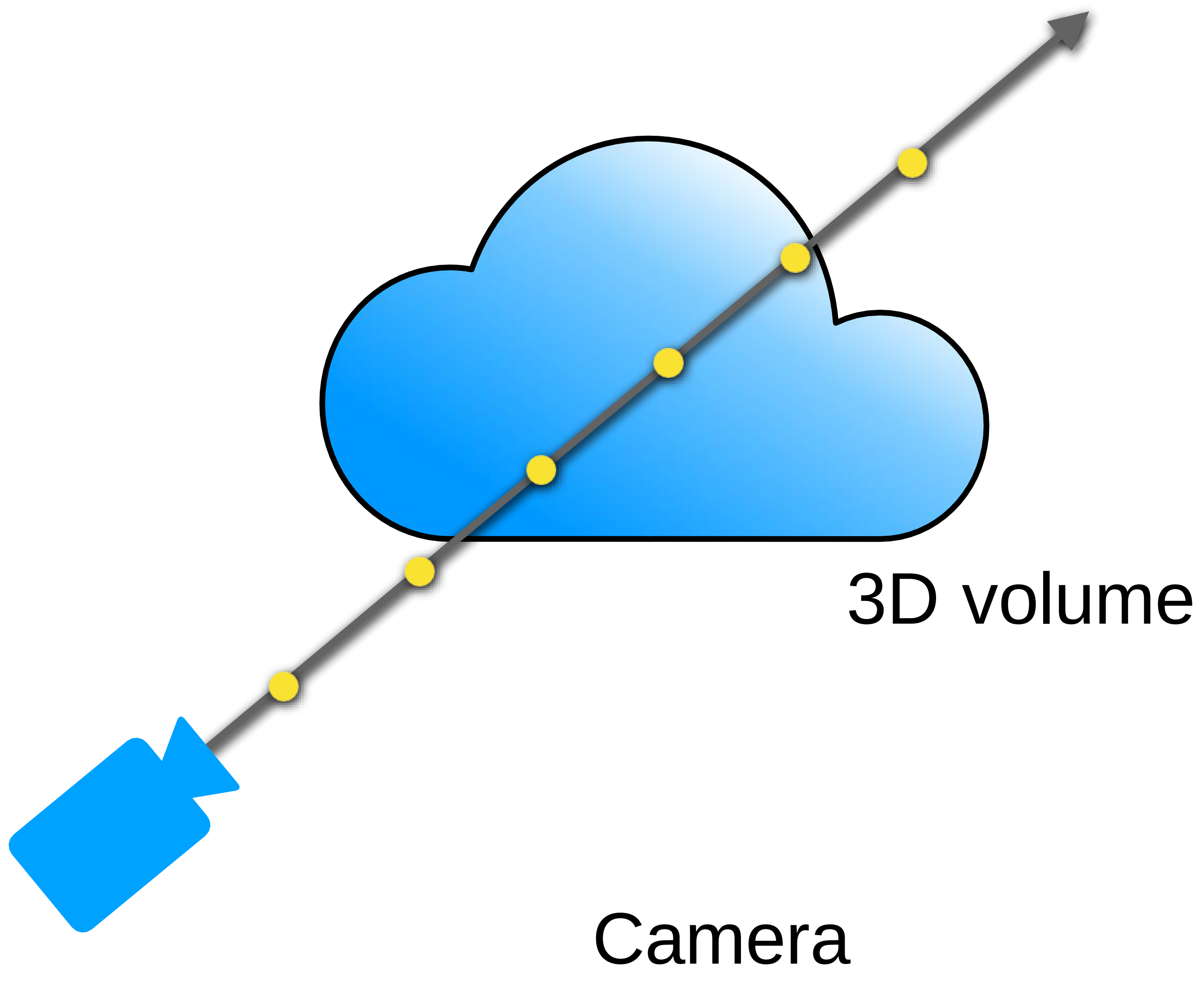


Three Key Components

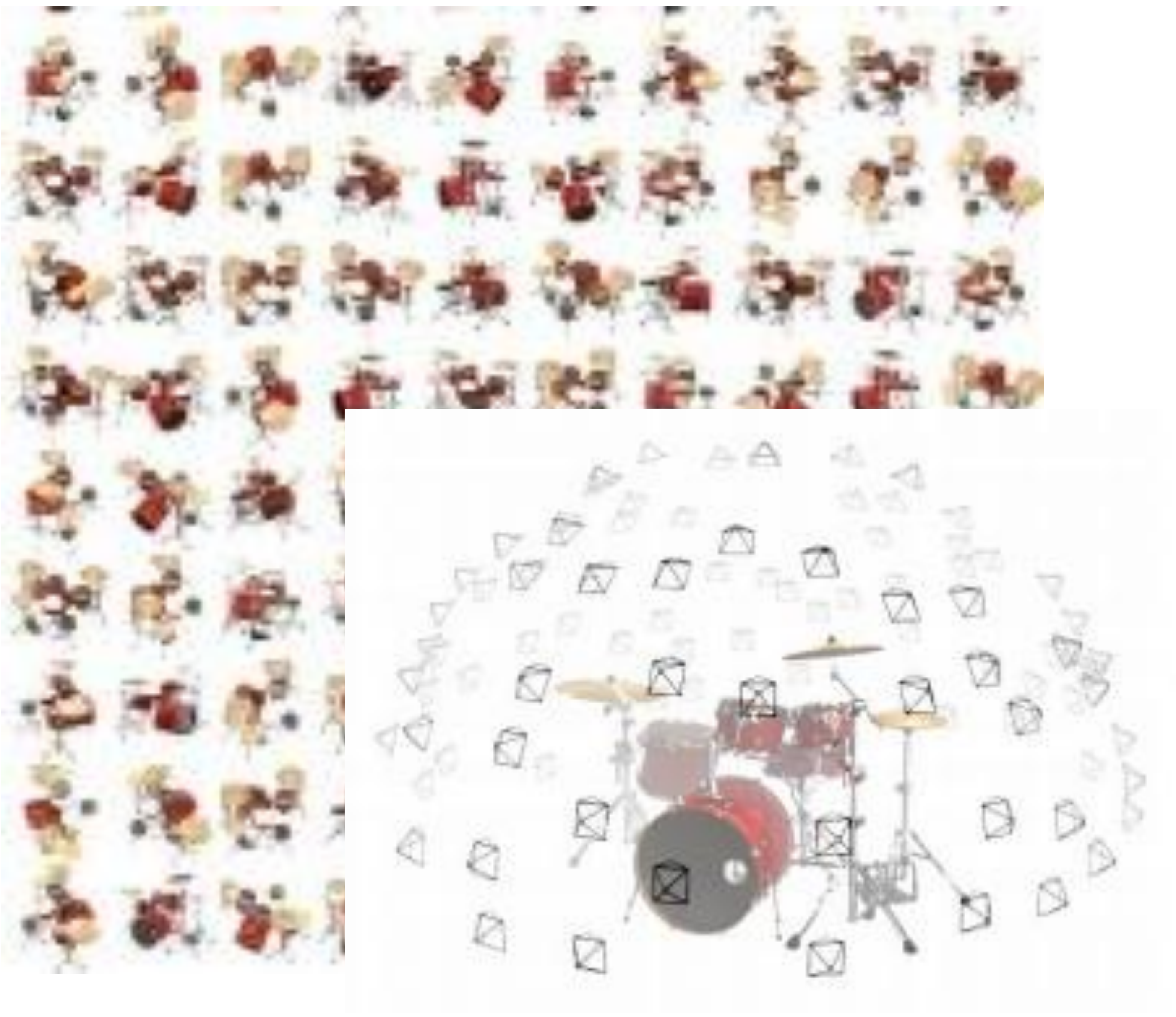
Objective: Synthesize
all training views



Neural Volumetric 3D
Scene Representation



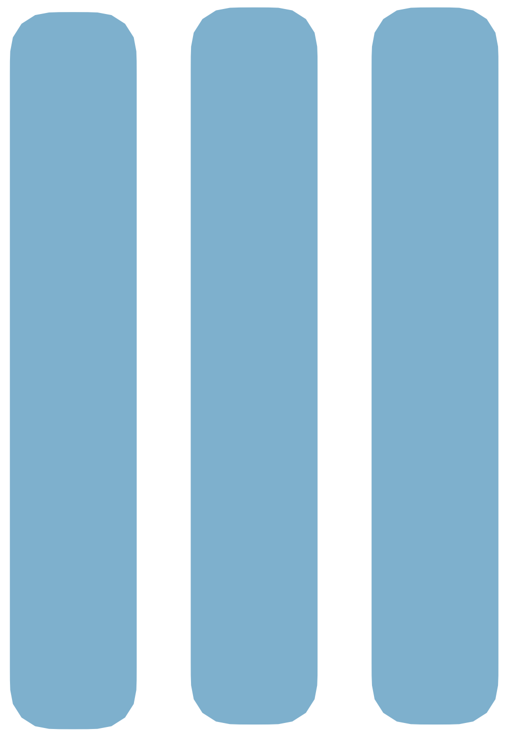
Differentiable Volumetric
Rendering Function



Optimization via
Analysis-by-Synthesis

Representing a 3D scene as a continuous 5D function

$$(x, y, z, \theta, \phi) \longrightarrow F_\Omega \longrightarrow (r, g, b, \underbrace{\sigma})$$



Spatial location

Viewing direction

F_Ω

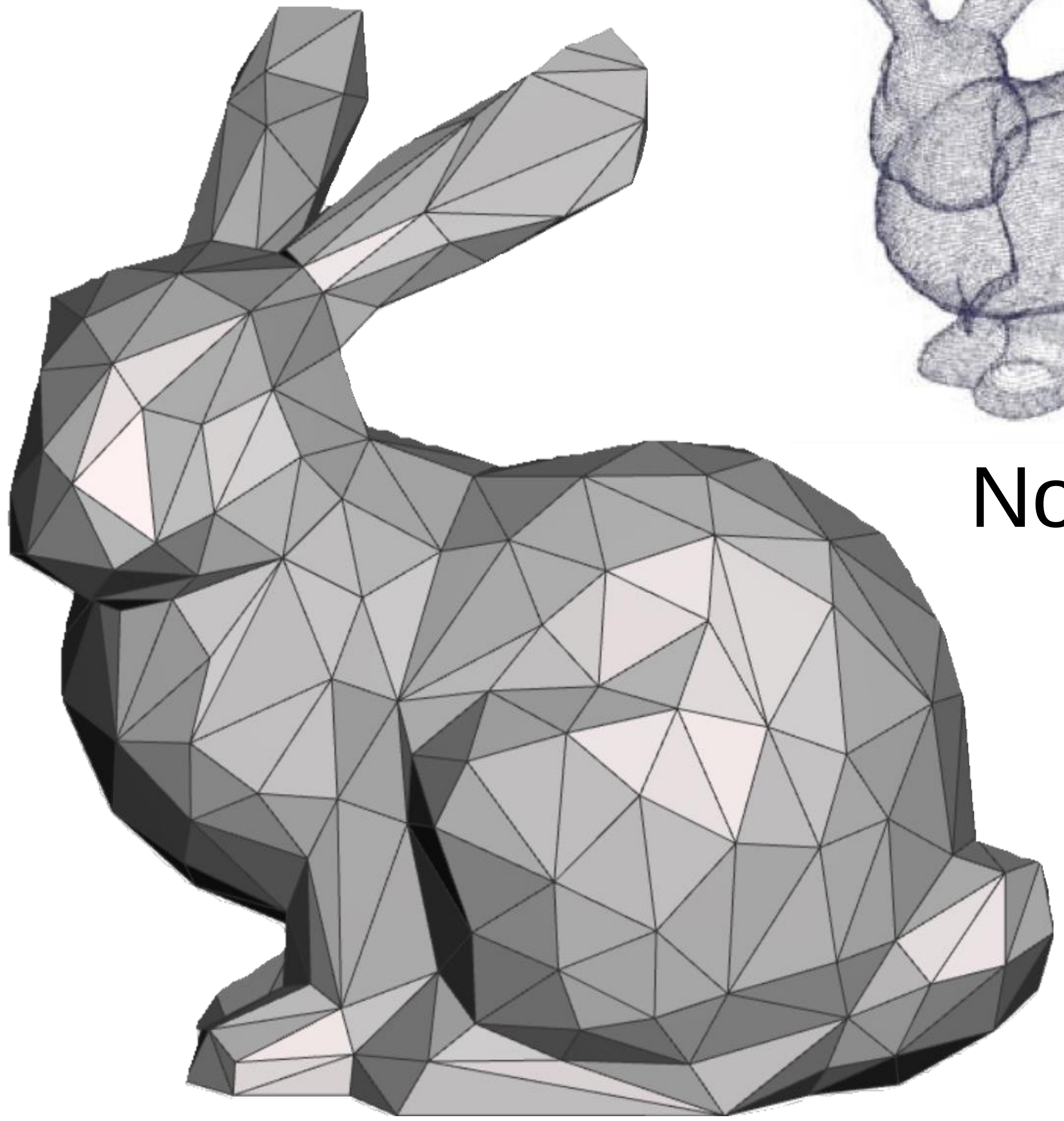
MLP
9 layers,
256 channels

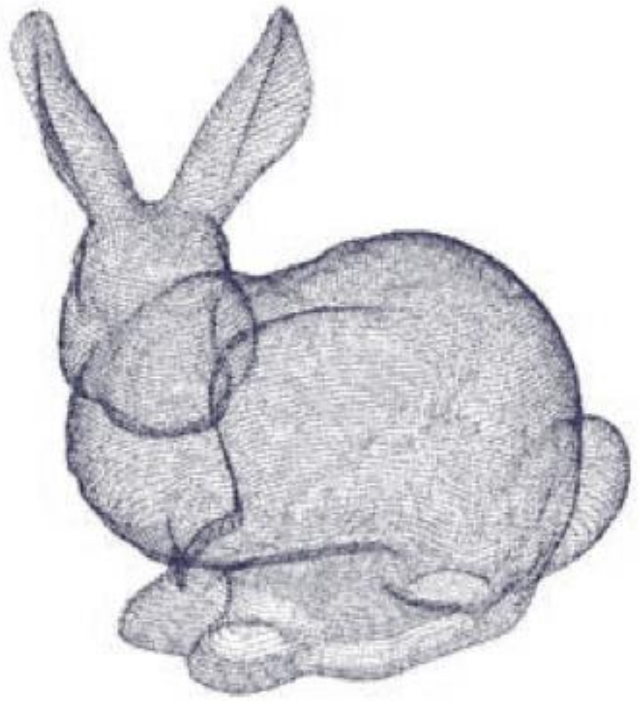
Output color

Output density

What kind of a 3D representation is this?

It is not a Mesh





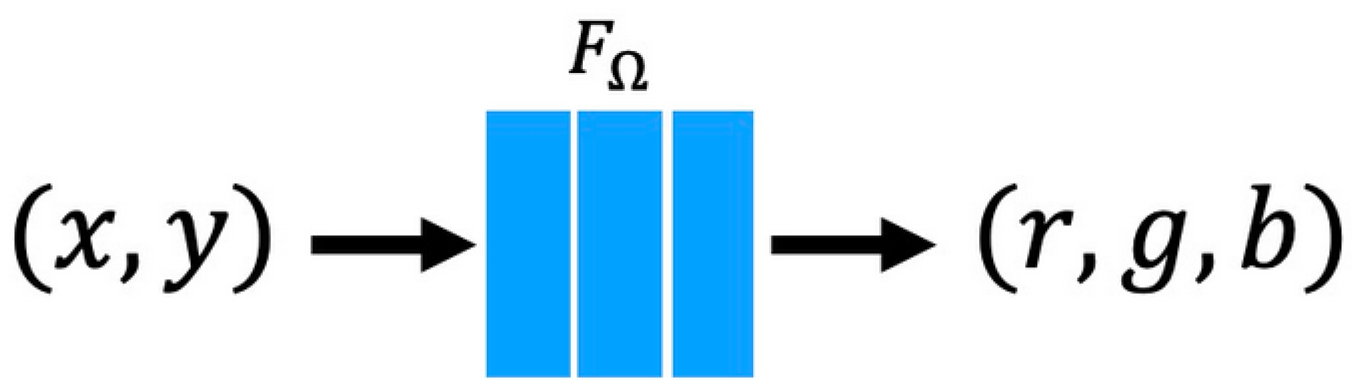
Not a point cloud
either

It is volumetric

It’s *continuous* voxels made of shiny transparent cubes

Part 1(b): 2D Image Fitting

Now, let's try to fit a 2D image with a multilayer perceptron (MLP)! In class we learn that we can store an 2D image with a coordinate-based MLP (as shown in the figure below). The input to this MLP is 2D pixel coordinate (x, y) as a pair of floating point numbers, and the output is RGB color of the corresponding pixel. This is a simple supervised learning problem, and we can just use simple gradient descent to train the network weights and see what happens.



First, let's define the network architecture for this 2D fitting task. We provide an example of network architecture called `Model2d` below. You can run all the way to the last cell in TODO 1(b) to execute the training process. Without any modification, you should get PSNR* $\sim$=27 after training for 10,000 iterations.

Now, your task is to modify `Model2d`, such that **after training for 10,000 iterations with num_encoding_functions=6, PSNR is greater than or equal to 30**. Please do not change the model name, the name of the existing arguments, or the input/output dimensions. Hint: You can try different model structure (e.g. more/fewer layers, smaller/bigger hidden dimensions).

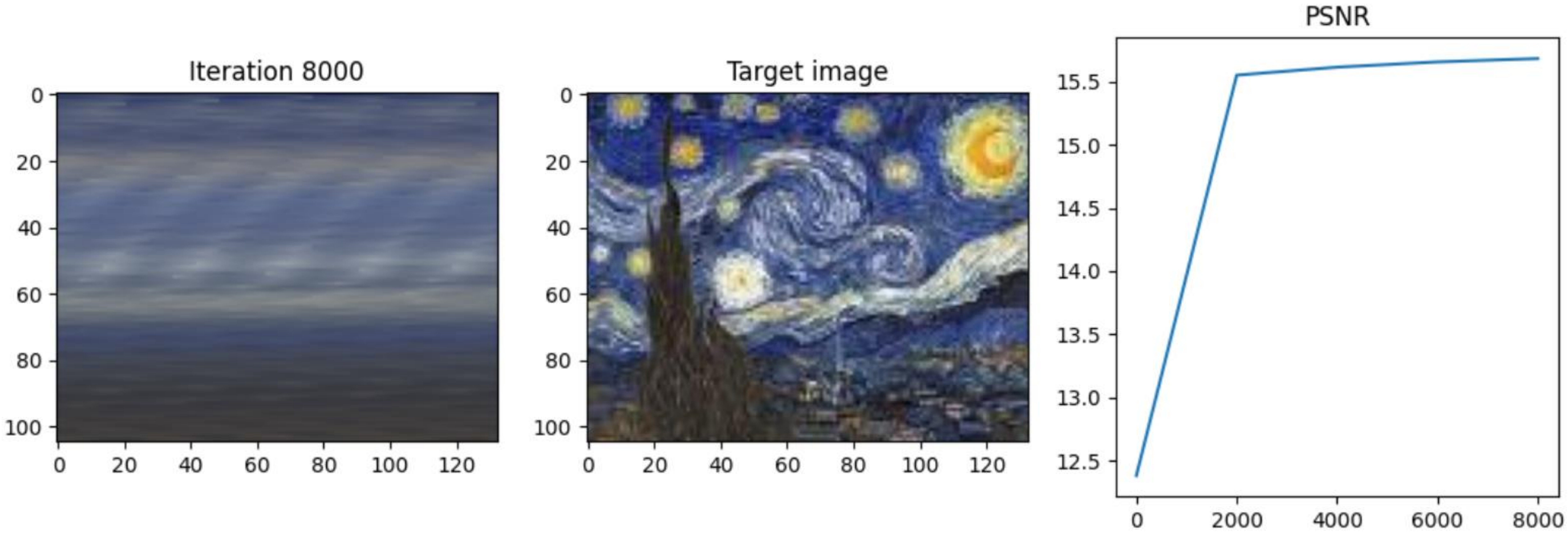
*PSNR is an image quality measurement. Higher PSNR generally indicates that the reconstruction is of higher quality.

Training comparison w/ and w/o positional encoding

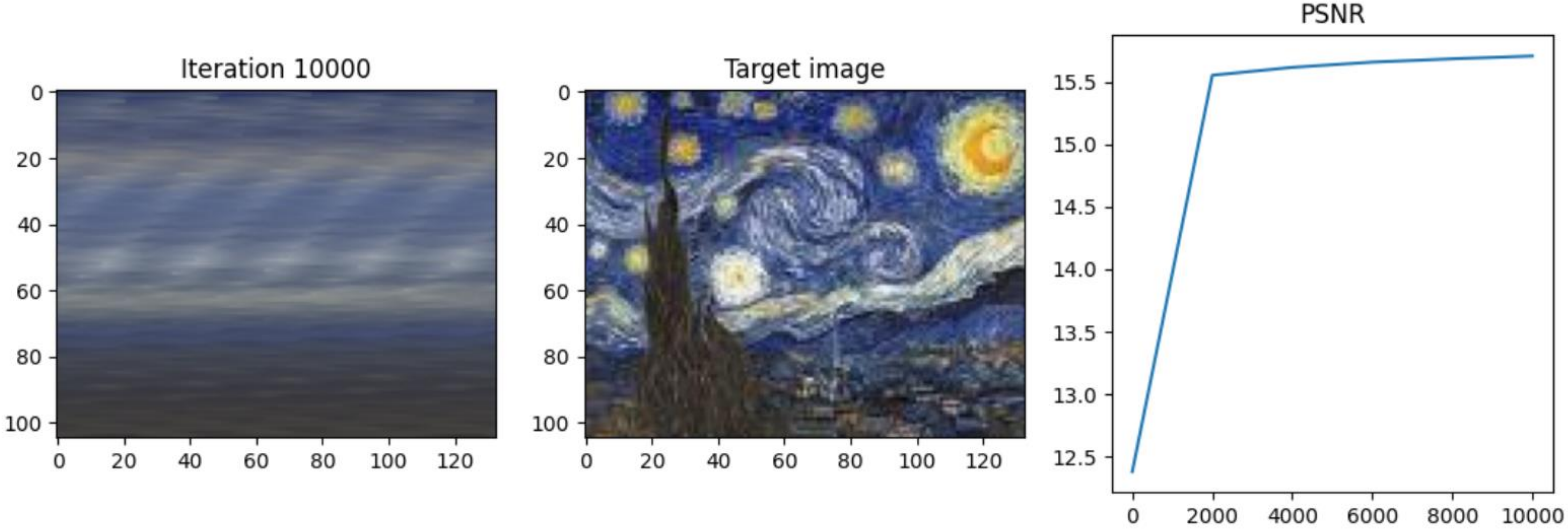
Run the following cell to initialize the training function.

