Neural Radiance Fields pt 2

Video from the original ECCV'20 paper

CS180/280A: Intro to Computer Vision and Computational Photography

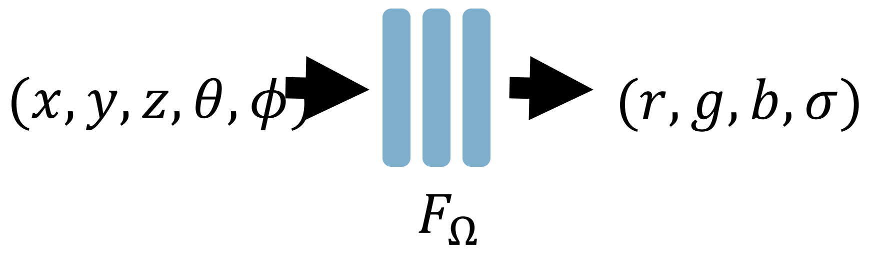
Angjoo Kanazawa and Alexei Efros

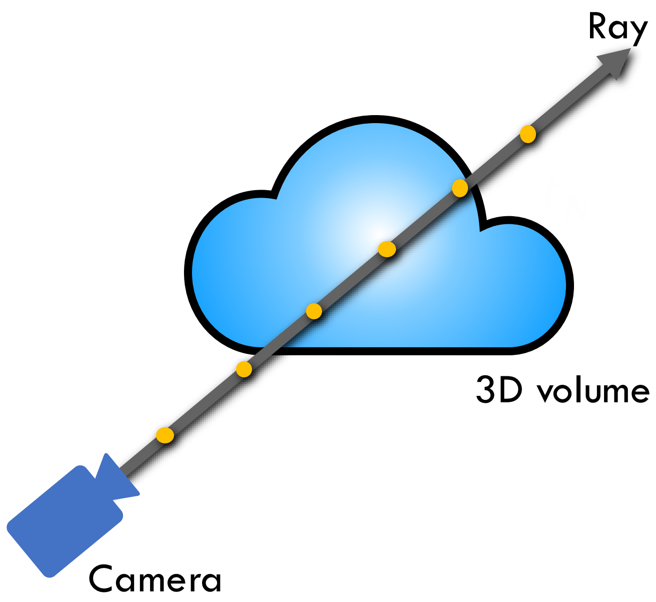
UC Berkeley Fall 2025

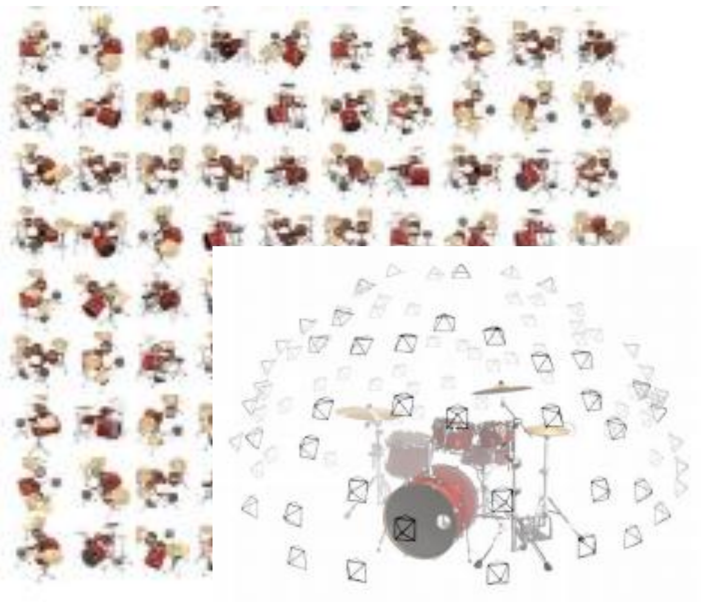
Lots of content from Noah Snavely and [ECCV 2022 Tutorial on Neural Volumetric Rendering for Computer Vision]

Recap: 3 Key Components

Objective: Synthesize all training views







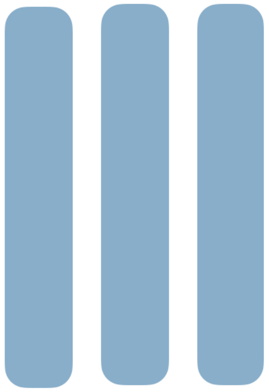
Neural Volumetric 3D Scene Representation

Differentiable Volumetric Rendering Function

Optimization via Analysis-by-Synthesis

"Neural Radiance Fields"

$$(\underbrace{x, y, z}_{\text{Spatial location}}, \underbrace{\theta, \phi}_{\text{Viewing direction}}) \rightarrow F_{\Omega} \rightarrow (\underbrace{r, g, b}_{\text{Output color}}, \underbrace{\sigma}_{\text{Output density}})$$