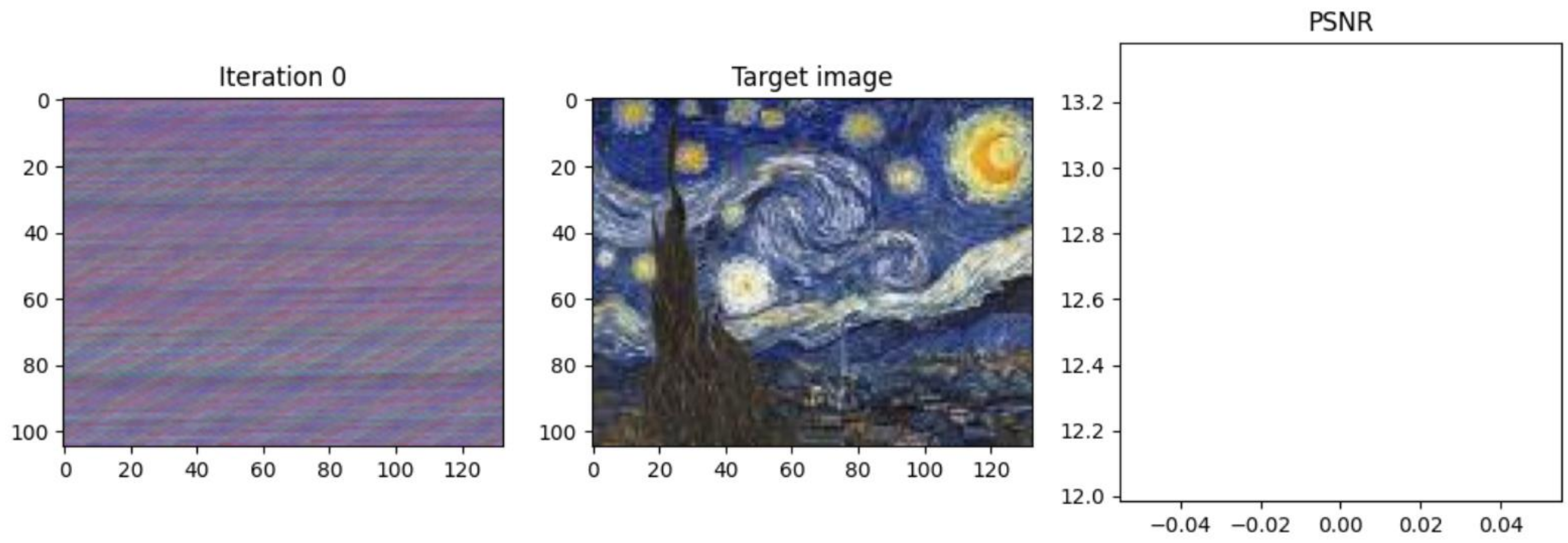
[4]:

```
# Load painting image
painting = imageio.imread("Starry-Night-canvas-Vincent-van-Gogh-New-1889_12.jpg")
painting = torch.from_numpy(np.array(painting, dtype=np.float32)/255.).to(device)
height_painting, width_painting = painting.shape[:2]

plt.figure(figsize=(13, 4))
plt.title("Starry Night painting")
plt.imshow(painting.detach().cpu().numpy())
plt.show()
```

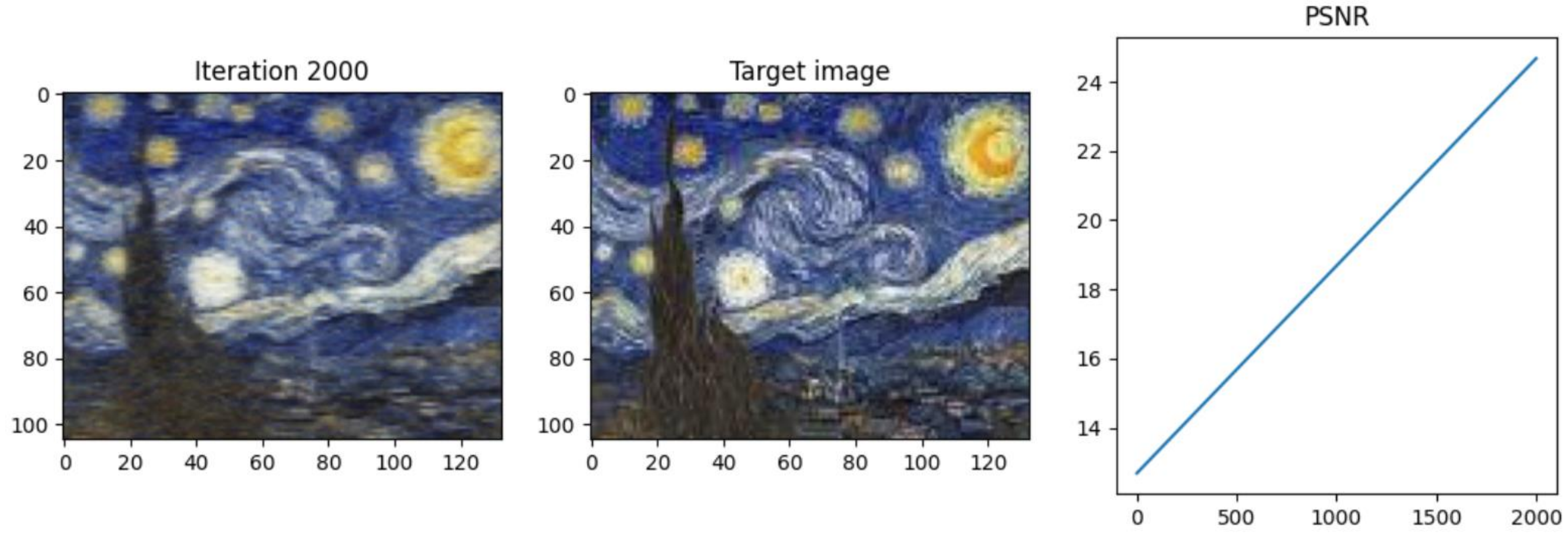


Iteration 10000 Loss: 0.0269 PSNR: 15.70 Time: 0.00 secs per iter 12.67 secs in total



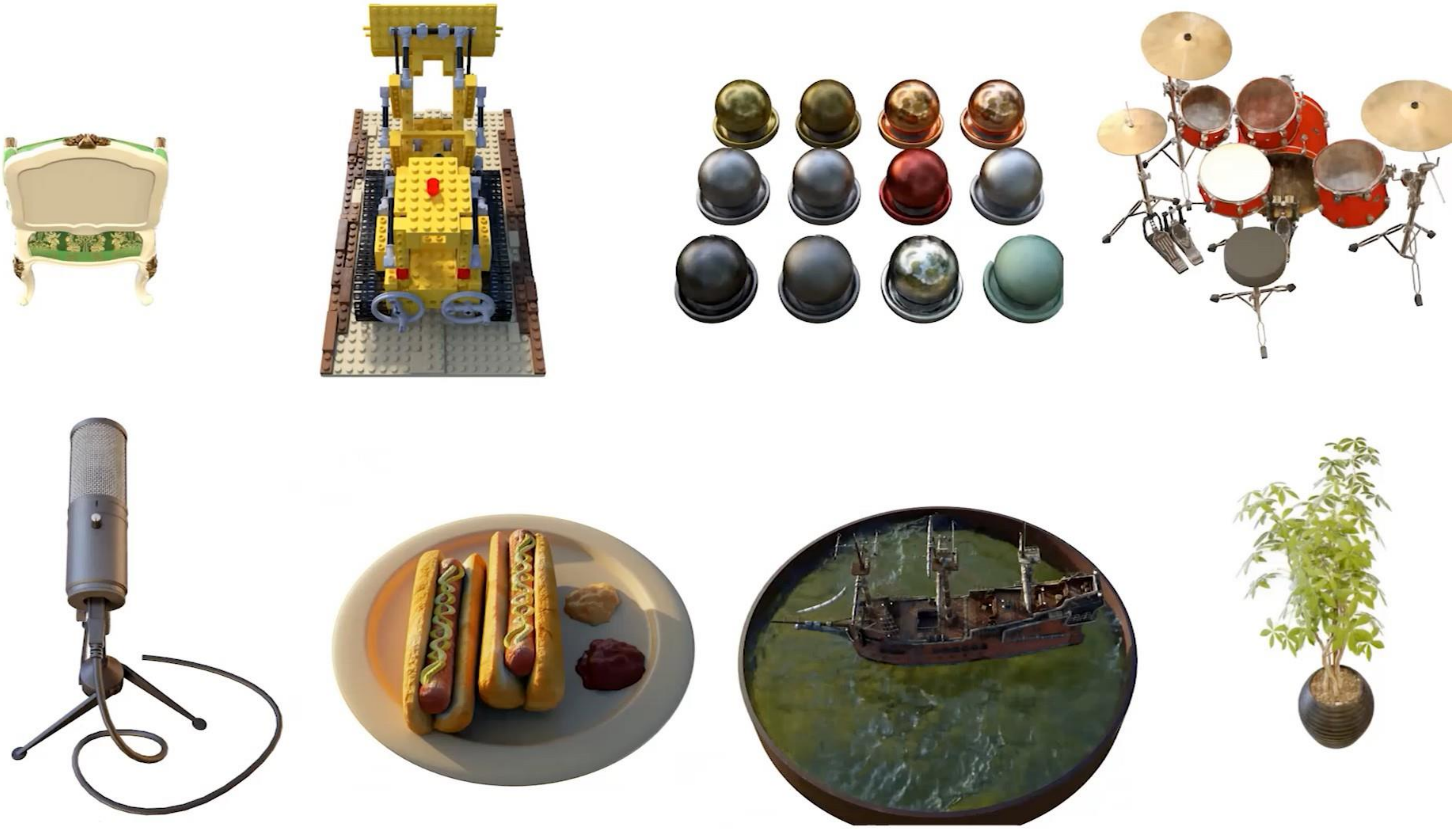



```
Iteration 2000  Loss: 0.0034  PSNR: 24.68 Time: 0.00 secs per iter 2.66 secs in total
```




```
Iteration 4000  Loss: 0.0024  PSNR: 26.23 Time: 0.00 secs per iter 5.15 secs in total
```

What is the problem that is being solved?



Plenoptic Function

Figure by Leonard McMillan

Q: What is the set of all things that we can ever see?

A: The Plenoptic Function (Adelson & Bergen '91)

Let's start with a stationary person and try to parameterize everything that they can see…

Slide credit: Alyosha Efros

Grayscale Snapshot

$P(\theta,\phi)$

- is intensity of light
  - Seen from a single position (viewpoint)
  - At a single time
  - Averaged over the wavelengths of the visible spectrum

Slides from Alyosha Efros

Color snapshot

$P(\theta,\phi,\lambda)$

- is intensity of light
  - Seen from a single position (viewpoint)
  - At a single time
  - As a function of wavelength

Slides from Alyosha Efros

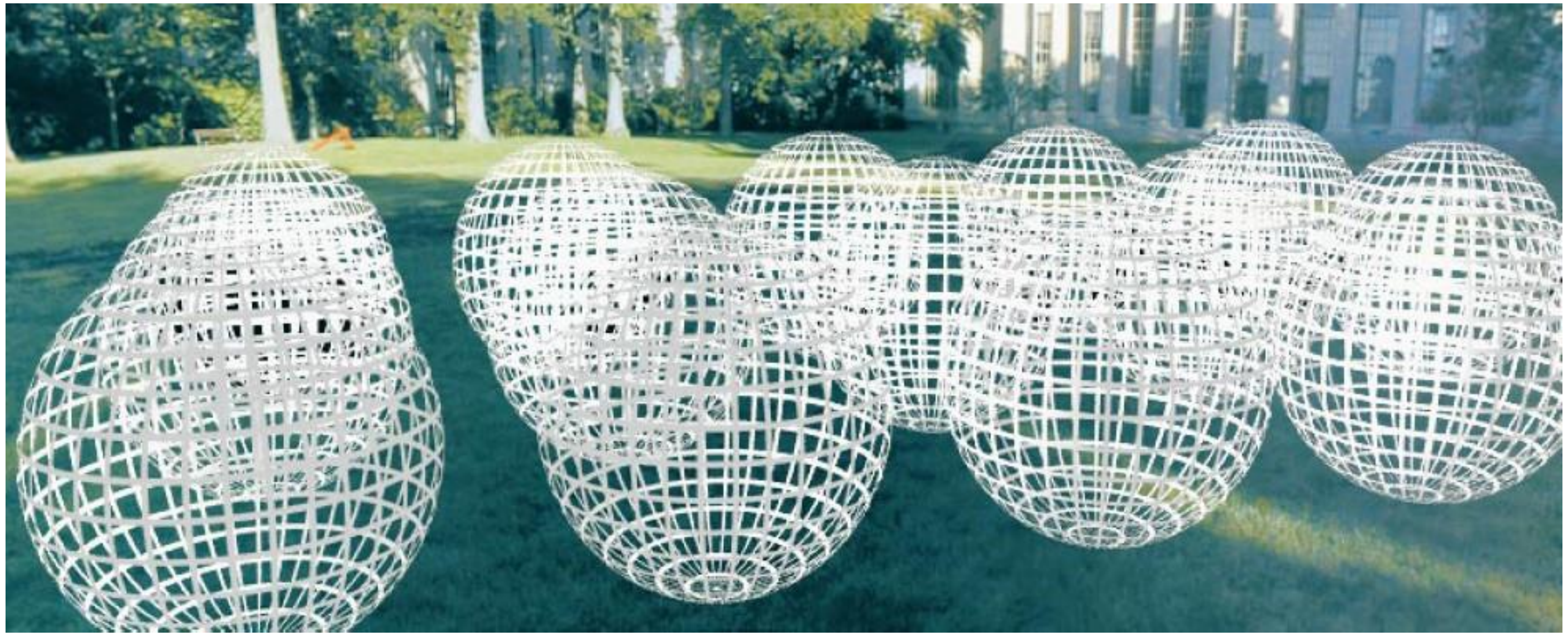
A movie

$P(\theta,\phi,\lambda,t)$

- is intensity of light
  - Seen from a single position (viewpoint)
  - Over time
  - As a function of wavelength

Slides from Alyosha Efros

A holographic movie

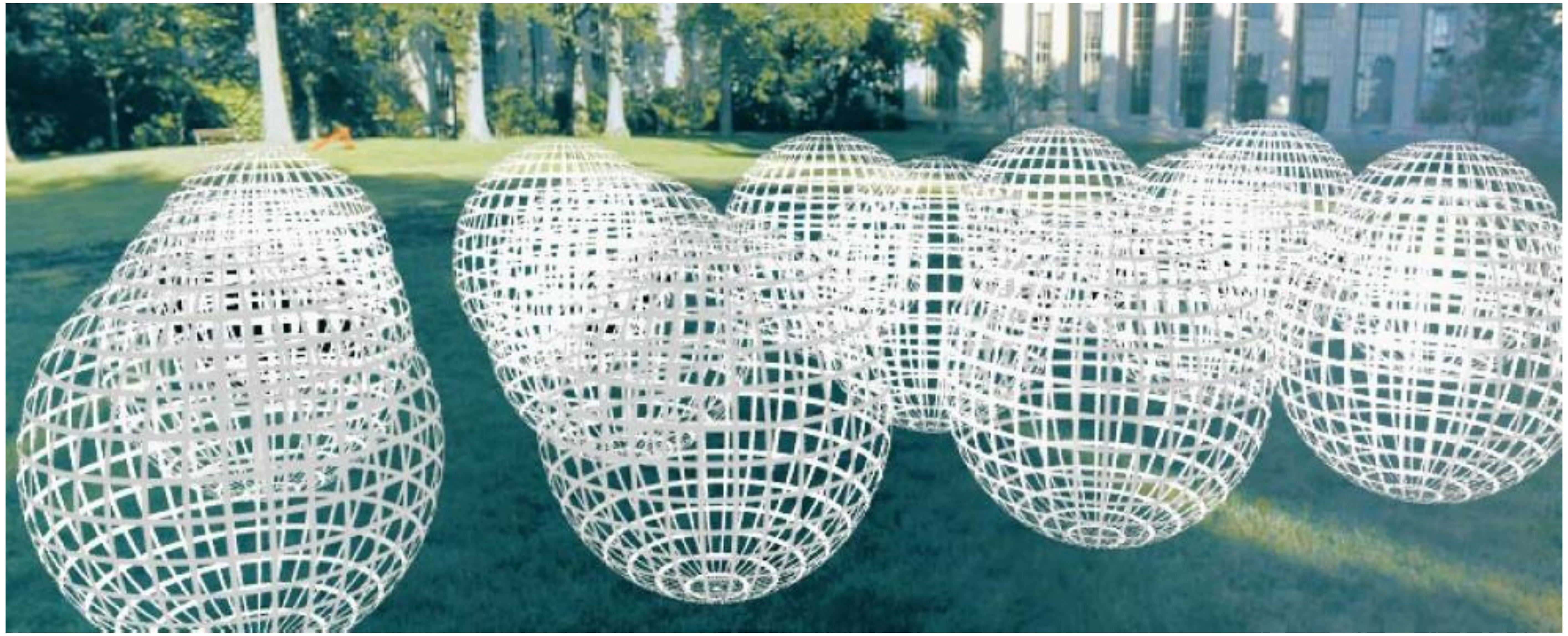


$$P(\theta,\phi,\lambda,t,V_x,V_y,V_z)$$

- is intensity of light
- Seen from ANY position and direction
- Over time
- As a function of wavelength

Slides from Alyosha Efros

The plenoptic function



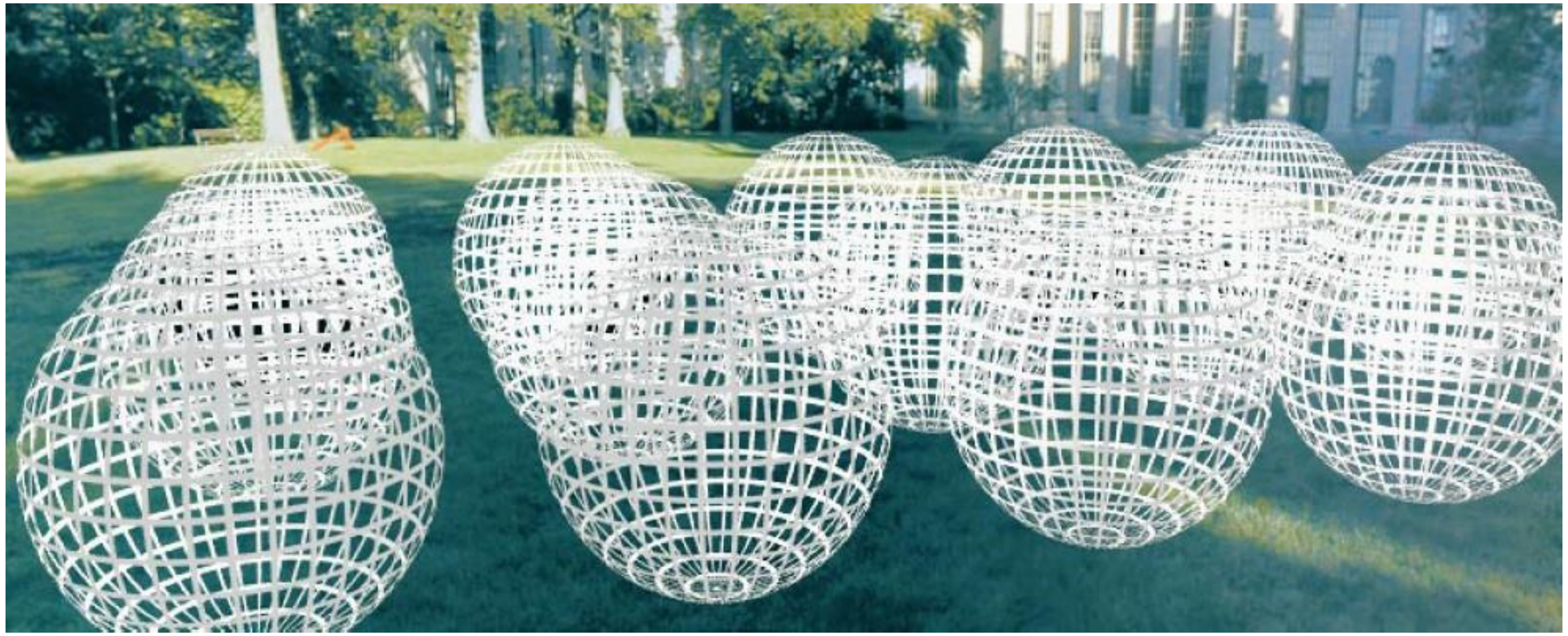
$$P(\theta,\phi,\lambda,t,V_x,V_y,V_z)$$

7D function, that can reconstruct every position & direction, at every moment, at every wavelength

= it recreates the entirety of our visual reality!