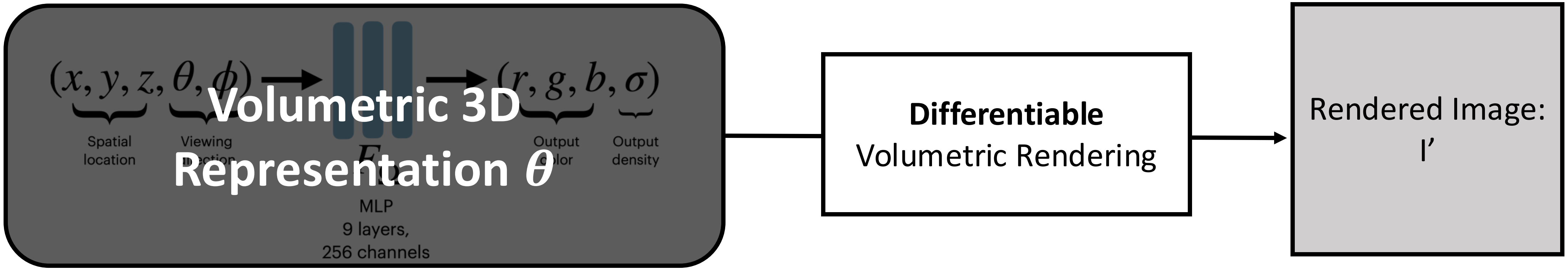


F_{Ω}

MLP
9 layers,
256 channels

“Neural Radiance Fields”

How an image is made (“Inference”)

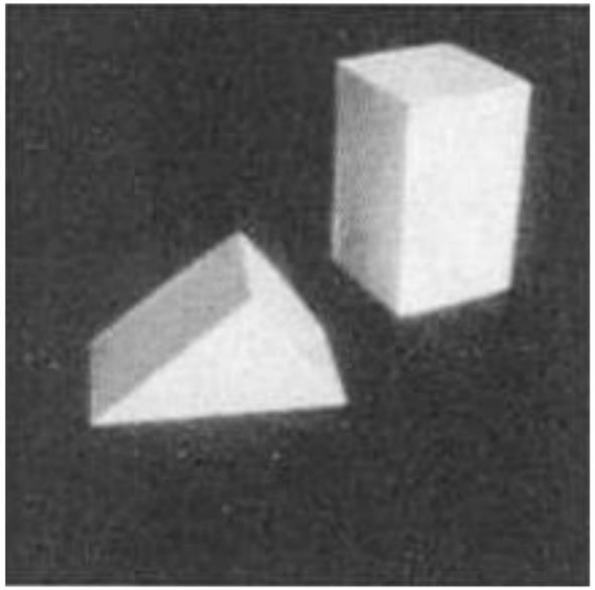


“Training” Objective (aka Analysis-by-Synthesis):

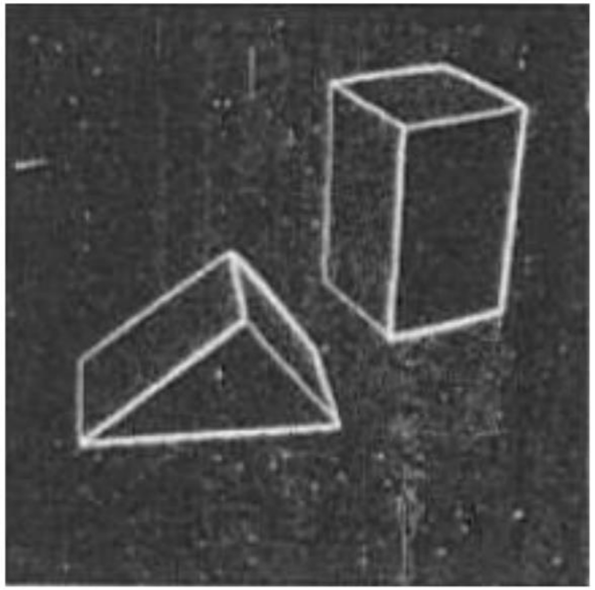
$$\min_{\theta} \| \text{Rendered Image: } I' - \text{Observed Image: } I \|_2$$