Slides from Alyosha Efros

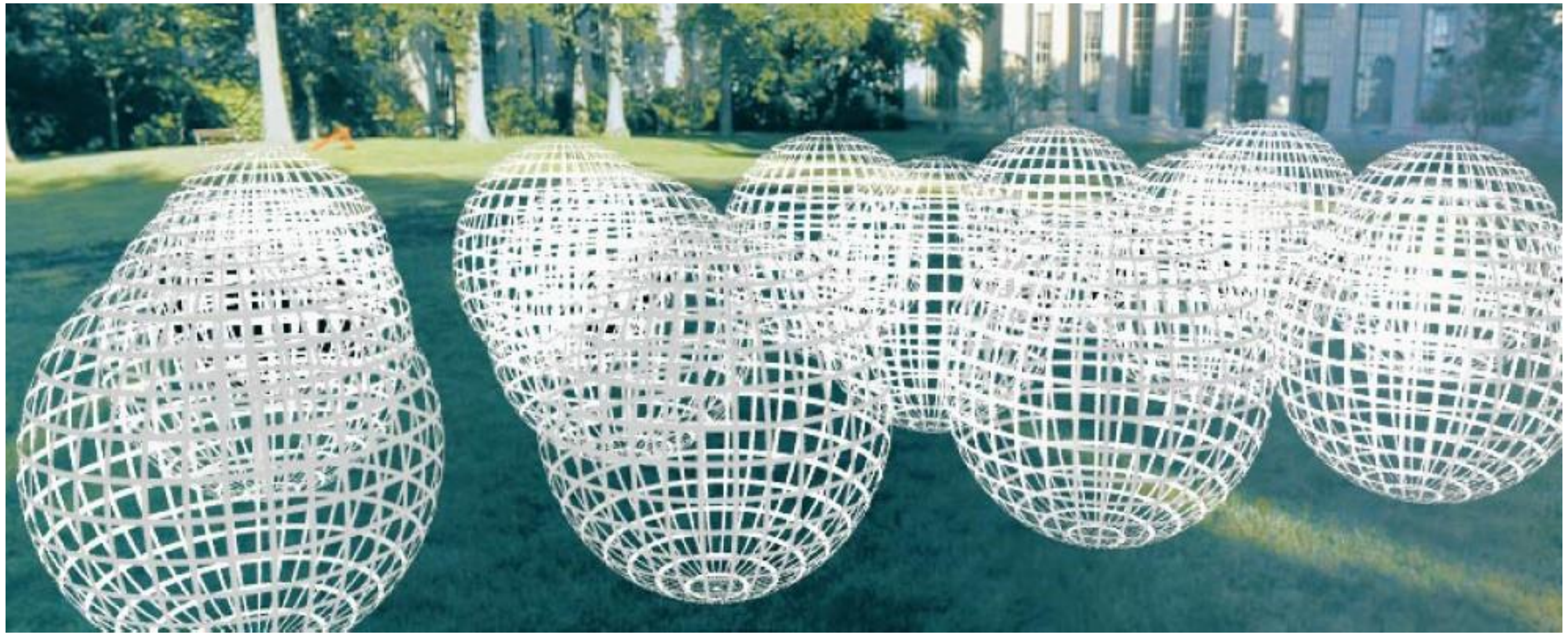
Goal: Plenoptic Function from a set of images



- Objective: Recreate the visual reality
- All about recovering photorealistic pixels, not about recording 3D point or surfaces

—Image Based Rendering aka **Novel View Synthesis**

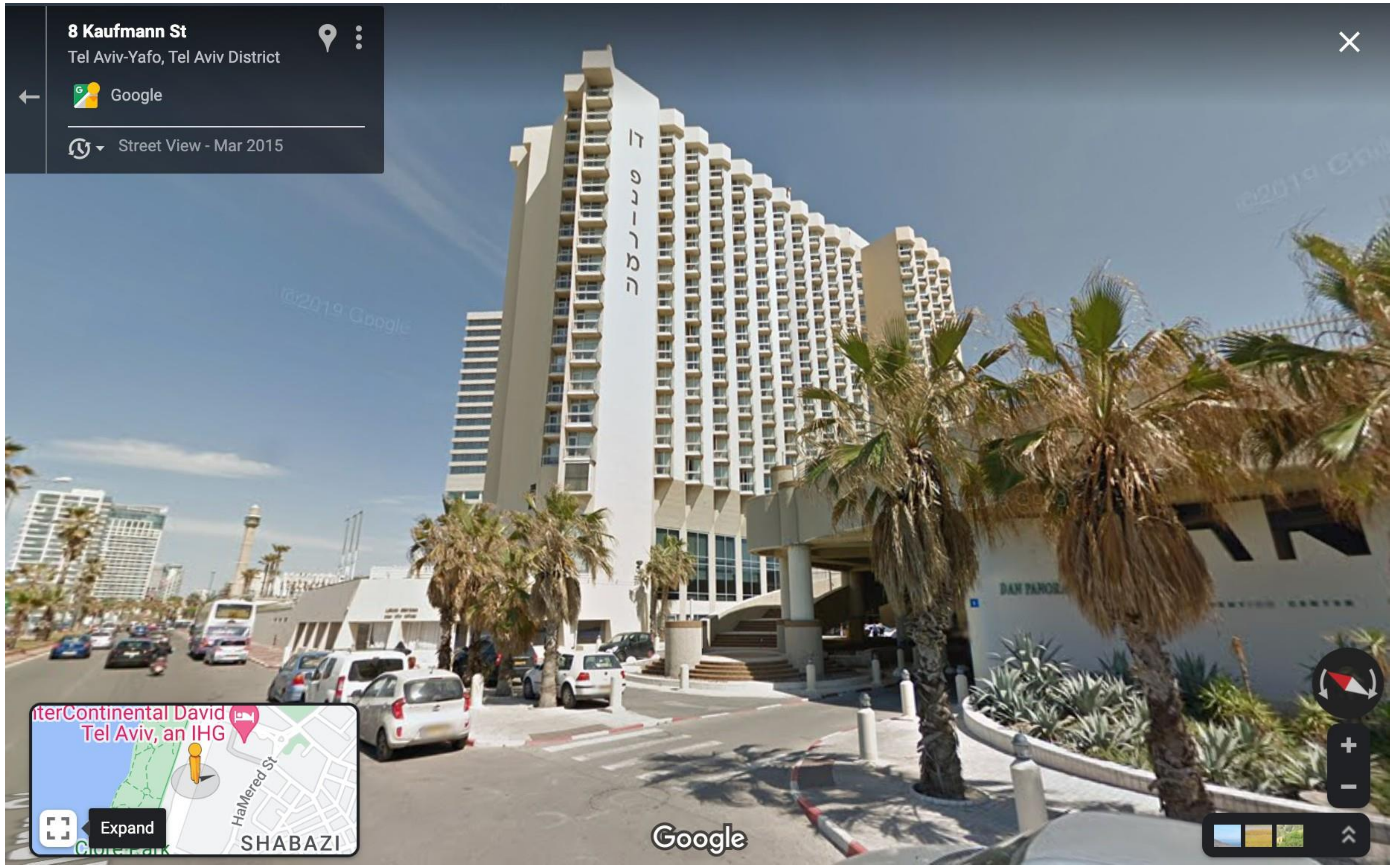
Goal: Plenoptic Function from a set of images



It is a conceptual device

Adelson & Bergen do not discuss how to solve this

An example of a sparse plenoptic function

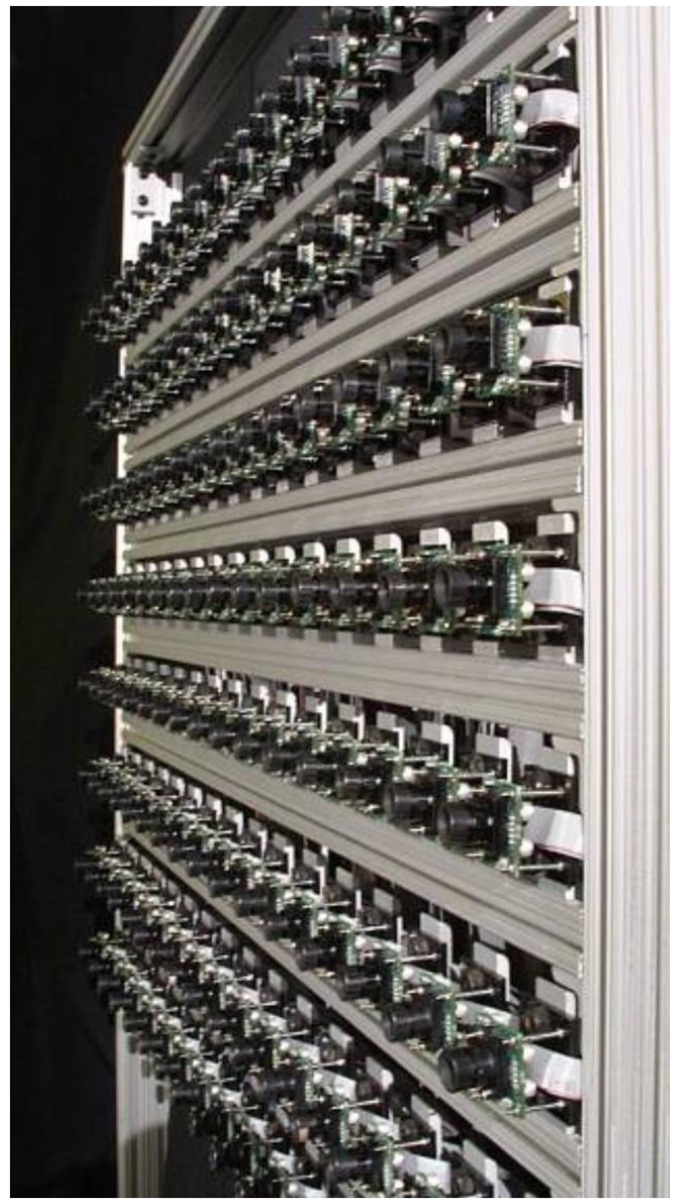


If street view was super dense
(360 view from any view point)
then it is the Plenoptic Function

Lightfield / Lumigraph

Levoy and Hanrahan, SIGGRAPH 1996
Gortler et al. SIGGRAPH 1996

- An approach for modeling the Plenoptic Function
- Take a lot of pictures from many views
- Interpolate the rays to render a novel view



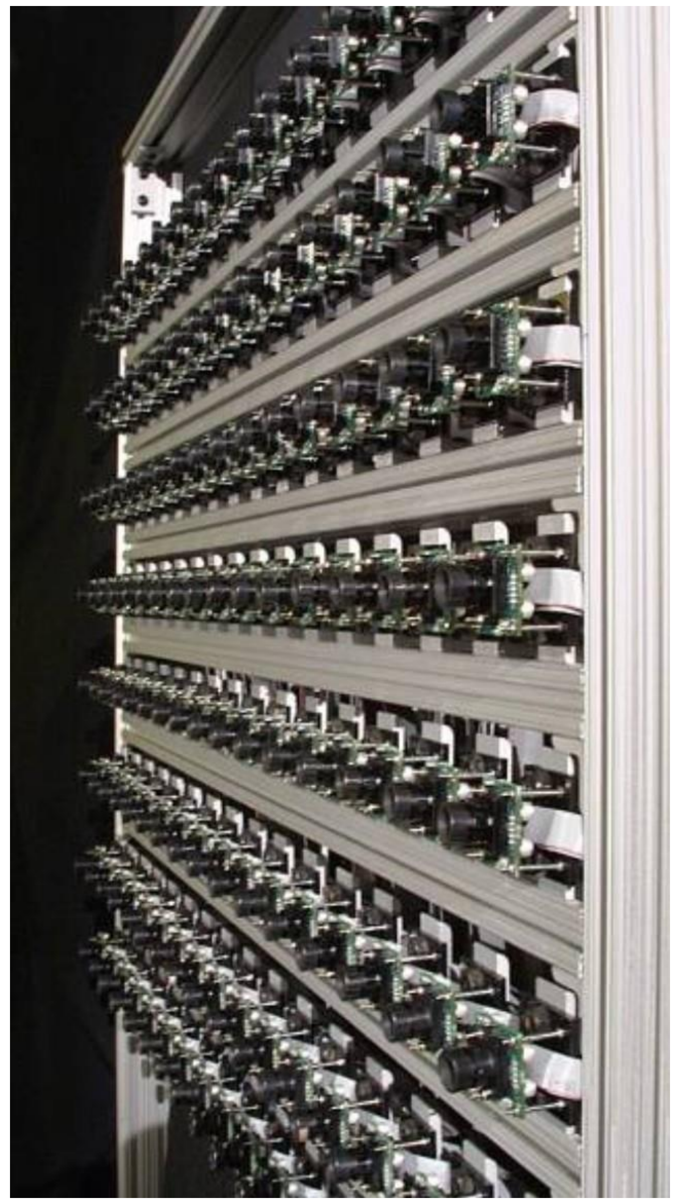
Stanford Gantry
128 cameras

Lytro camera

Levoy and Hanrahan, SIGGRAPH 1996
Gortler et al. SIGGRAPH 1996

Lightfield / Lumigraph

- An approach for modeling the Plenoptic Function
- Take a lot of pictures from many views
- Interpolate the rays to render a novel view



Stanford Gantry
128 cameras

Lytro camera

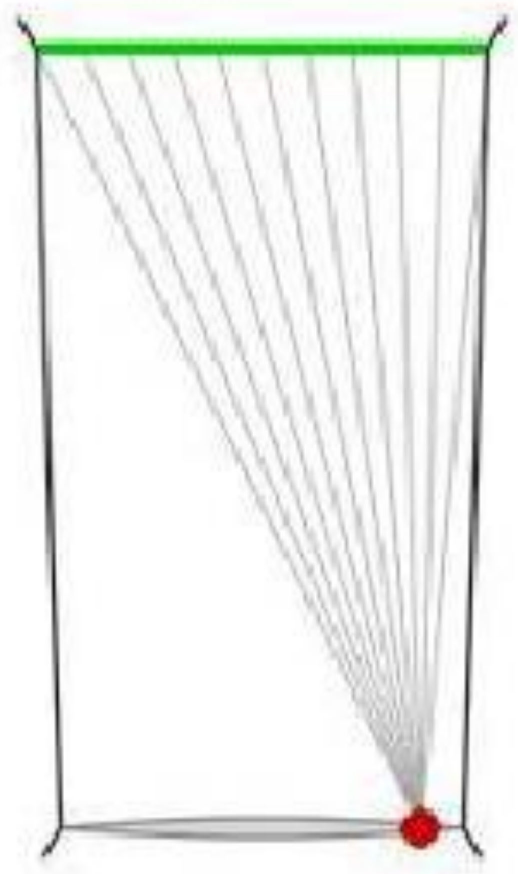
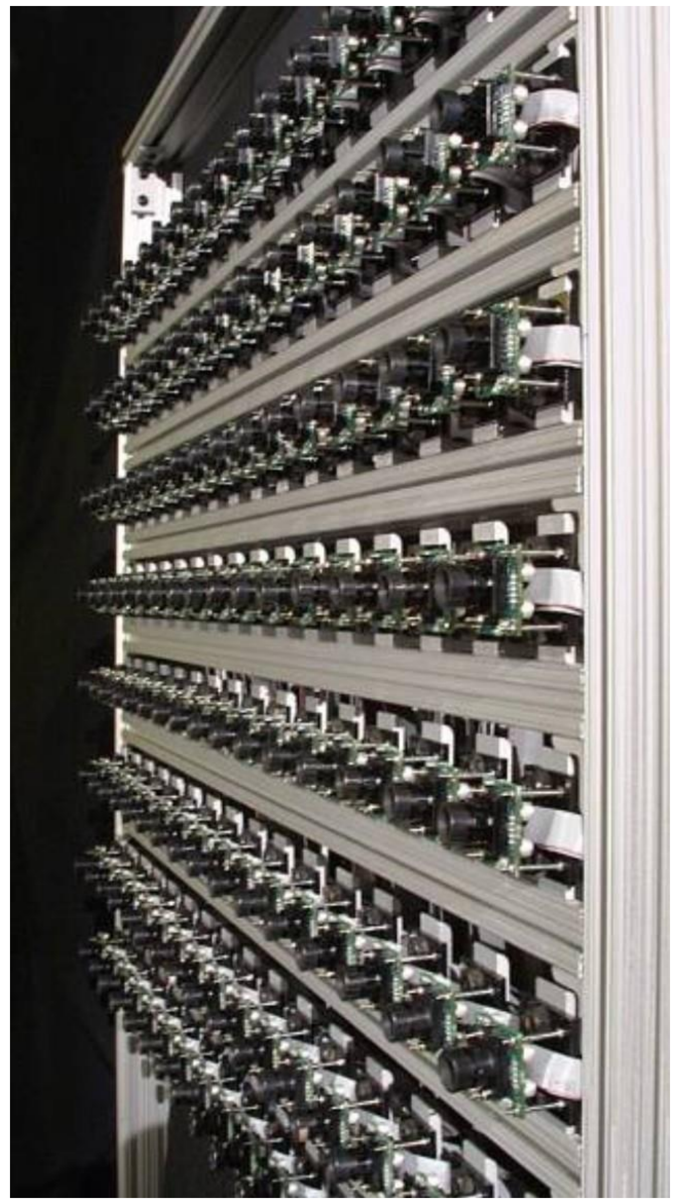


Figure from Marc Levoy

Levoy and Hanrahan, SIGGRAPH 1996
Gortler et al. SIGGRAPH 1996

Lightfield / Lumigraph

- An approach for modeling the Plenoptic Function
- Take a lot of pictures from many views
- Interpolate the rays to render a novel view



Stanford Gantry
128 cameras

Lytro camera

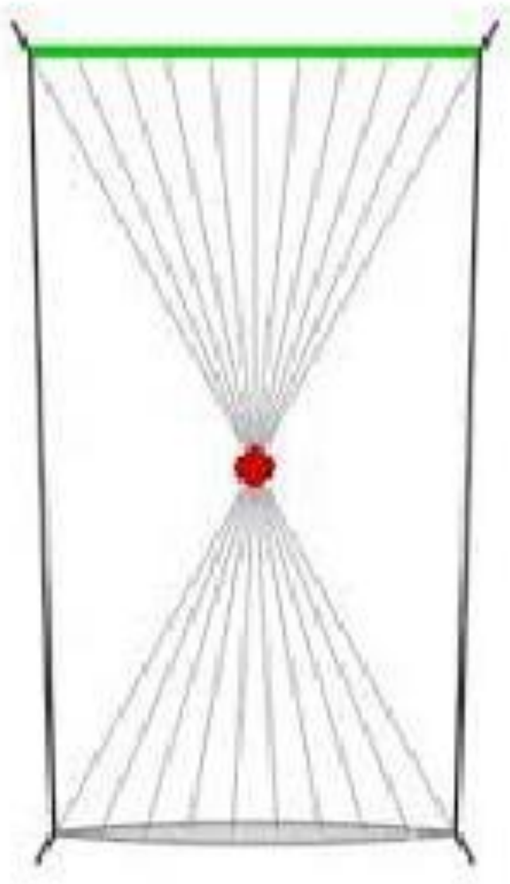
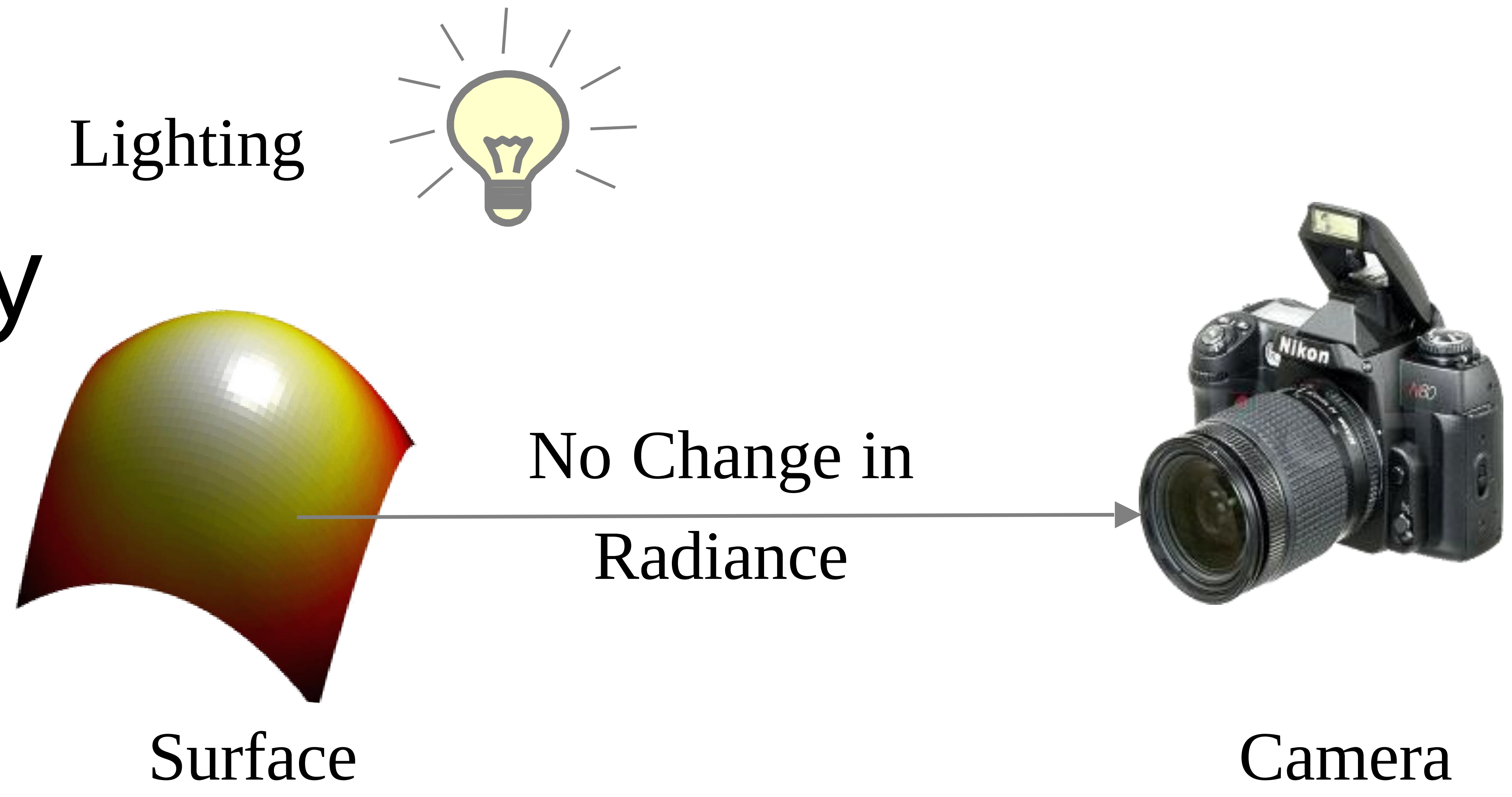


Figure from Marc Levoy

Lightfield / Lumigraph

Levoy and Hanrahan, SIGGRAPH 1996
Gortler et al. SIGGRAPH 1996

Lightfields assume that the ray shooting out from a pixel is never occluded.



Because of this it only models the plenoptic surface:

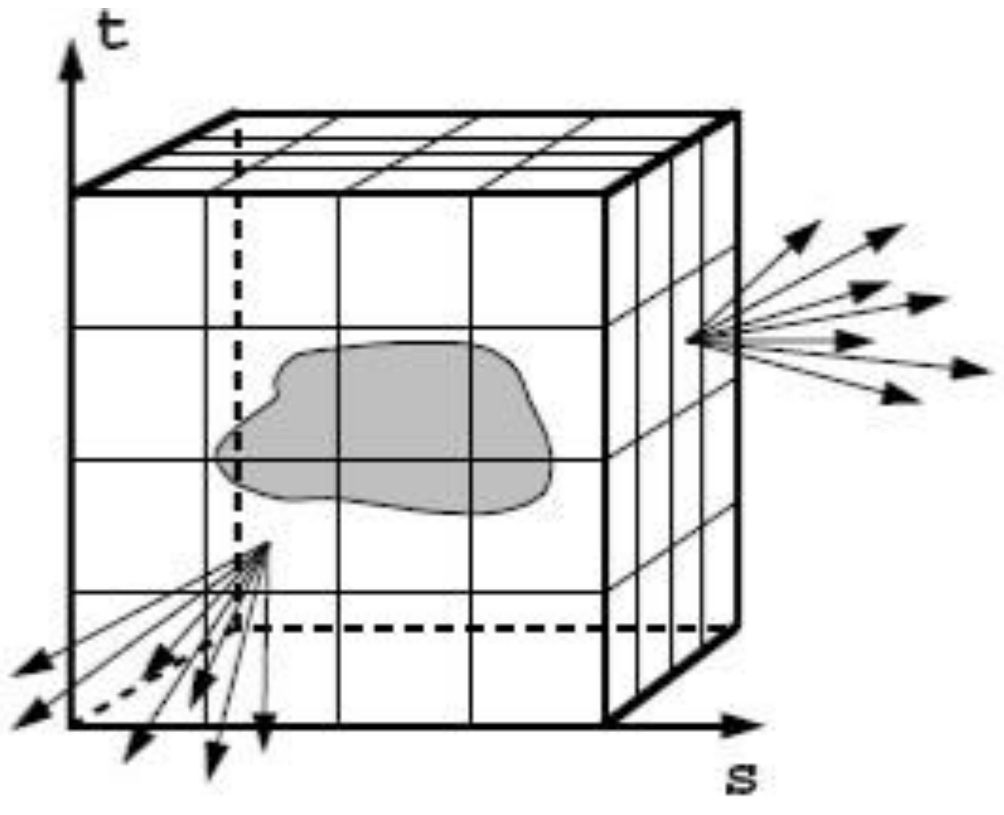


Figure 1: The surface of a cube holds all the radiance information due to the enclosed object.

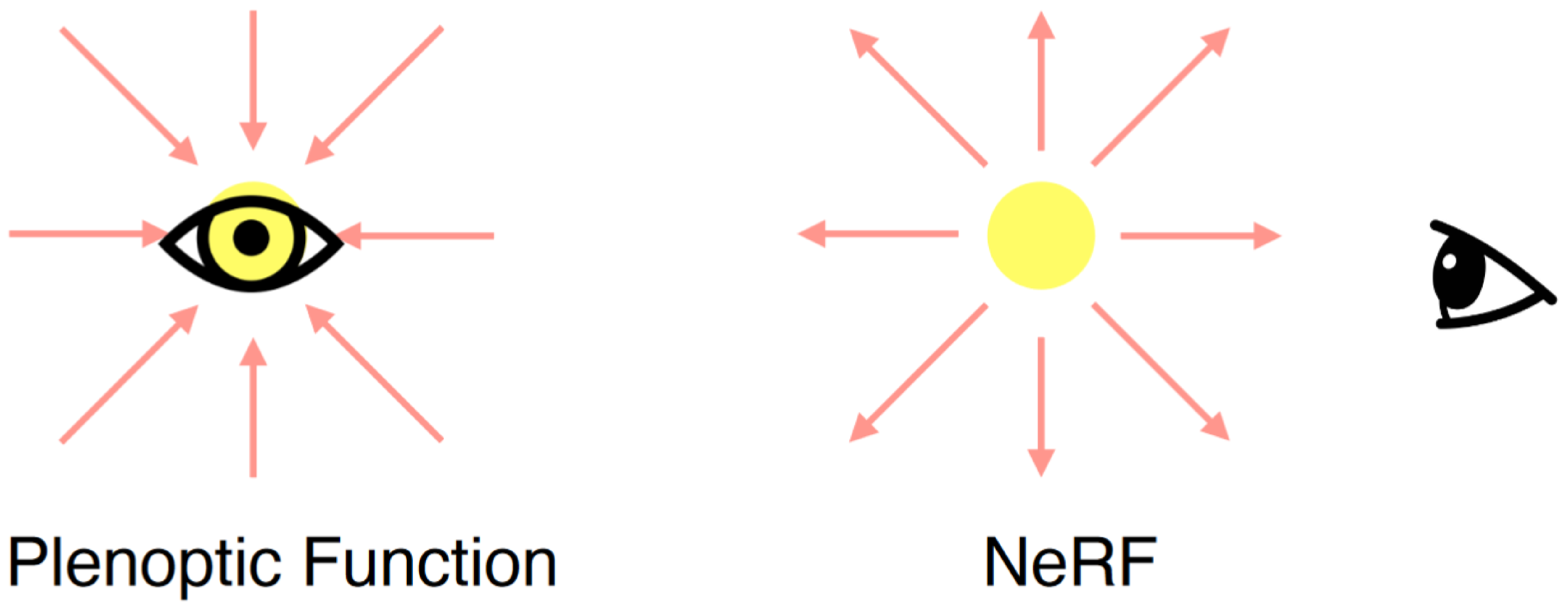
How NeRF models the Plenoptic Function

$$P(\theta,\phi,V_X,V_Y,V_Z)$$

Look familiar 🙂?

NeRF takes the same input as the Plenoptic Function!

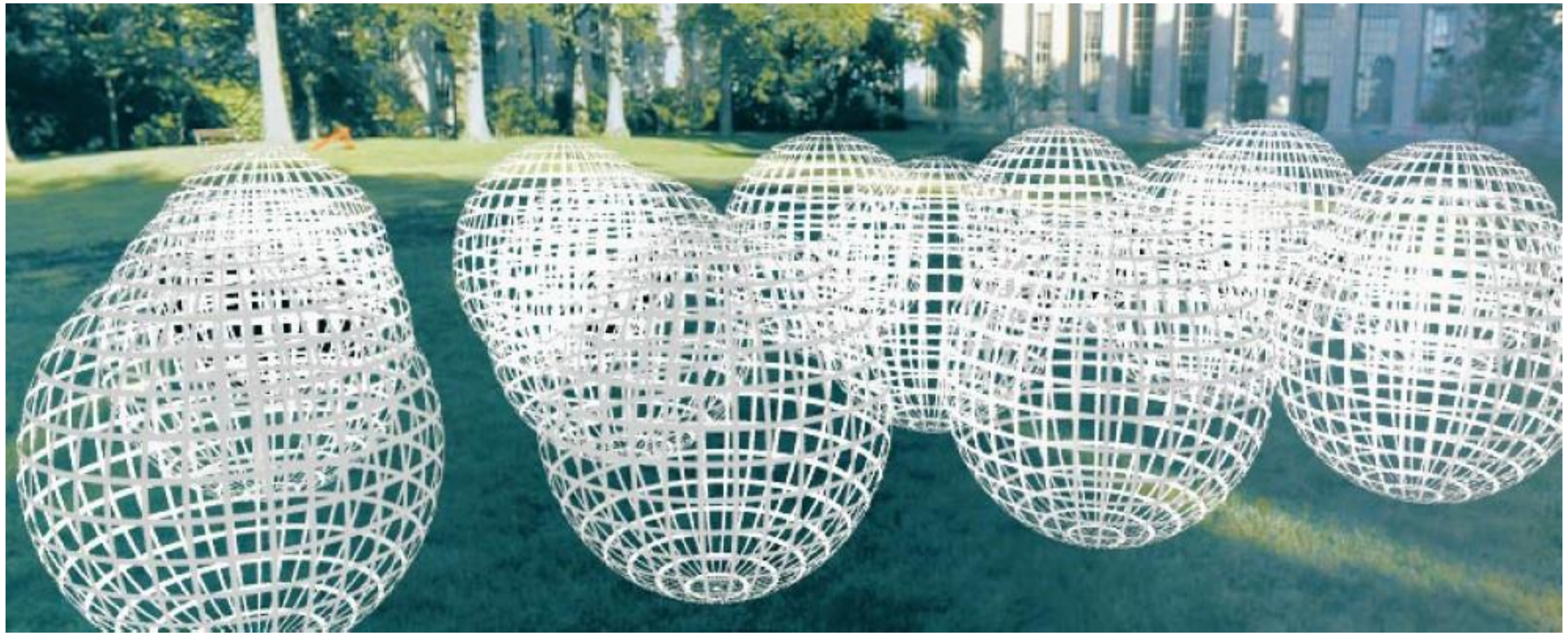
A subtle difference:



So NeRF requires the integration along the viewing ray to compute the Plenoptic Function

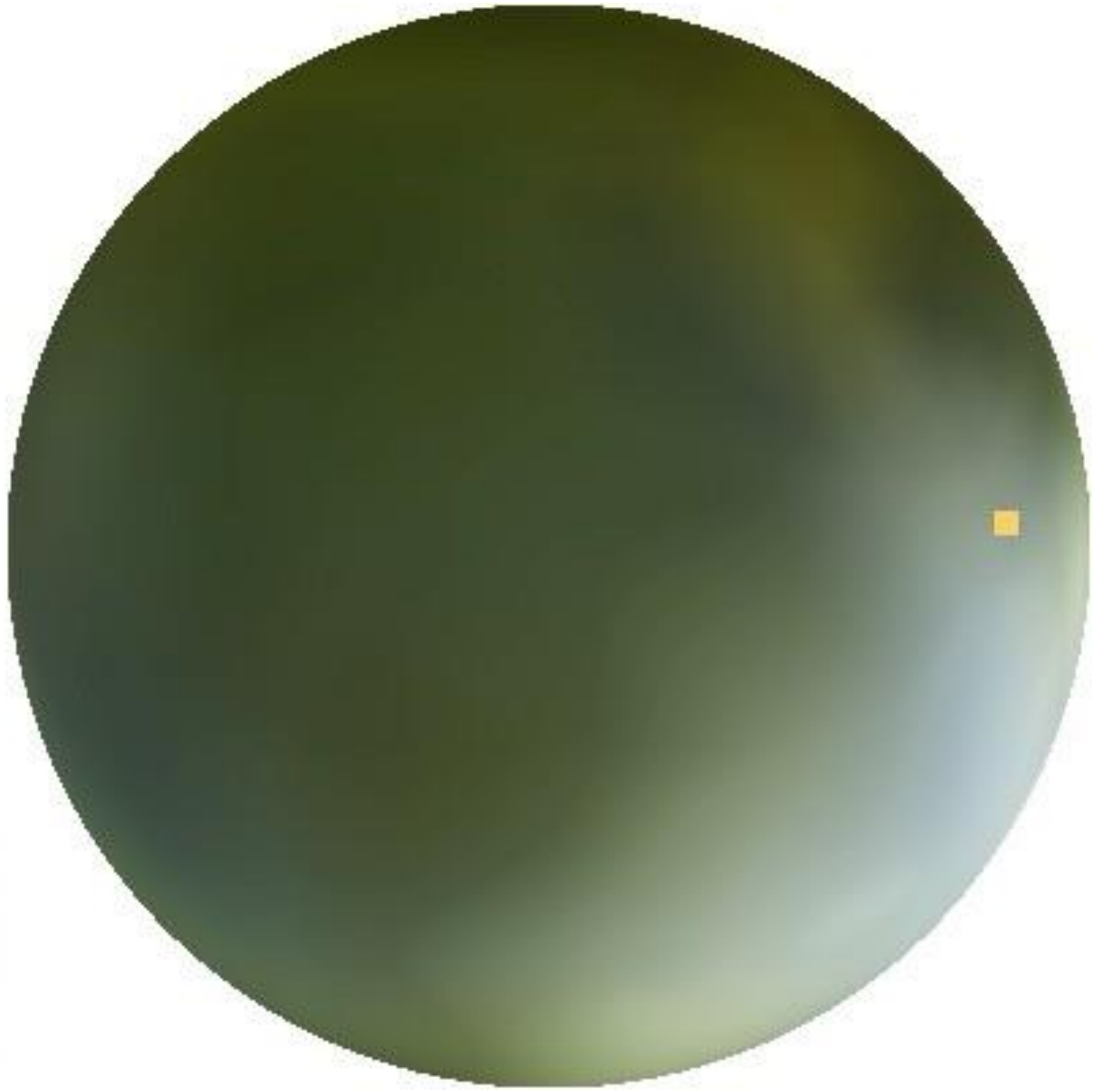
Bottom line: it models the full plenoptic function!

5D function



- For every location (3D), all possible views (2D)✨🧊✨
- NeRF models this space with a continuous view-dependent volume with opacity
- The color emitted by every point is composited to render a pixel
- Unlike a light field, the entire 5D plenoptic function can be modeled (you can fly through the world)

Visualizing the 2D function on the sphere



Outgoing radiance distribution
for point on side of ship

Outgoing radiance distribution
for point on water’s surface

Baking in Light

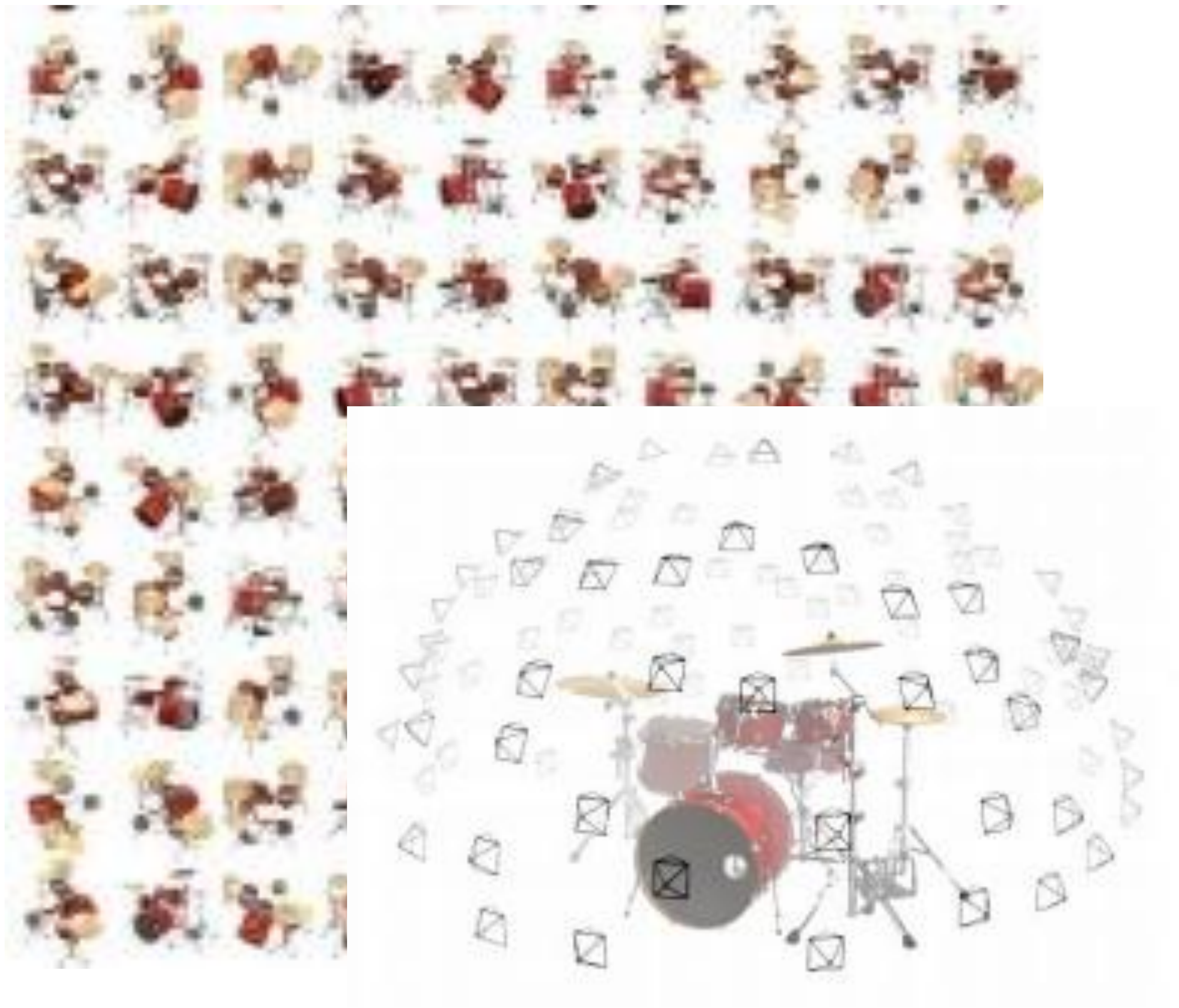
- NeRF can capture non-Lambertian (specular, shiny surfaces) because it models the color in a view-dependent manner
- This is hard to do with meshes unless you model the physical materials & lighting interactions
- But, with Image Based Rendering — All lighting effects are baked in

NeRF in a Slide

Objective: Reconstruct all training views

Volumetric 3D Scene Representation

Differentiable Volumetric Rendering Function



Optimization via Analysis-by-Synthesis

Unmentioned caveat so far

- Training a NeRF requires a **calibrated** camera!!!!

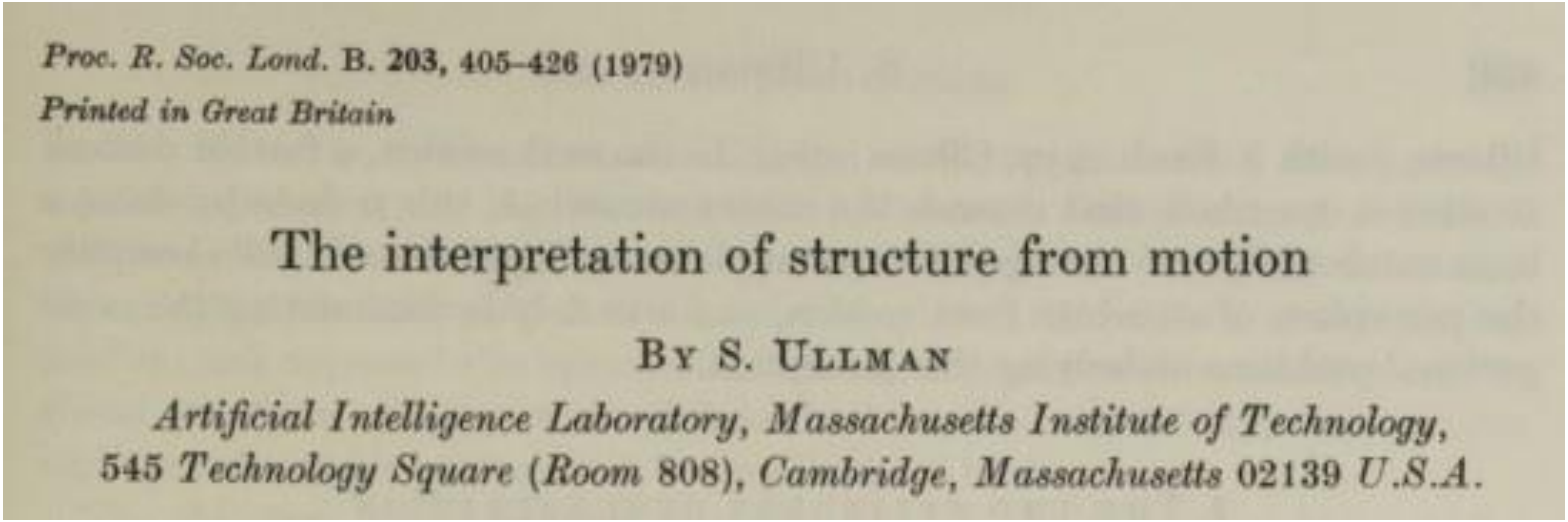
- Need to know the camera parameters: extrinsic (viewpoint) & intrinsics (focal length, distortion, etc)

How do we get this from images?

Structure from Motion

Or Photogrammetry (1850~)

Long history in Computer Vision



Proc. R. Soc. Lond. B. **203**, 405–426 (1979)
Printed in Great Britain

The interpretation of structure from motion

By S. Ullman

Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 545 Technology Square (*Room* 808), *Cambridge, Massachusetts* 02139 *U.S.A.*

NeRF is AFTER Structure from Motion

- In order to train NeRF you need to run SfM/SLAM on the images to estimate the camera parameters
- In this sense, the problem category is same as that of **Multi-view Stereo**

Colmap: Schönberger et al. 2016

Where NeRF stands

- can do Image Based Rendering well, while also being a 3D representation
- Does not suffer from limitations of surface models
- Easy to optimize from images

Appearance Based Reconstruction (Image Based Rendering)

NeRFs

Physics based Reconstruction (3D Surface Modeling)

Lightfield/Lumigraph
(No 3D representation)

Layered Depth Images (LDIs)

Multi-Plane Images (MPIs)

One 3D Surface, View-Dependent Texture Mapping

One 3D Surface, Single Albedo Texture

Conventional Graphics Pipeline

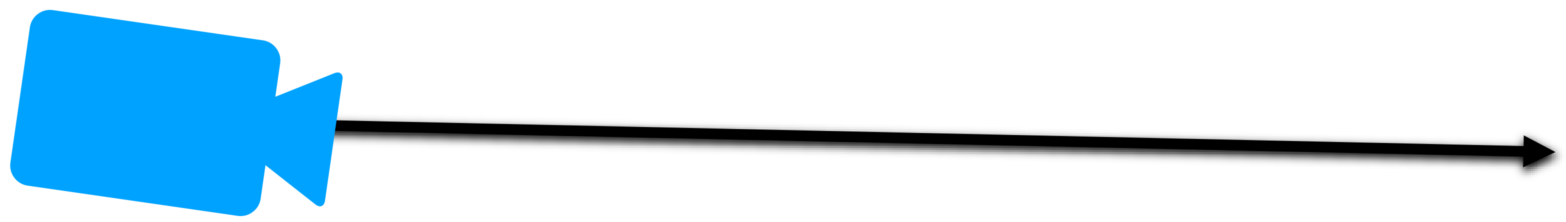
Analysis by Synthesis Requires Differentiable Renderers

Next: Deep dive into Volumetric Rendering Function

Neural Volumetric Rendering

Neural Volumetric Rendering

computing color along rays
through 3D space



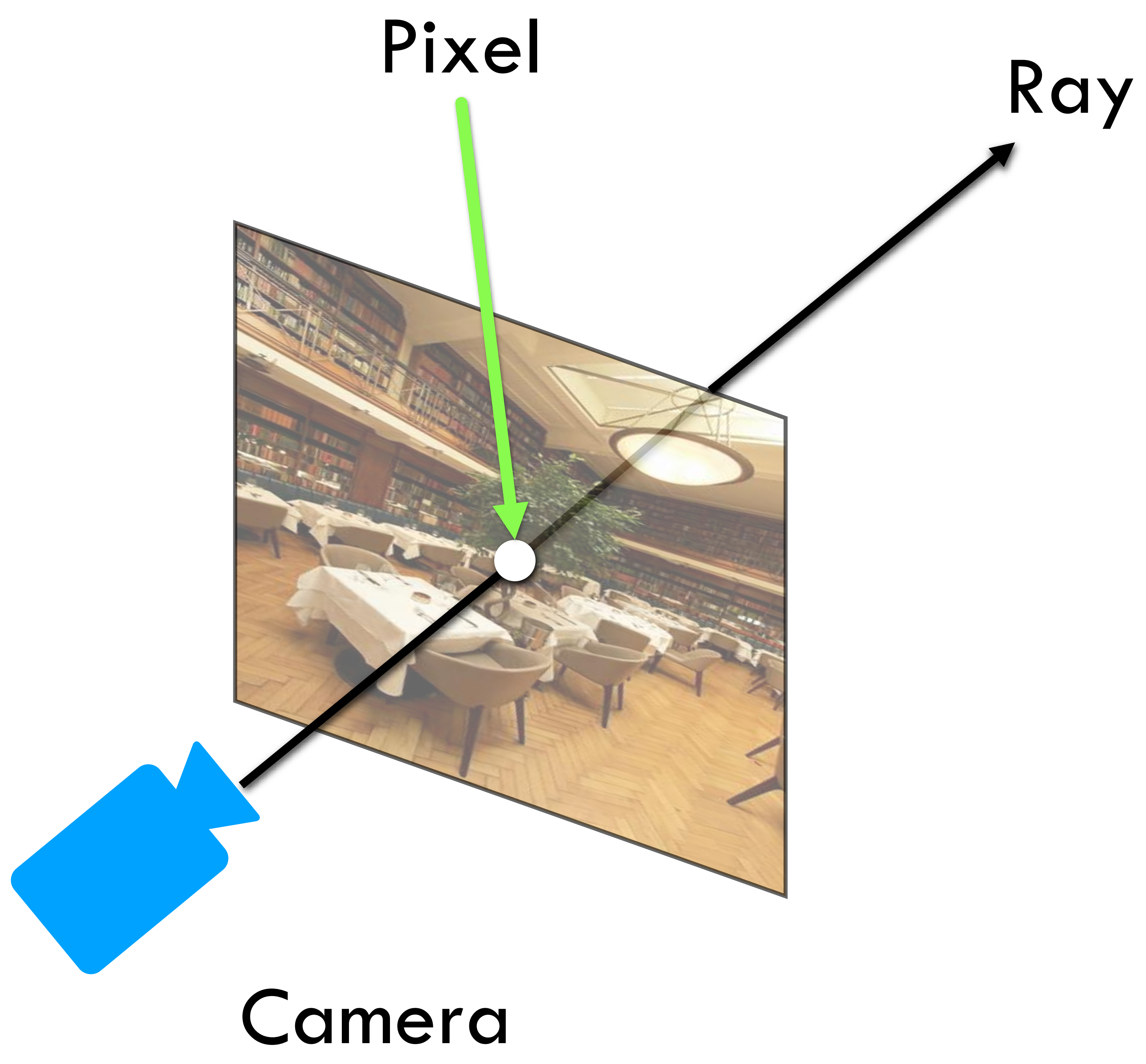
What color is this pixel?