Analysis-by-Synthesis

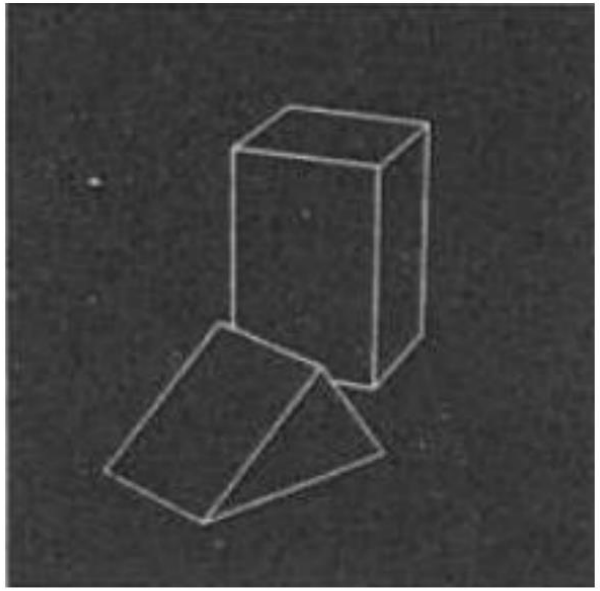
Larry Roberts
"Father of Computer Vision"



Input image



2x2 gradient operator

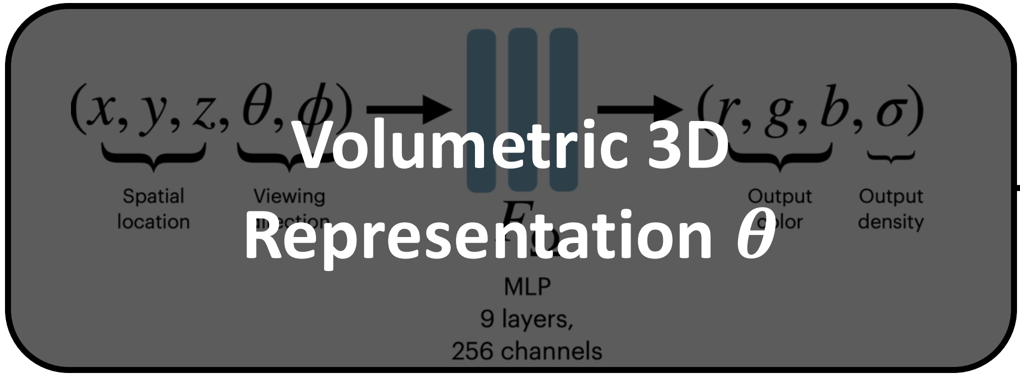


computed 3D model
rendered from new viewpoint

- History goes way back to the **first** Computer Vision paper!
  Roberts: Machine Perception of Three-Dimensional Solids, MIT, 1963

"Neural Radiance Fields"

Forward Function: How an image is made (Inference)



→ **Differentiable** Volumetric Rendering → Rendered Image: I'

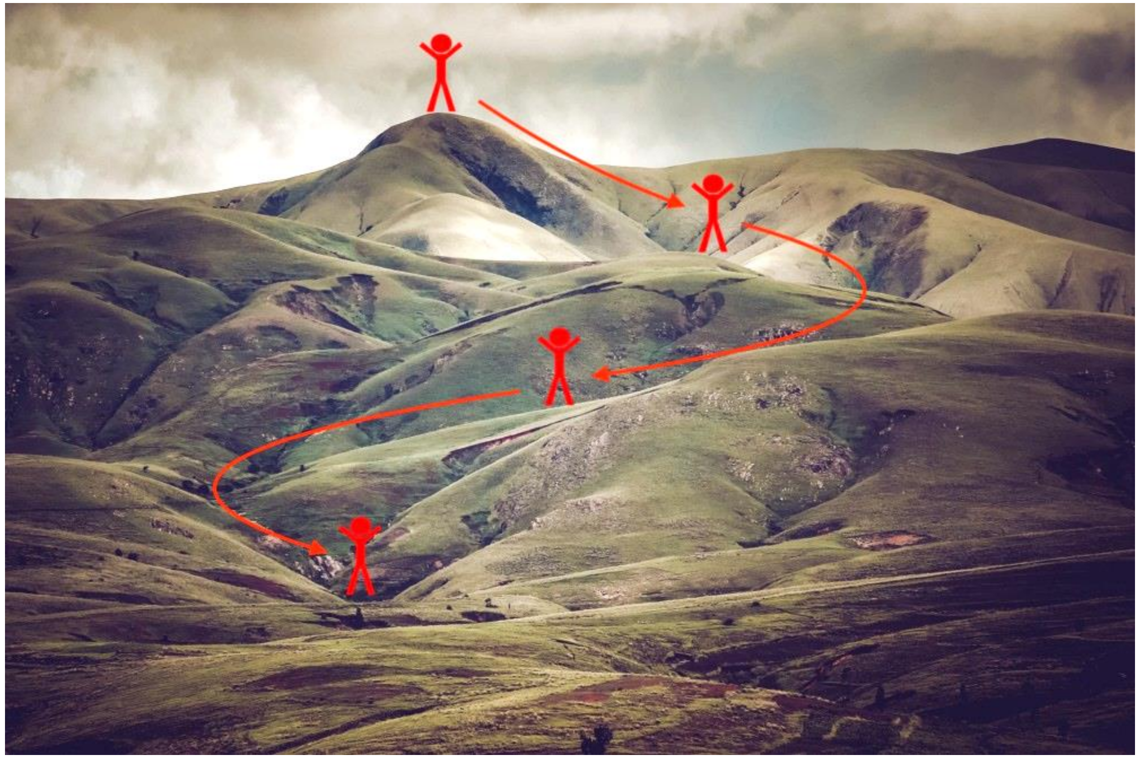
"Training" Objective (aka Analysis-by-Synthesis):