Cameras and rays

Cameras and rays

- We need the mathematical mapping from (*camera*, *pixel*) → *ray*

- Then can abstract underlying problem as learning the function *ray* → *color* (the “plenoptic function”)



Coordinate frames + Transforms: world-to-camera

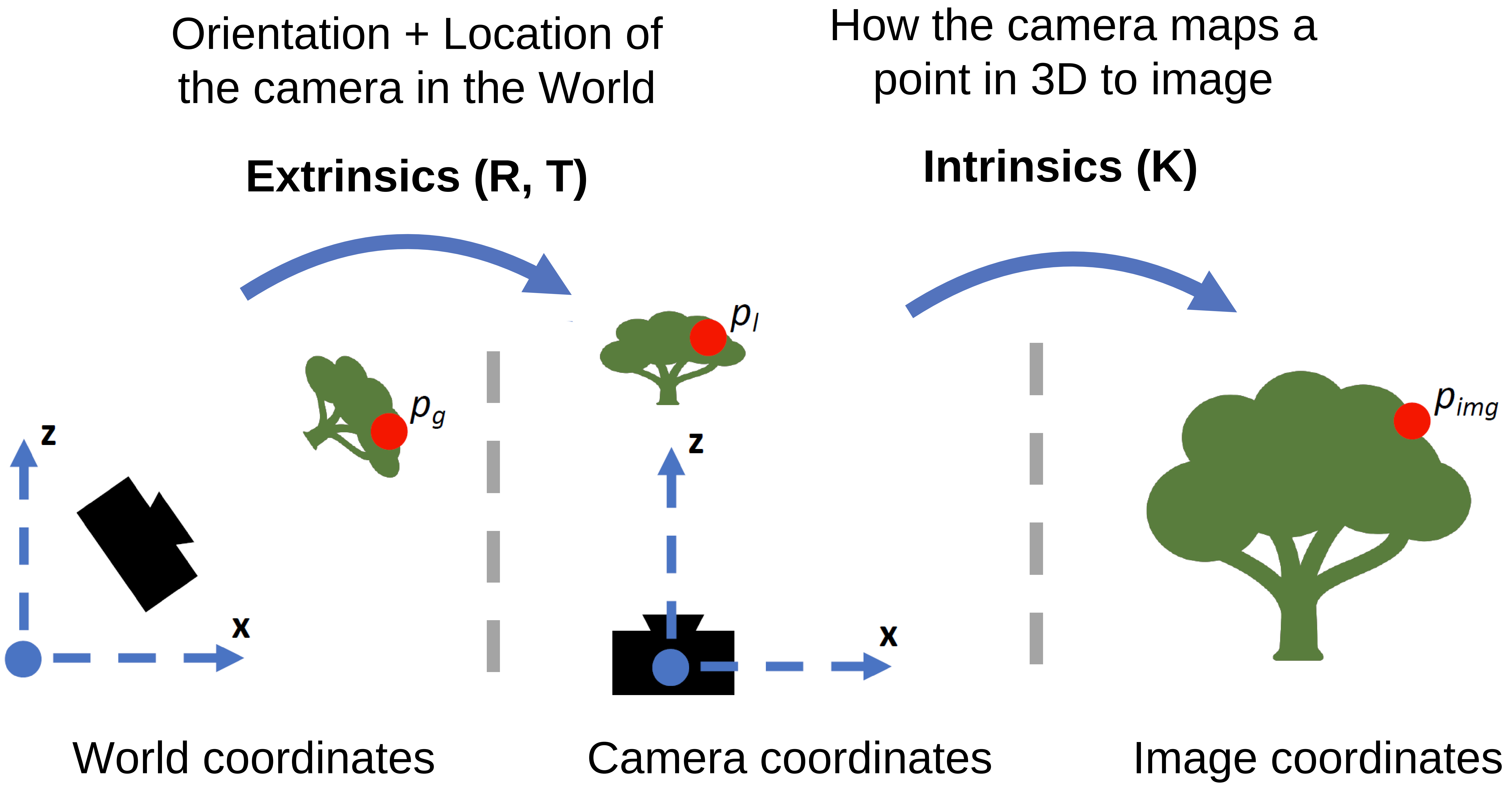


Figure credit: Peter Hedman

Coordinate frames + Transforms: camera-to-world

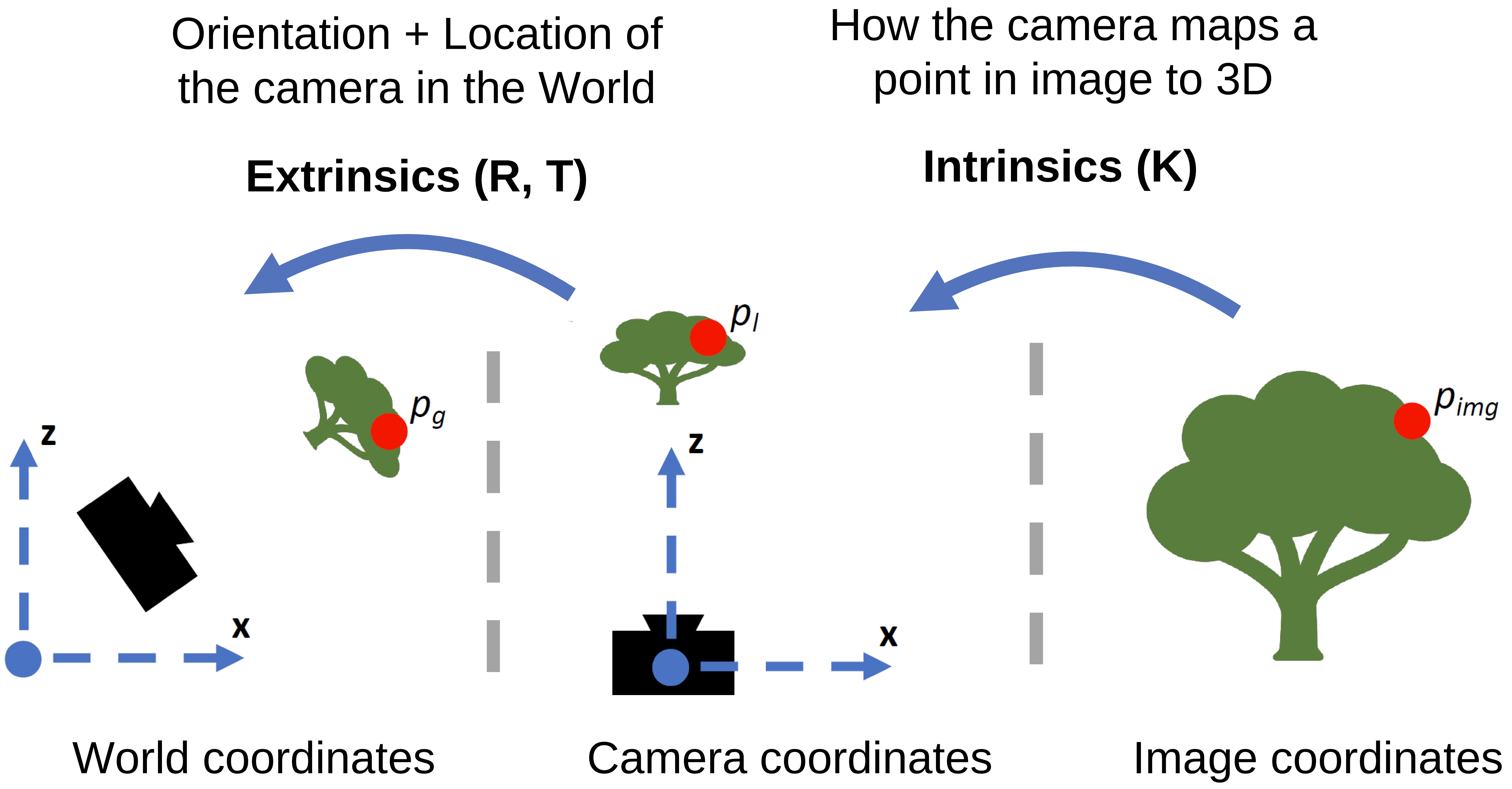
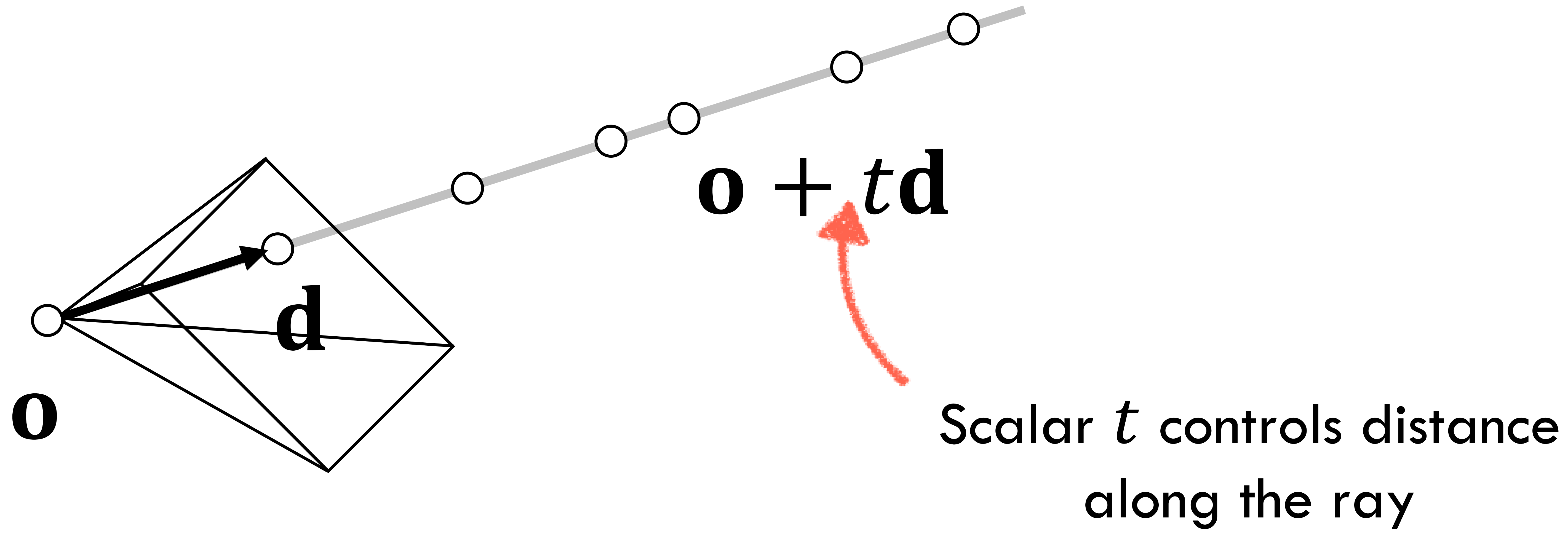


Figure credit: Peter Hedman

Calculating points along a ray

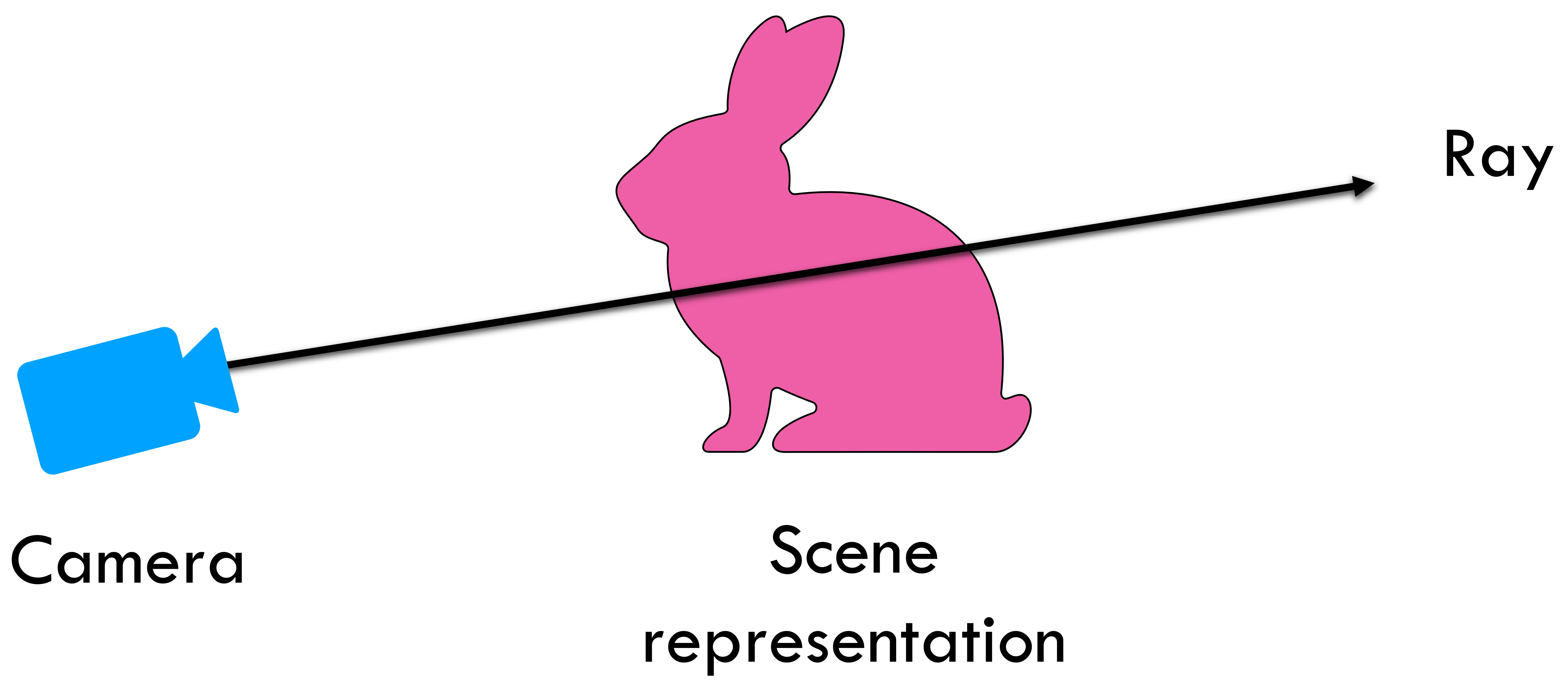


Neural Volumetric Rendering

Neural Volumetric Rendering

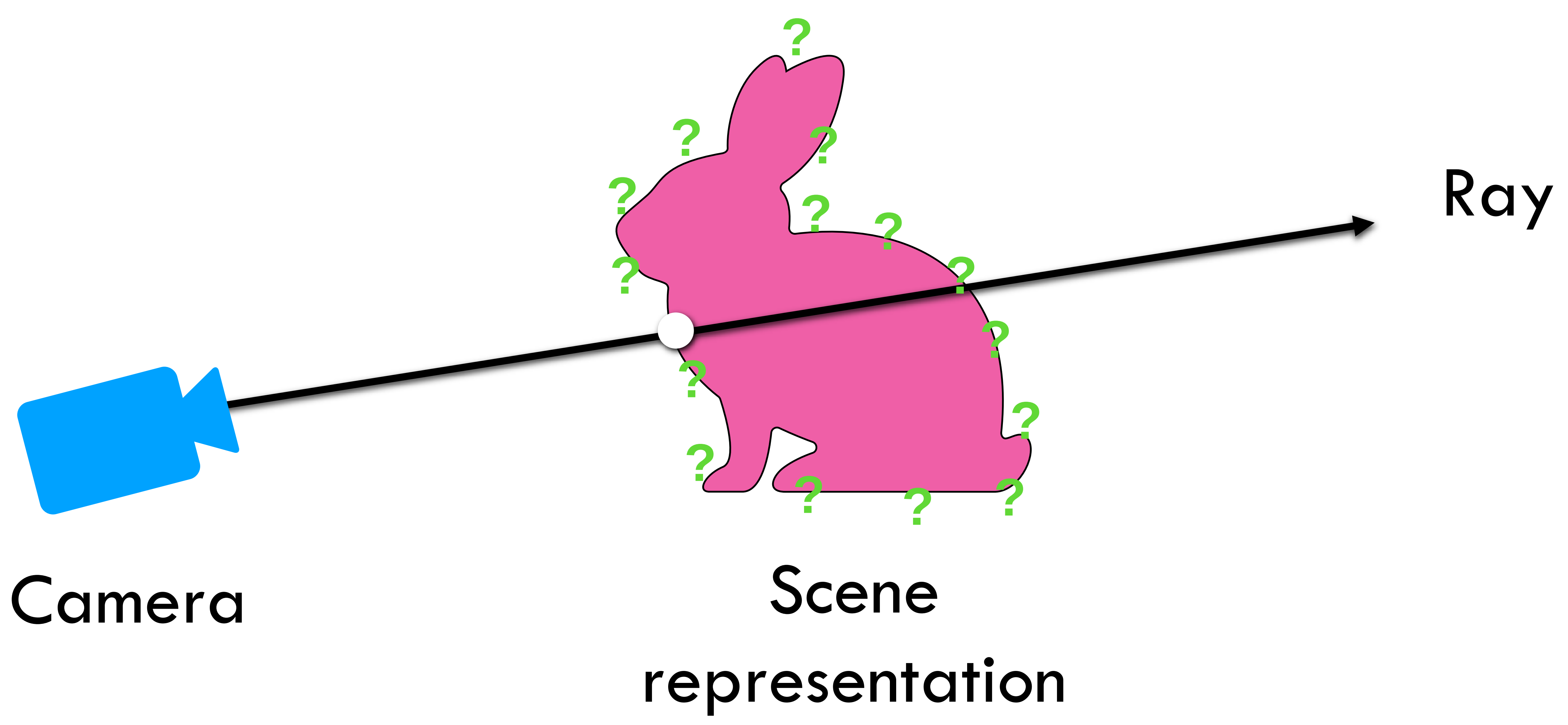
continuous, differentiable rendering model without concrete ray/surface intersections

Surface vs. volume rendering



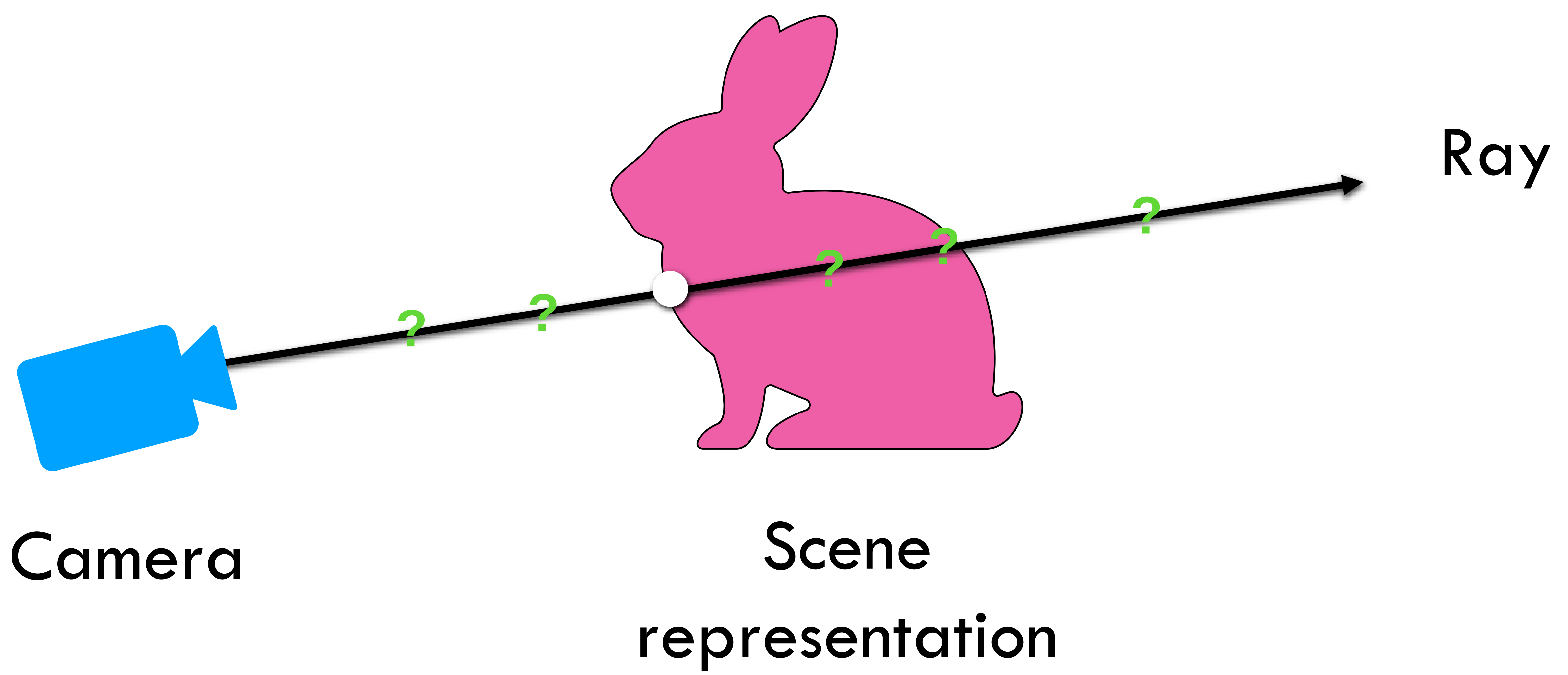
Want to know how ray interacts with scene

Surface vs. volume rendering



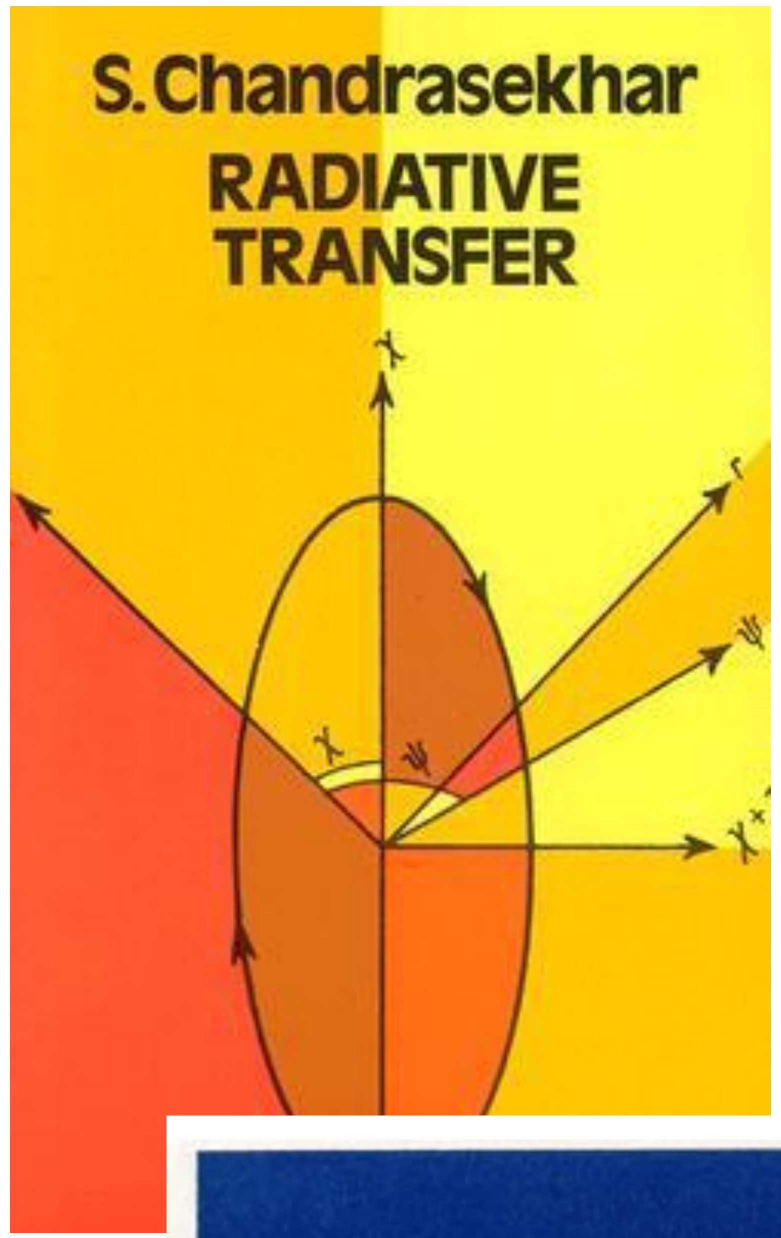
Surface rendering — loop over geometry, check for ray hits

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]

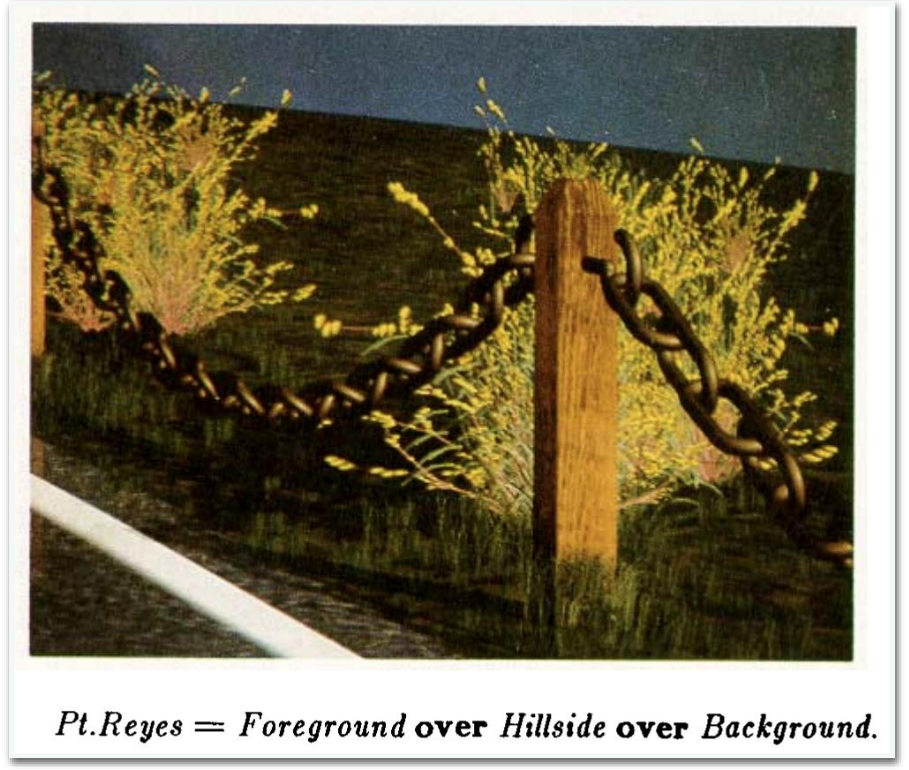
Chandrasekhar 1950, Radiative Transfer

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

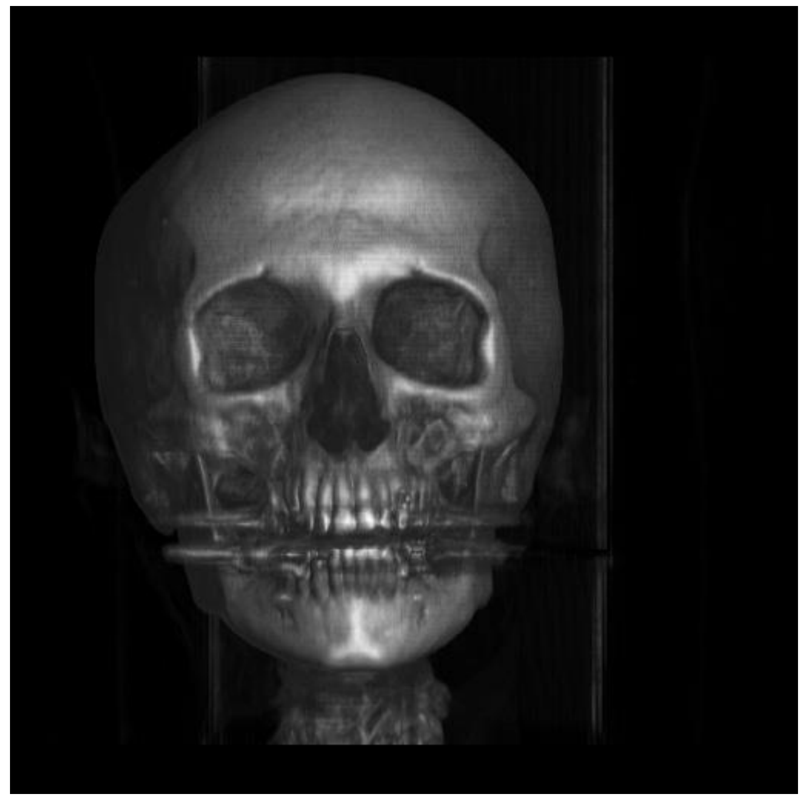
- Alpha rendering developed for digital compositing in VFX movie production



Alpha compositing [Porter and Duff]

Porter and Duff 1984, Compositing Digital Images

Volume rendering for visualization



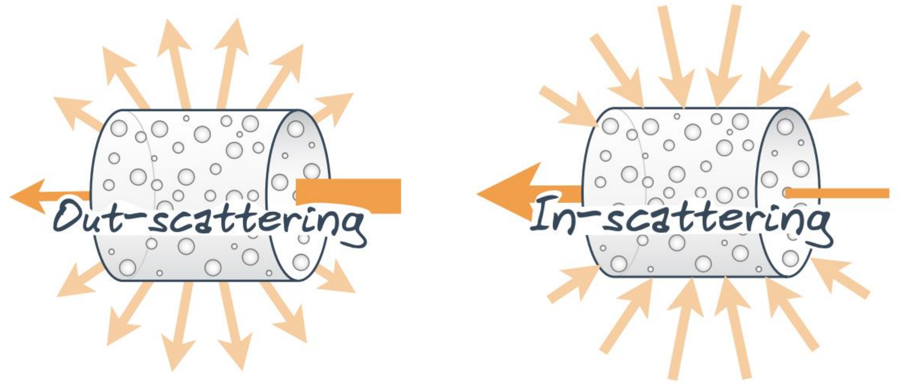
Medical data visualisation [Levoy]

- 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

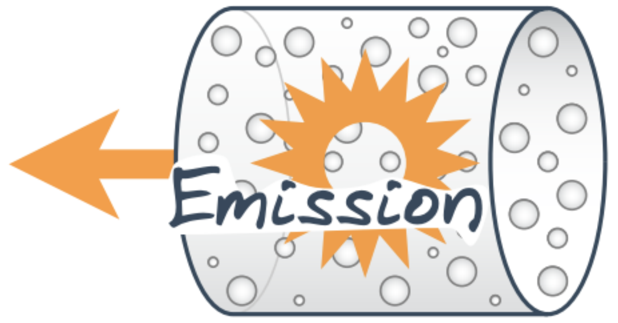
Chandrasekhar 1950, Radiative Transfer
Kajiya 1984, Ray Tracing Volume Densities
Porter and Duff 1984, Compositing Digital Images
Levoy 1988, Display of Surfaces from Volume Data
Max 1995, Optical Models for Direct Volume Rendering

Volume rendering derivations

Absorption

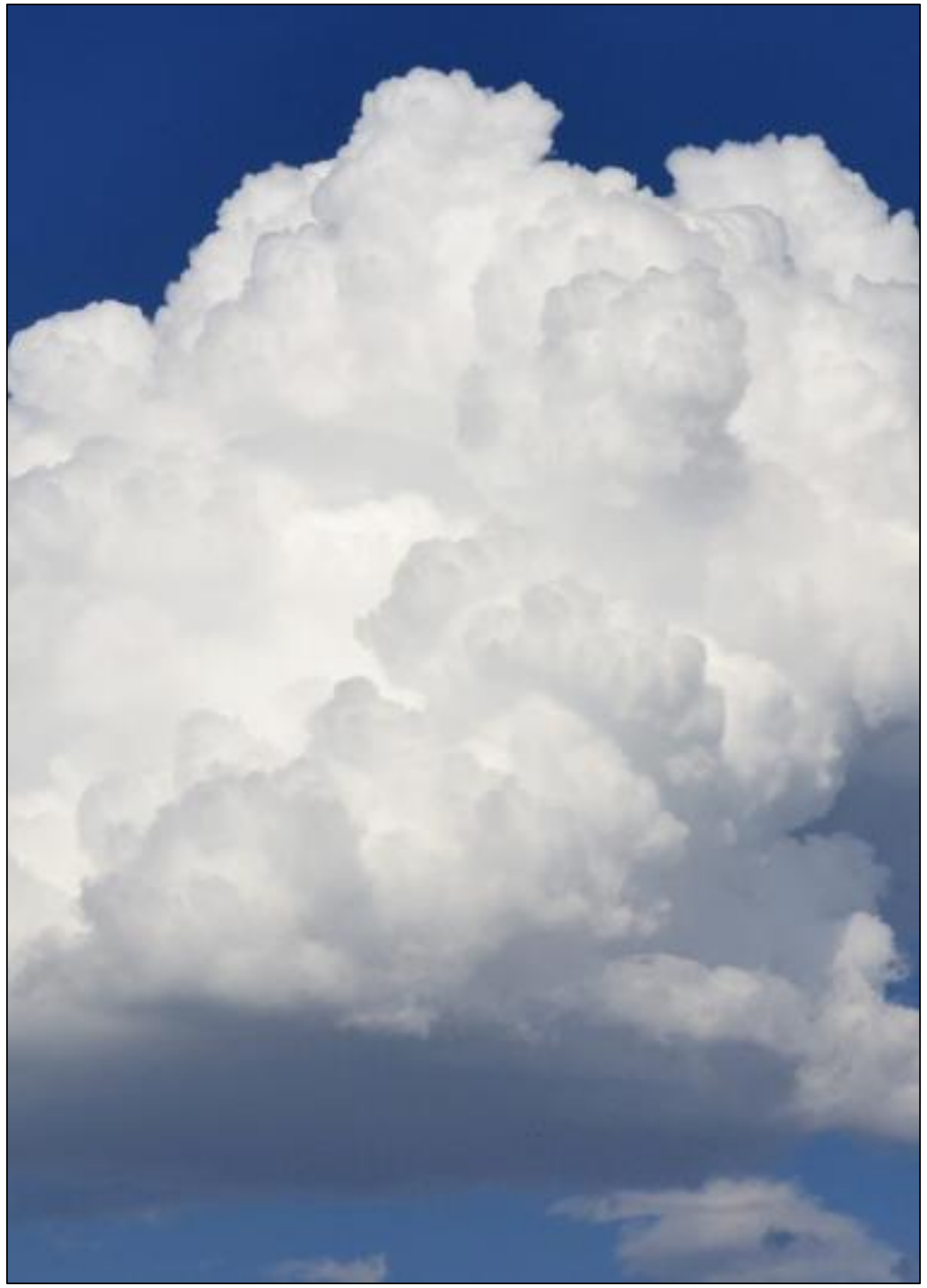


Scattering



Emission

http://commons.wikimedia.org



http://wikipedia.org

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

Simplify

Absorption

http://commons.wikimedia.org

Scattering

Emission

http://wikipedia.org

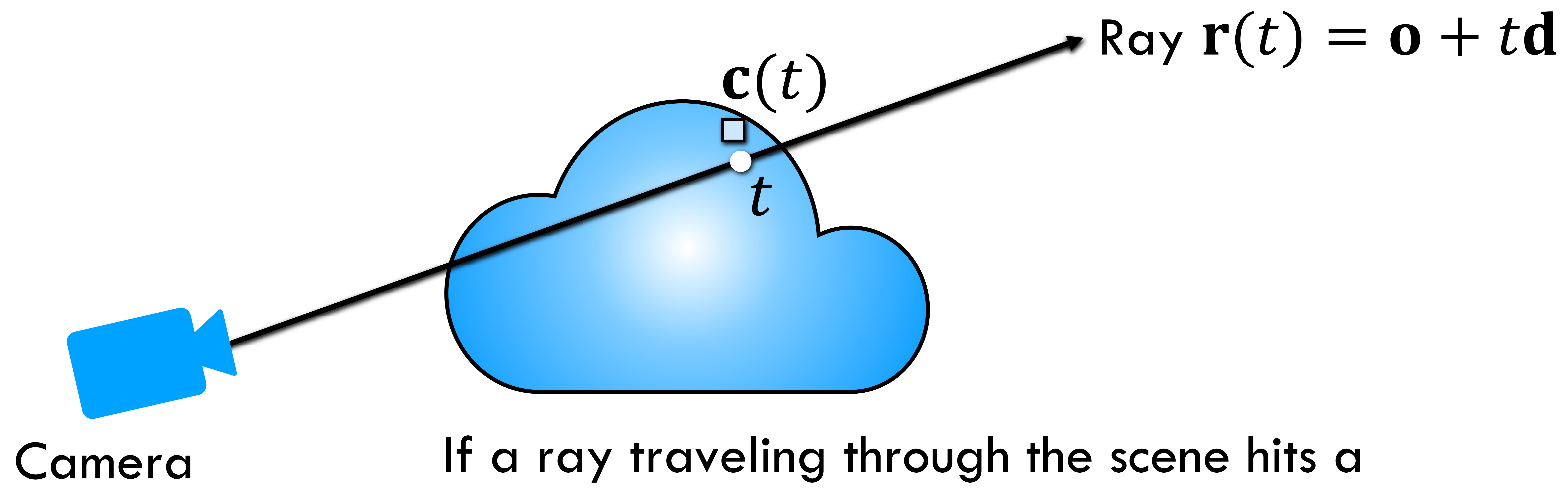
Slide credit: Novak et al 2018, Monte Carlo methods for physically based volume rendering

Volumetric formulation for NeRF

Scene is a cloud of tiny colored particles

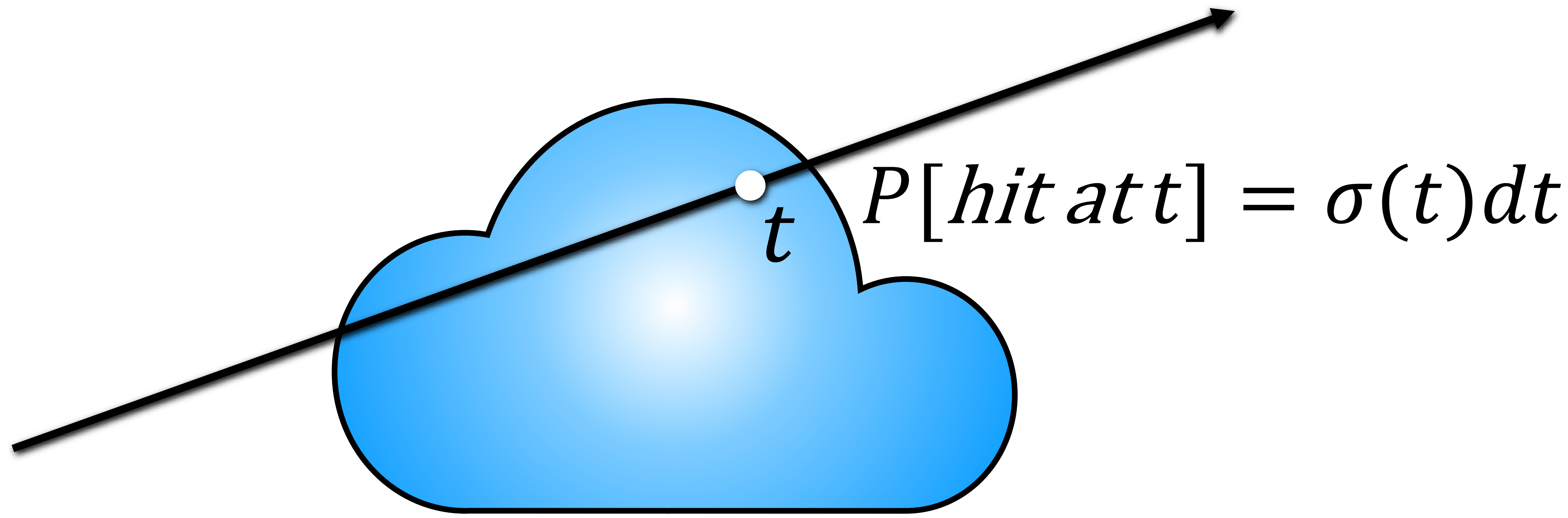
Max and Chen 2010, Local and Global Illumination in the Volume Rendering Integral

Volumetric formulation for NeRF



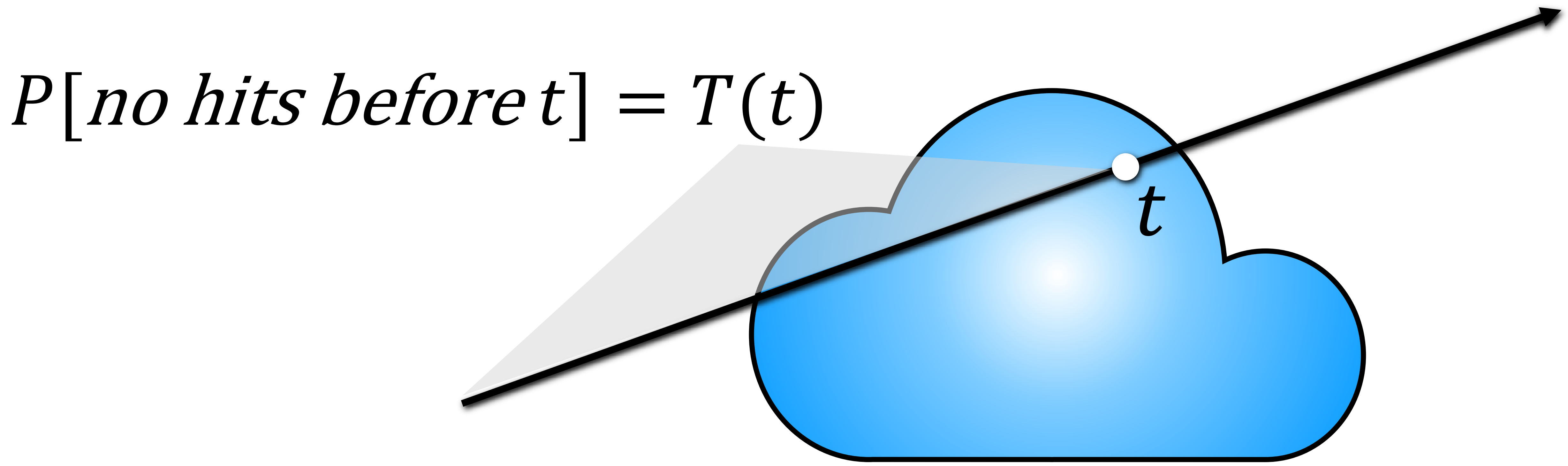
If a ray traveling through the scene hits a particle at distance t along the ray, we return its color $\mathbf{c}(t)$

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



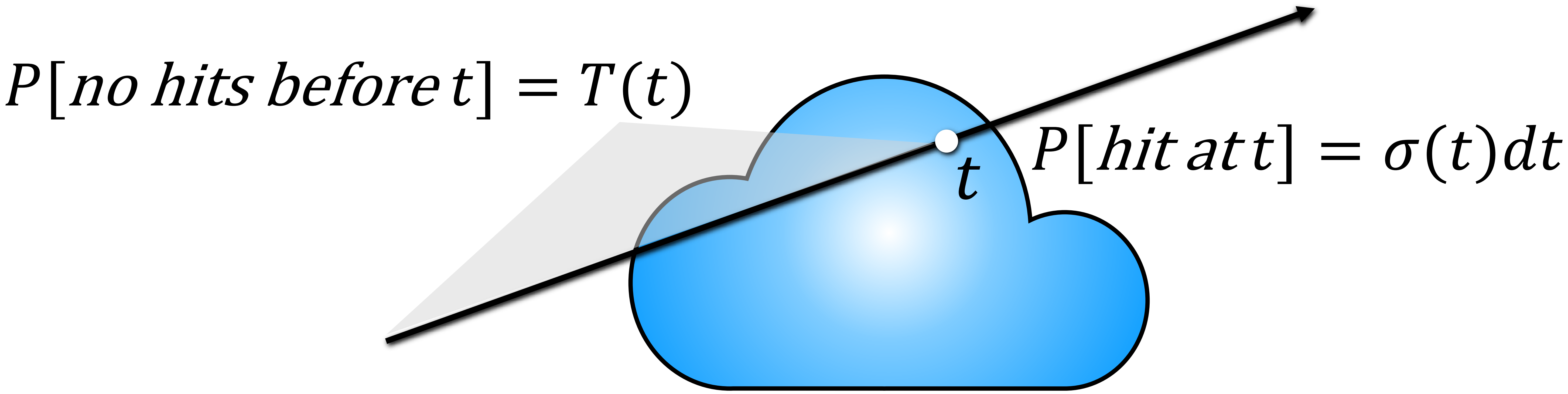
This notion is probabilistic: chance that ray hits a particle in a small interval around t is $\sigma(t)dt$.
σ is called the "volume density"

Probabilistic interpretation



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"

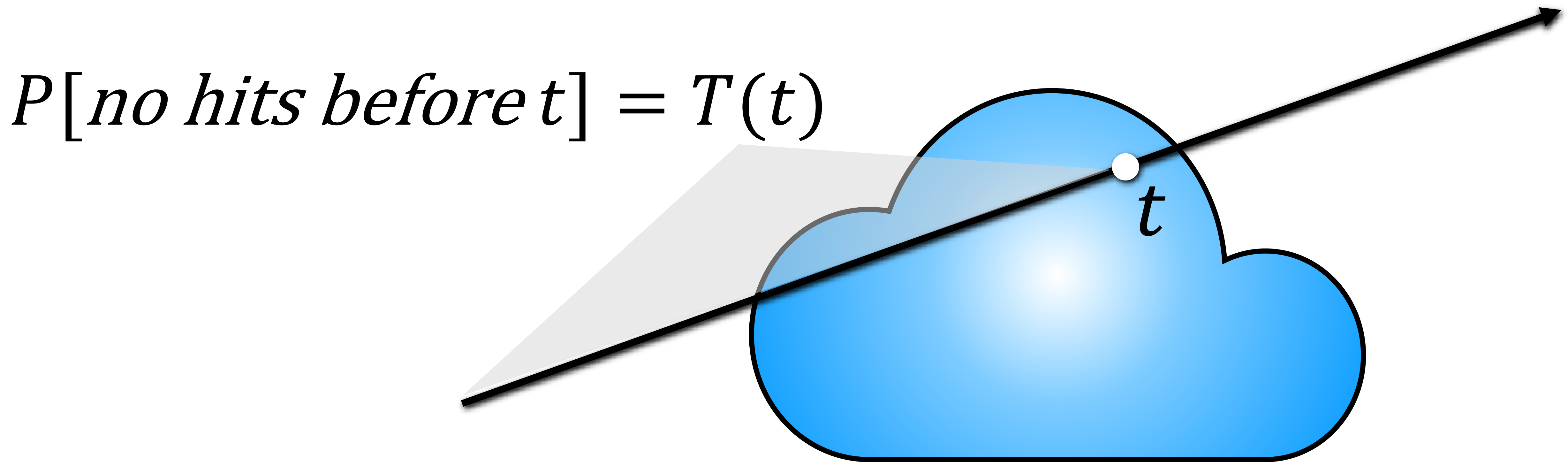
Probabilistic interpretation



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

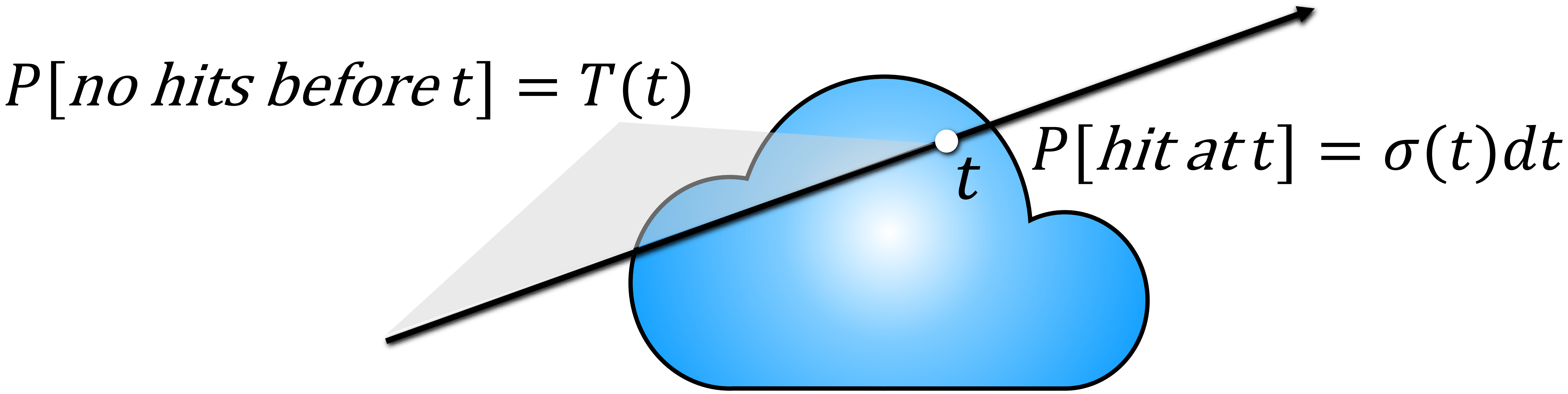
$P[first\ hit\ at\ t] = P[no\ hit\ before\ t] \times P[hit\ at\ t]$
$= T(t)\sigma(t)dt$

Calculating T given σ



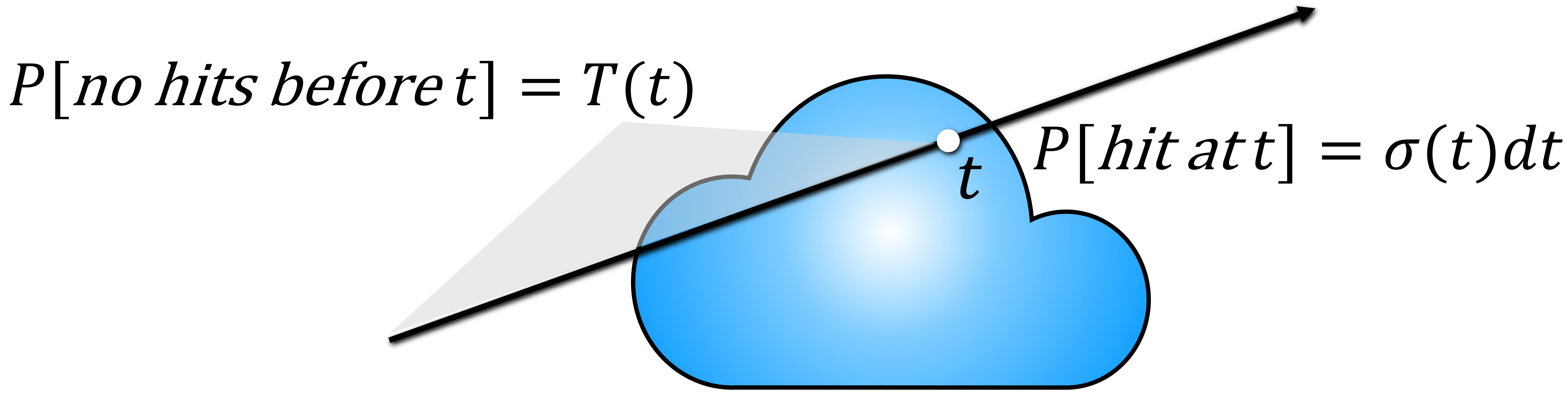
If σ is known, T can be computed… How?

Calculating T given σ



σ and T are related by the probabilistic fact that
$P[no\ hit\ before\ t + dt] = P[no\ hit\ before\ t] \times P[no\ hit\ at\ t]$

Calculating transmittance 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

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

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

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

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

Solve for T

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

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

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

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

Solve for T

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

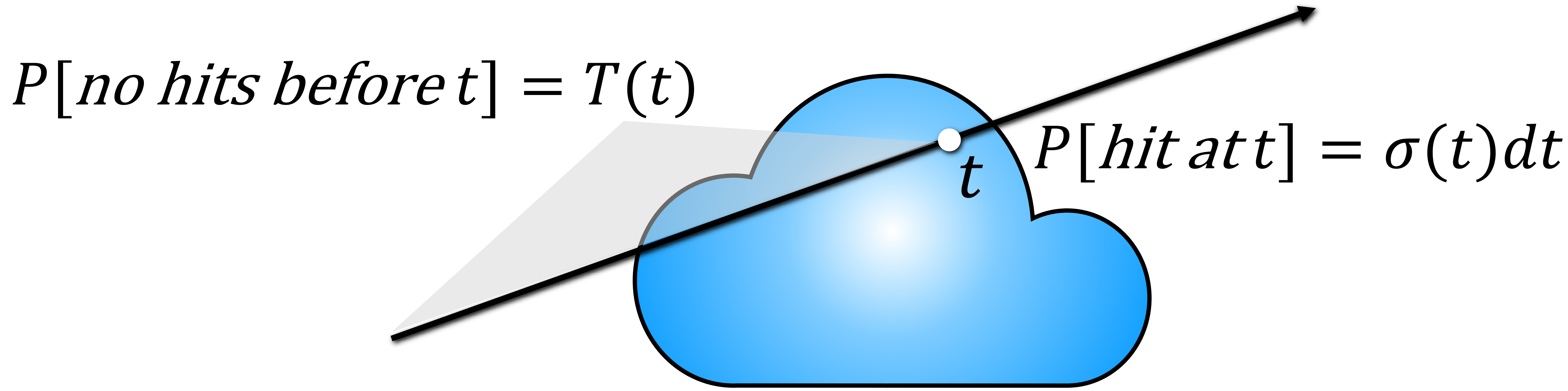
Taylor expansion for T$\Rightarrow T(t) + T'(t)dt = T(t) - T(t)\sigma(t)dt$

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

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

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

PDF for ray termination



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

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

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

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

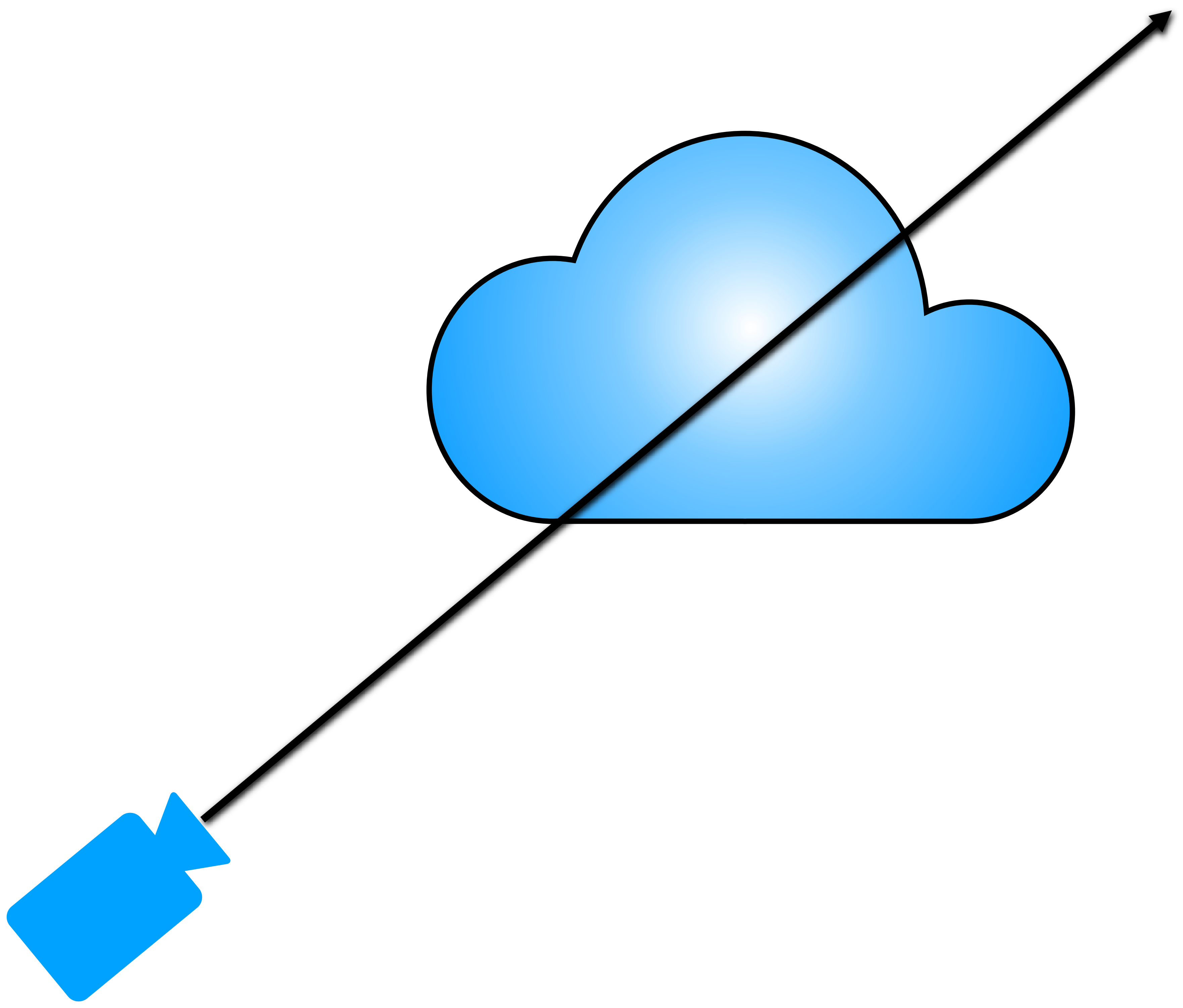
Expected value of color along ray

This means the expected color returned by the ray will be

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

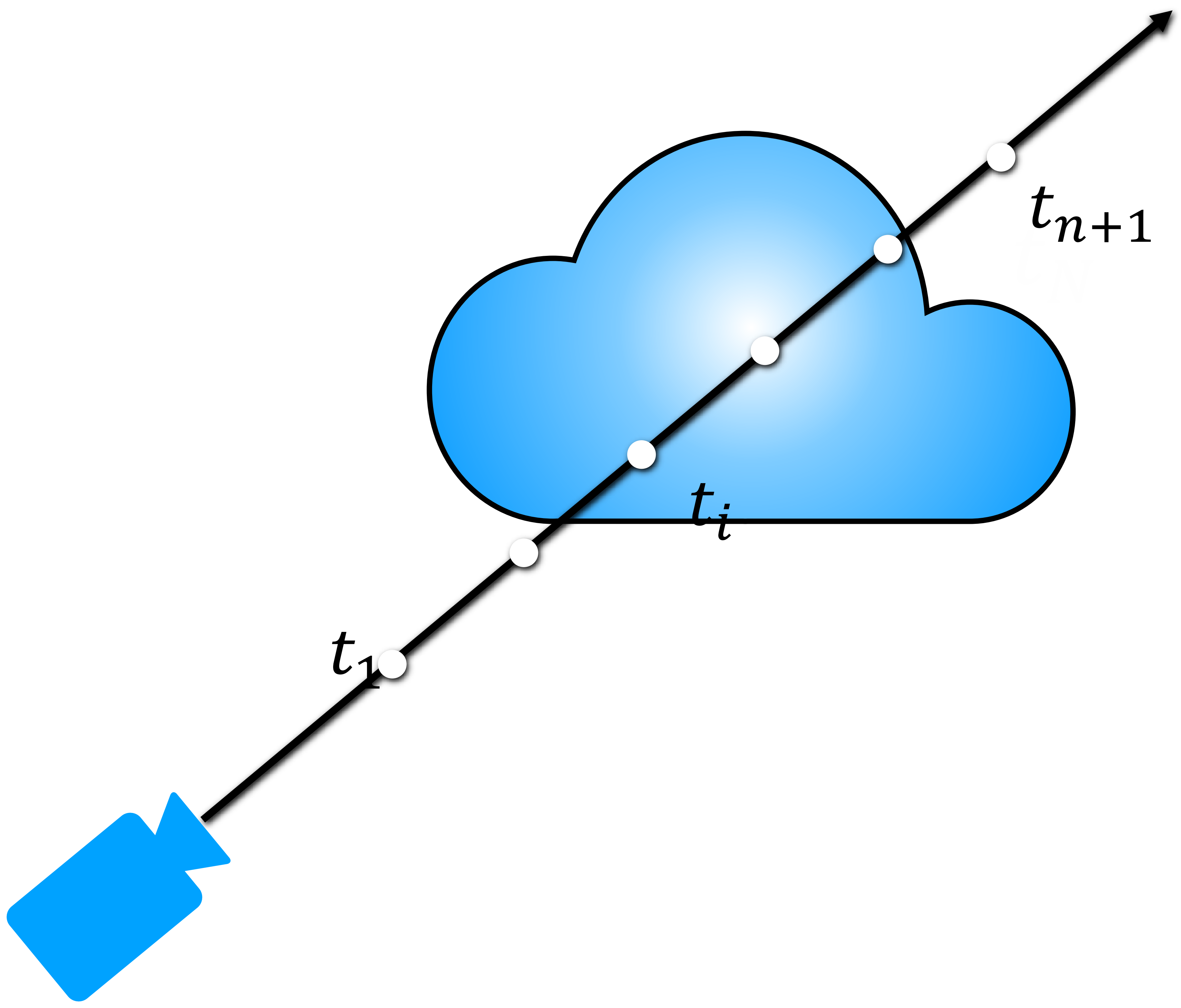
Note the nested integral!

Approximating the nested integral



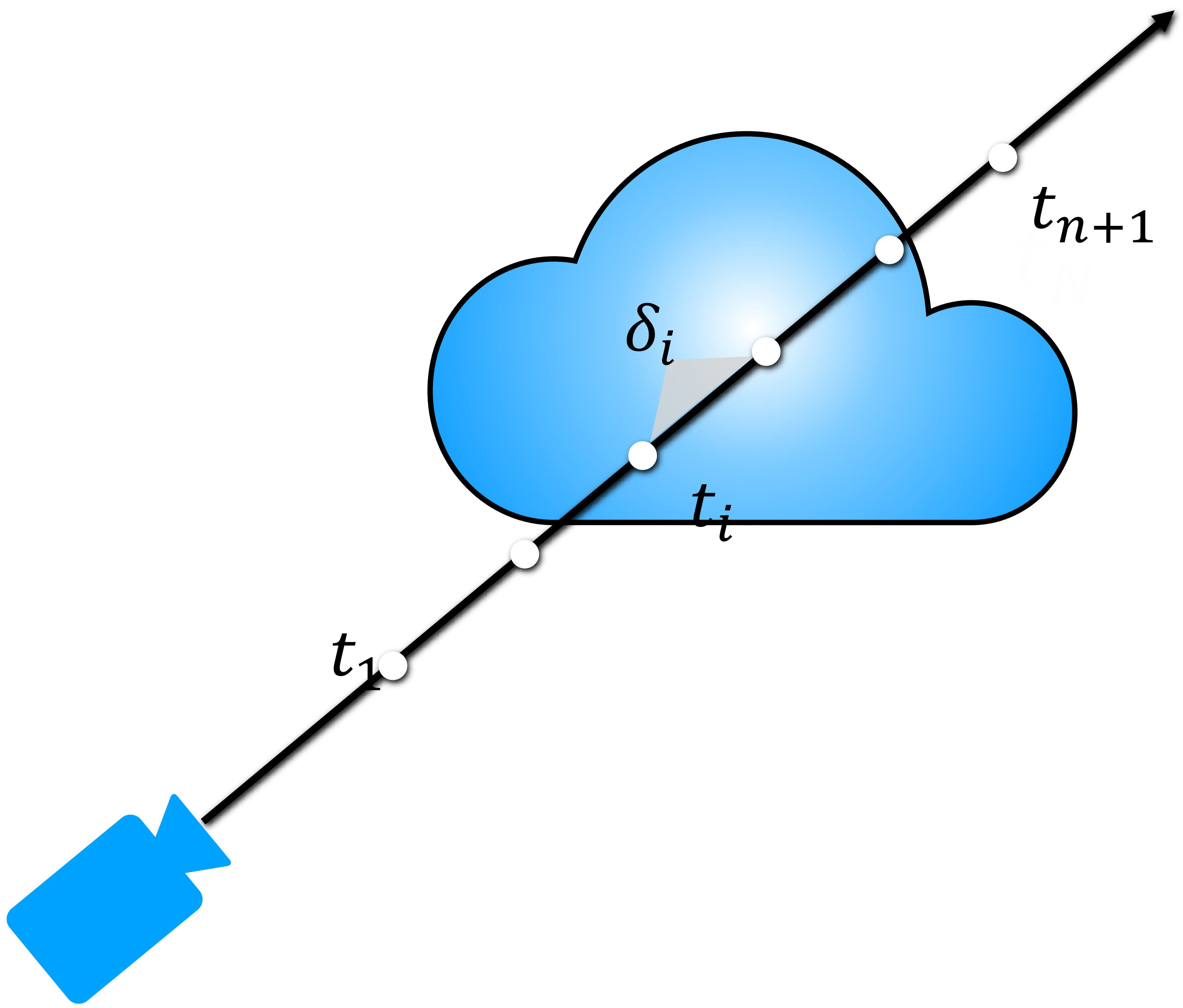
We use quadrature to approximate the nested integral,

Approximating the nested integral



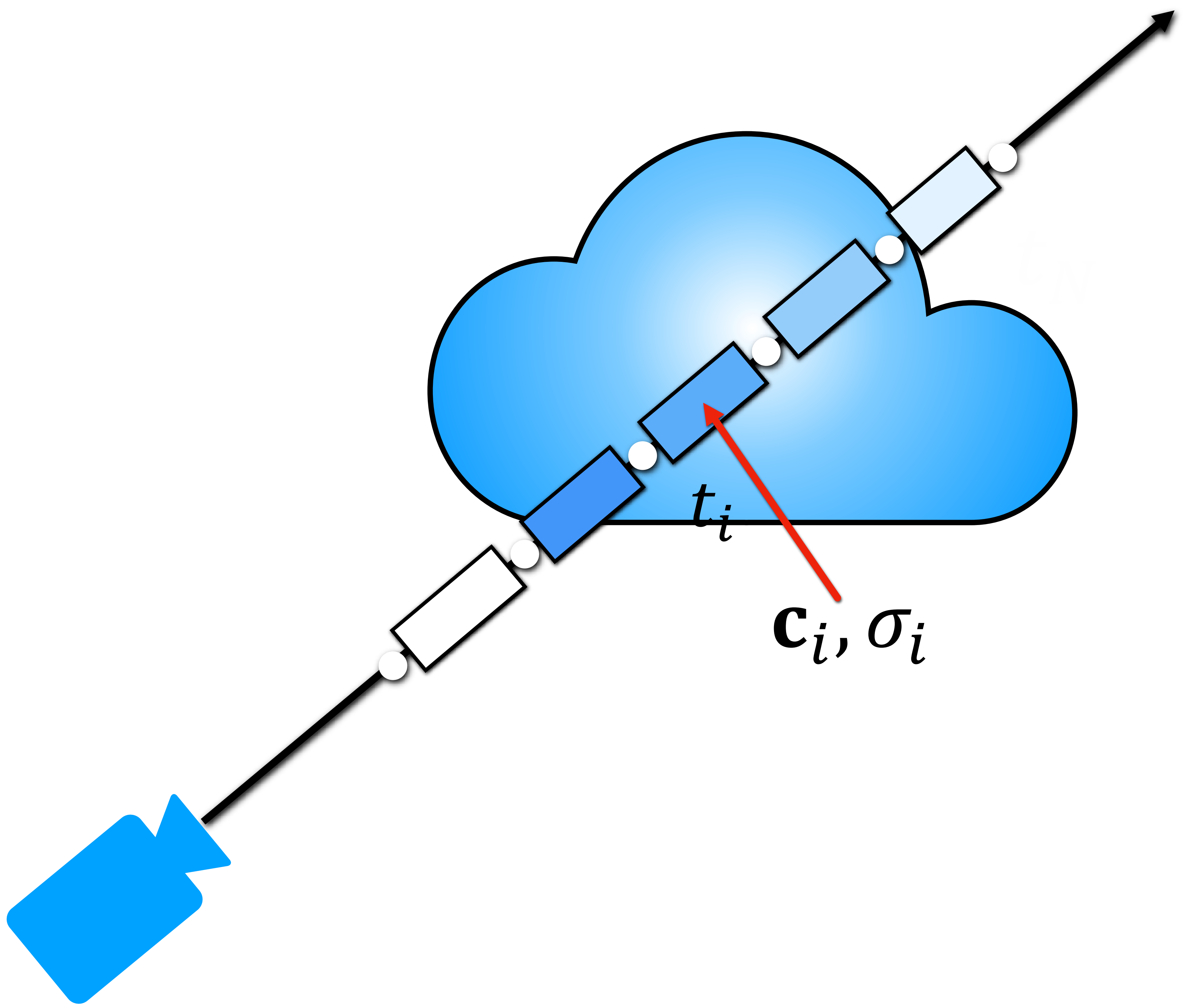
We use quadrature to approximate the nested integral,
splitting the ray up into n segments with endpoints $\{t_1, t_2, \dots, t_{n+1}\}$

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

Deriving quadrature estimate

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

This allows us to break the outer integral into a

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

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

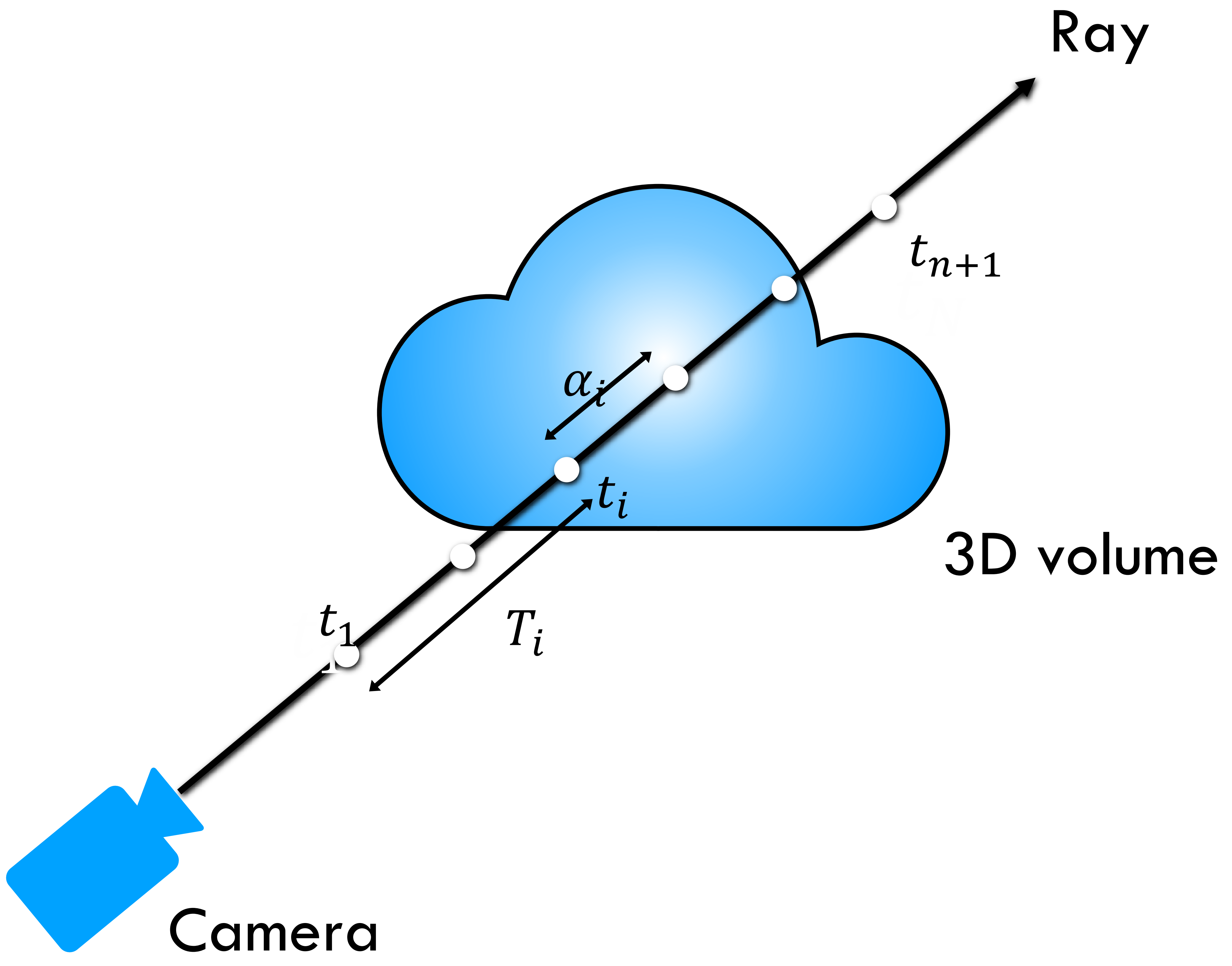
weights

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 $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

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

differentiable w.r.t. $\mathbf{c}$, σ

colors

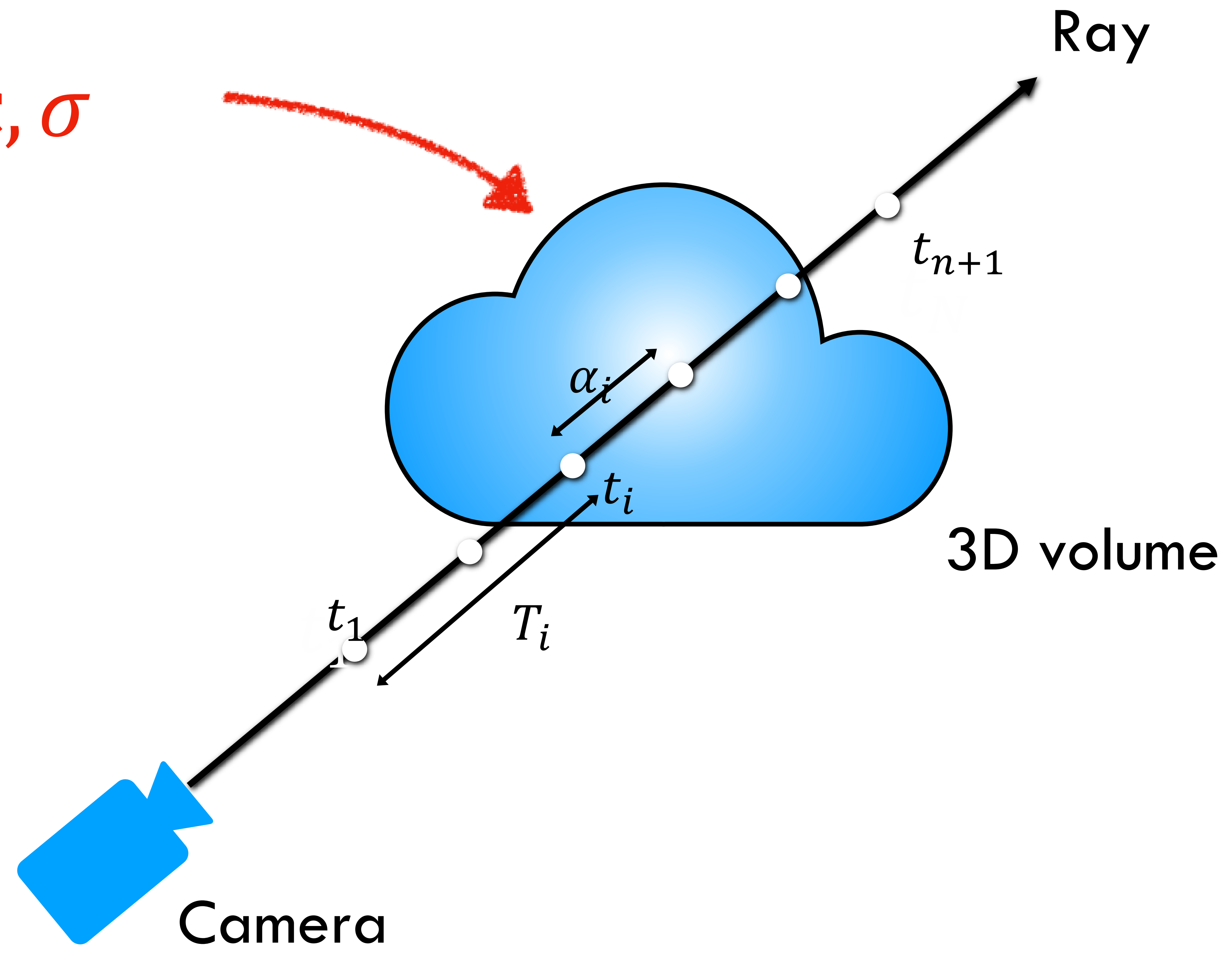
weights

How much light is blocked earlier along ray:

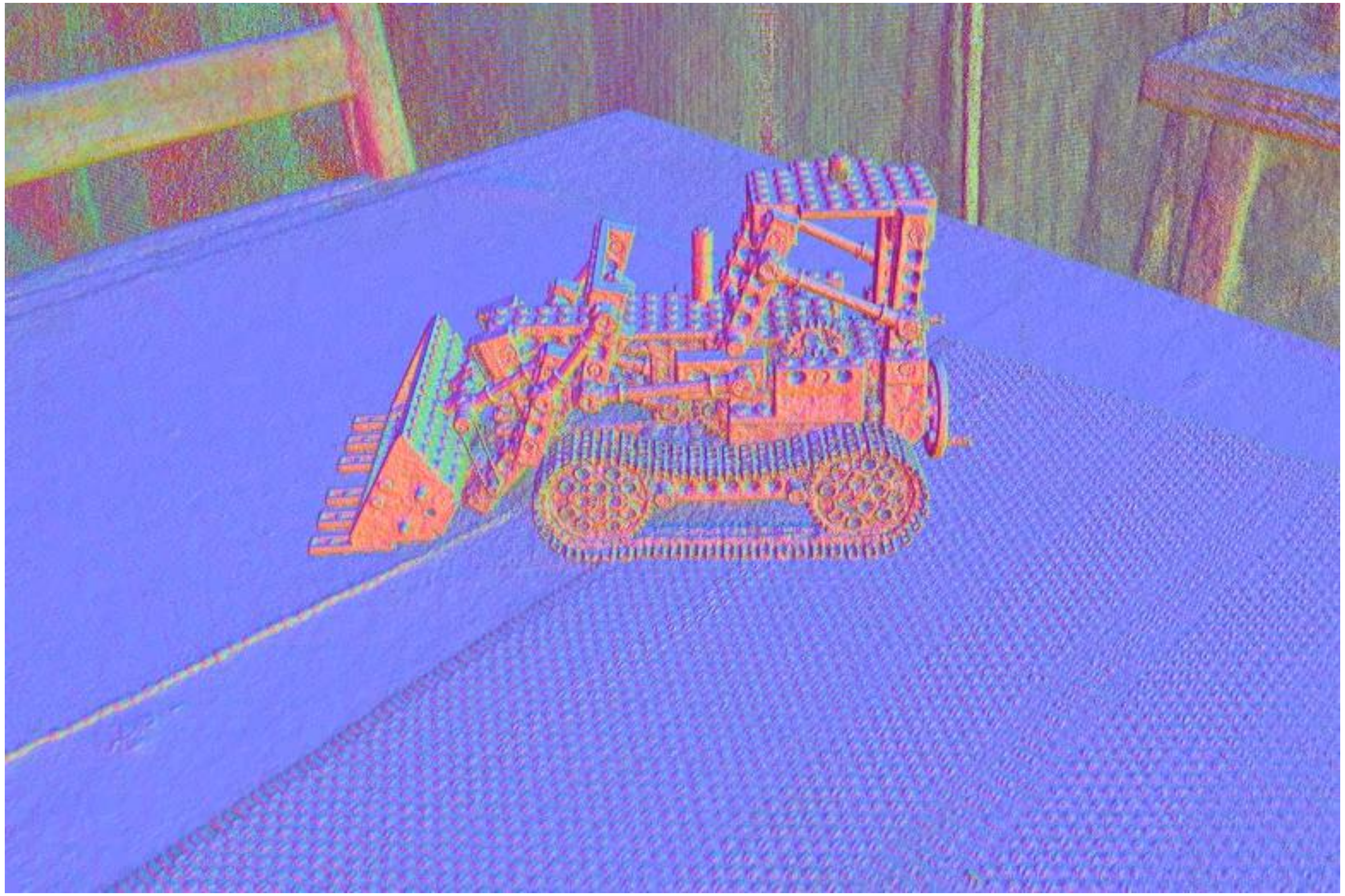
$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$

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

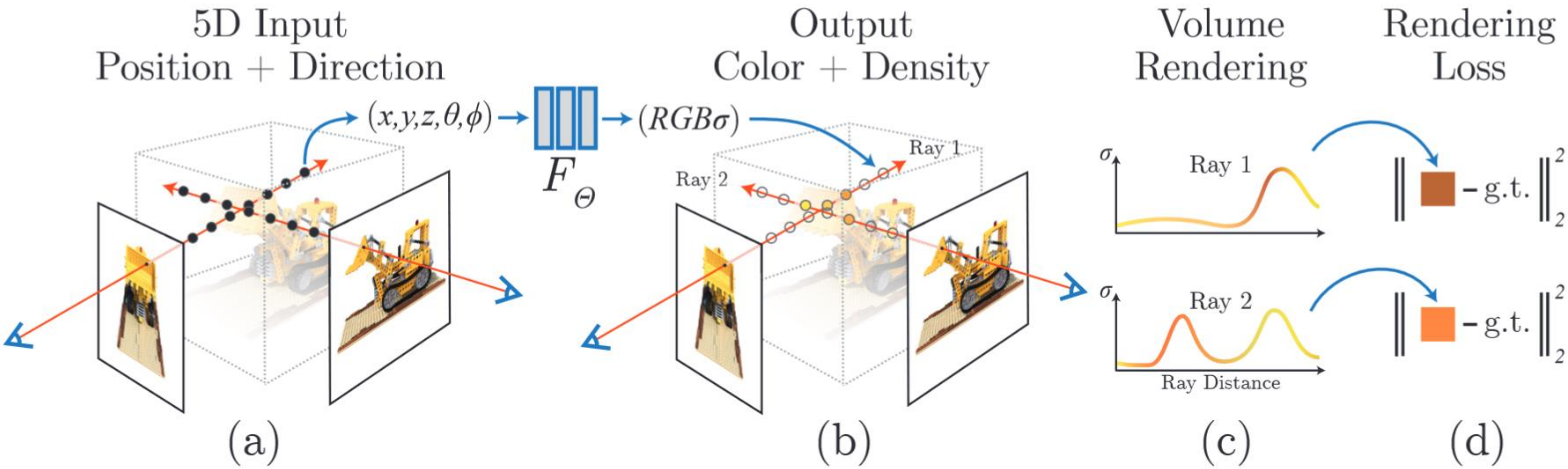
$$\alpha_i = 1 - \exp(-\sigma_i \delta_i)$$



Density as geometry

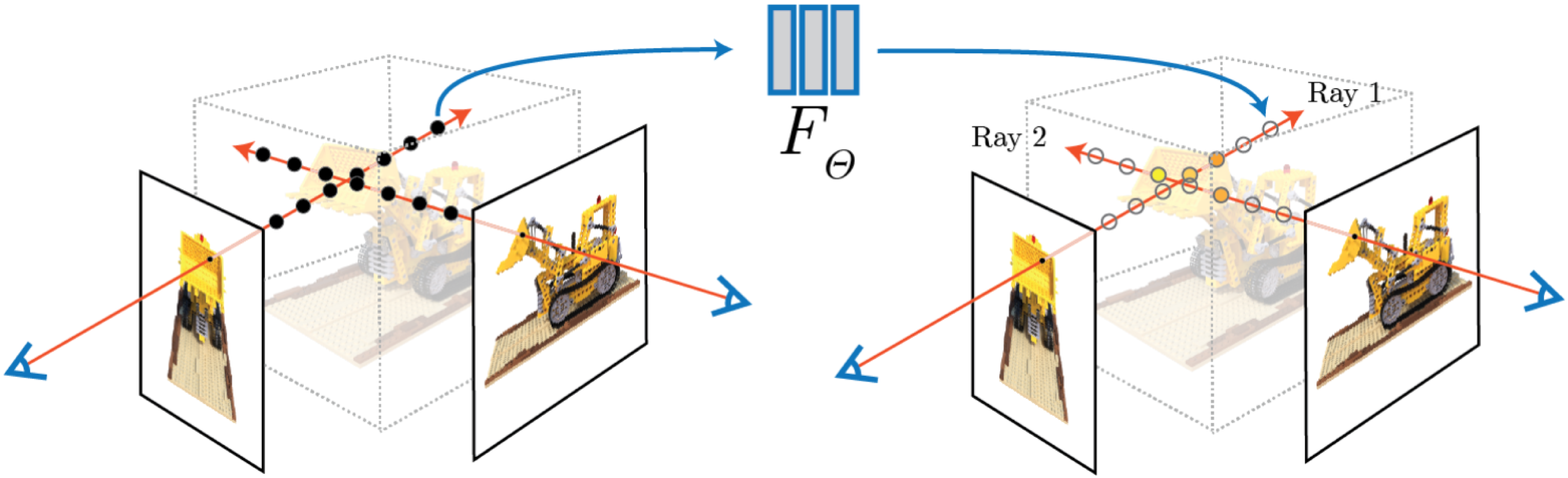


Normal vectors (from analytic gradient of density)


5D Input
Position + Direction
(x,y,z,θ,ϕ)
F_Θ
$(RGB\sigma)$
Output
Color + Density
Ray 1
Ray 2
Volume
Rendering
σ
Ray 1
σ
Ray 2
Ray Distance
Rendering
Loss
$\|\ \blacksquare - \text{g.t.}\ \|_2^2$
$\|\ \blacksquare - \text{g.t.}\ \|_2^2$
(a)
(b)
(c)
(d)

Par2 Neural Radiance Field Scene Representation

A simplified version of NeRF represents a continous scene as a function using the following MLP network, whose input is a 3D location $\mathbf{x}=(x, y, z)$ and whose output is an RGB color $\mathbf{c}=(r, g, b)$ and volume density σ at that 3D location.

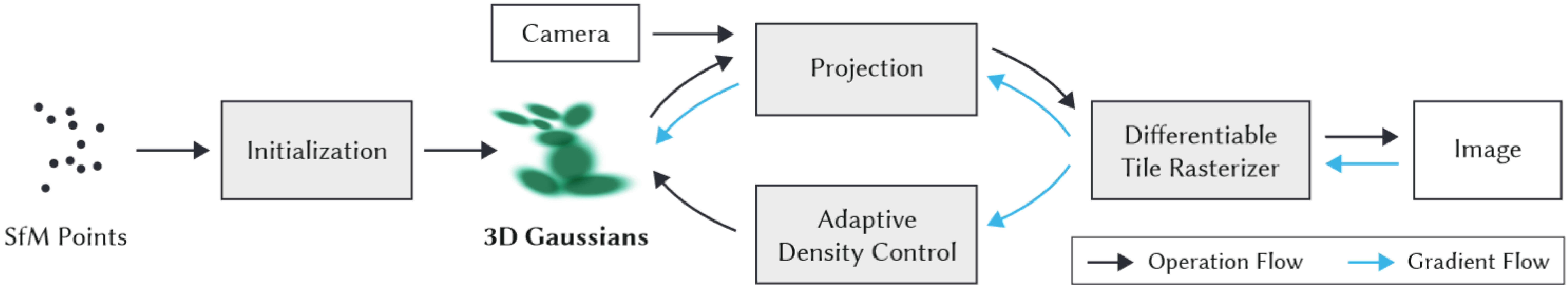


The following cell defines the network architecture of NeRF.

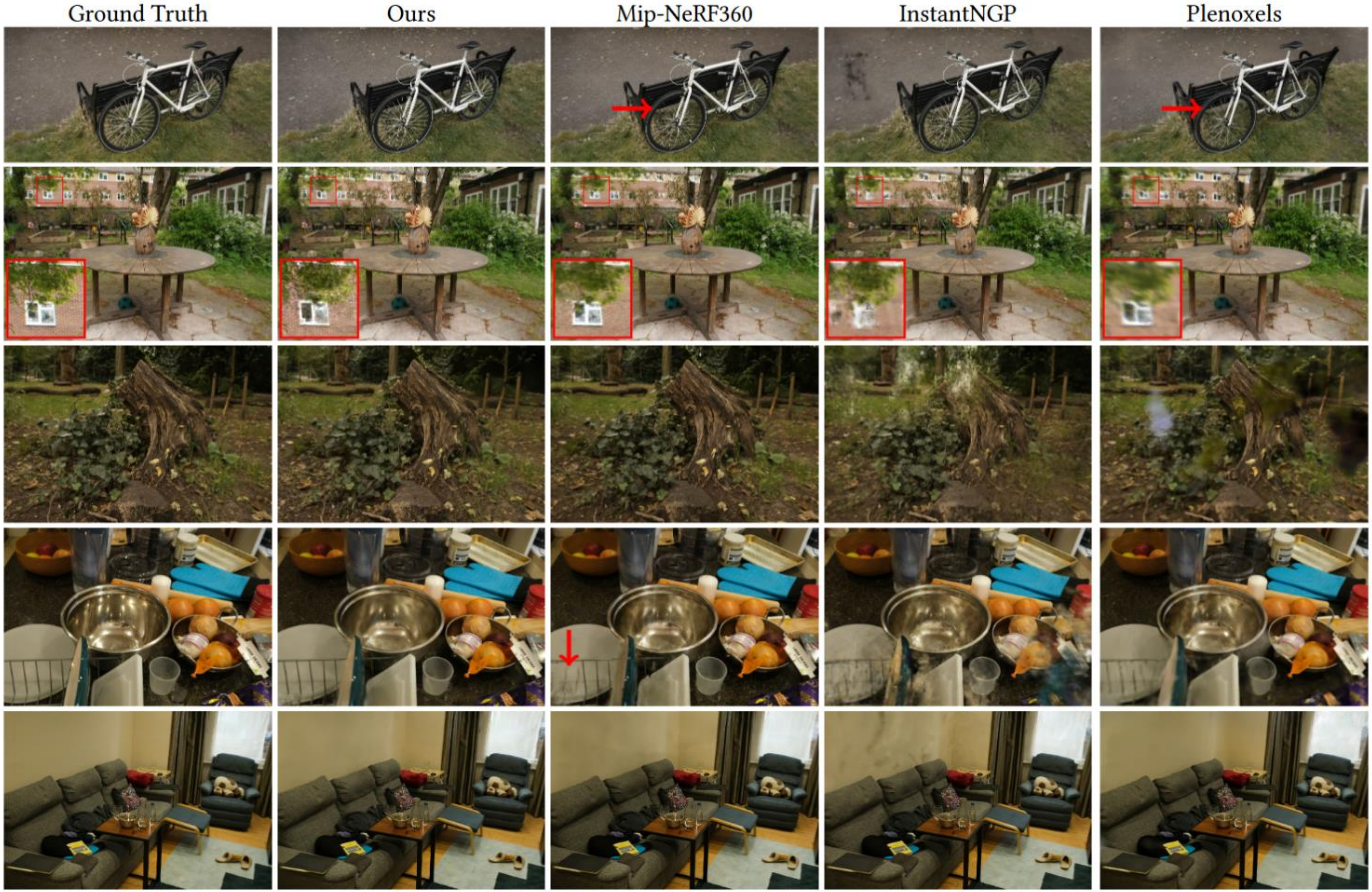
NERF model implementations are in part2.py

3D Gaussian Splatting for Real-Time Radiance Field Rendering

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



Camera
Projection
SfM Points
Initialization
3D Gaussians
Differentiable Tile Rasterizer
Image
Adaptive Density Control
Operation Flow
Gradient Flow


Ground Truth
Ours
Mip-NeRF360
InstantNGP
Plenoxels

Dataset	Mip-NeRF360						Tanks&Temples						Deep Blending					
Method|Metric	*SSIM*↑	*PSNR*↑	*LPIPS*↓	Train	FPS	Mem	*SSIM*↑	*PSNR*↑	*LPIPS*↓	Train	FPS	Mem	*SSIM*↑	*PSNR*↑	*LPIPS*↓	Train	FPS	Mem
Plenoxels	0.626	23.08	0.463	25m49s	6.79	2.1GB	0.719	21.08	0.379	25m5s	13.0	2.3GB	0.795	23.06	0.510	27m49s	11.2	2.7GB
INGP-Base	0.671	25.30	0.371	5m37s	11.7	13MB	0.723	21.72	0.330	5m26s	17.1	13MB	0.797	23.62	0.423	6m31s	3.26	13MB
INGP-Big	0.699	25.59	0.331	7m30s	9.43	48MB	0.745	21.92	0.305	6m59s	14.4	48MB	0.817	24.96	0.390	8m	2.79	48MB
M-NeRF360	0.792†	27.69†	0.237†	48h	0.06	8.6MB	0.759	22.22	0.257	48h	0.14	8.6MB	0.901	29.40	0.245	48h	0.09	8.6MB
Ours-7K	0.770	25.60	0.279	6m25s	160	523MB	0.767	21.20	0.280	6m55s	197	270MB	0.875	27.78	0.317	4m35s	172	386MB
Ours-30K	0.815	27.21	0.214	41m33s	134	734MB	0.841	23.14	0.183	26m54s	154	411MB	0.903	29.41	0.243	36m2s	137	676MB

https://youtu.be/T_kXY43VZnk?si=Ro2JF-gCz08W8vQH

Reminder:
Quiz in class on Wednesday