$$\min_{\theta} \| \text{Rendered Image: } I' - \text{Observed Image: } I \|_2$$

Differentiable Rendering

- How to change θ (network parameter) so that we get the final image?

- Gradient Descent "Hiking"



Same idea here, "hiking" now means you're going to change the network parameter little by little.

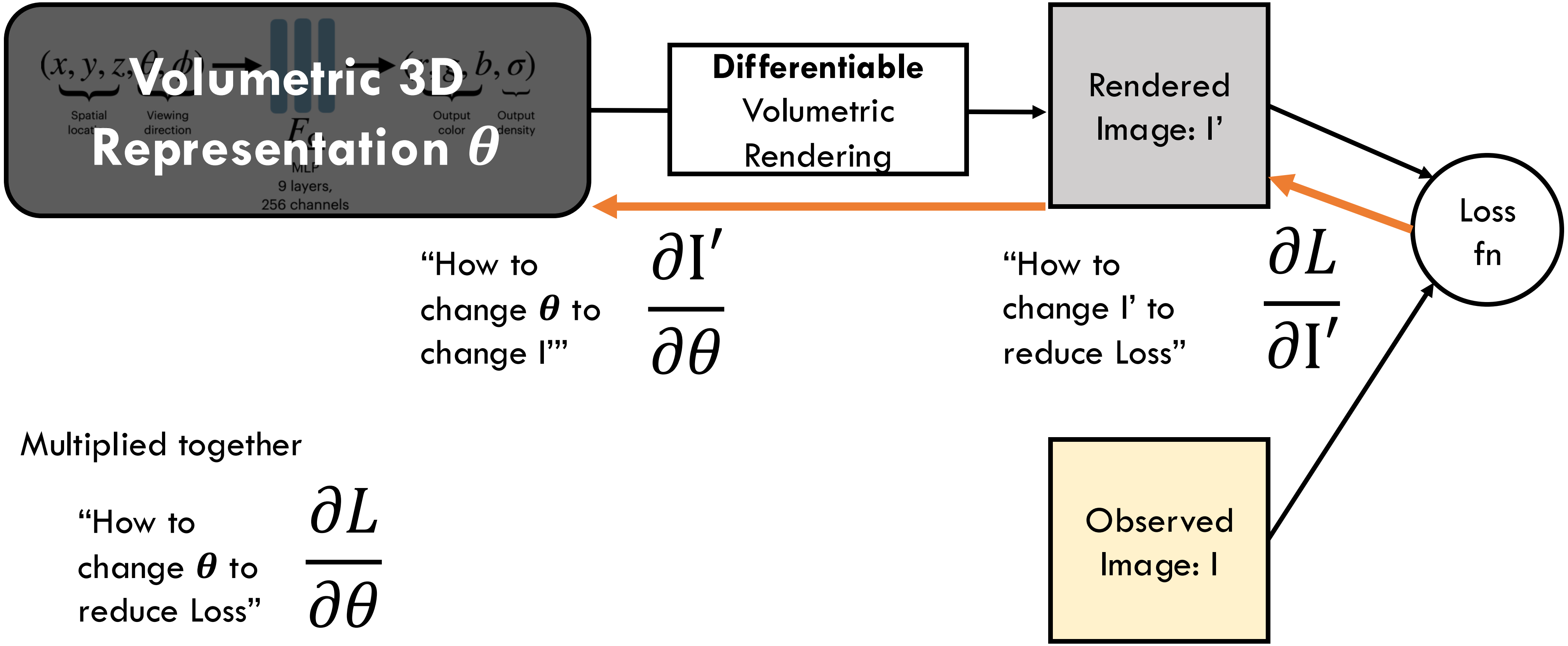
The "Mountain" or the "Loss" comes from the reconstruction loss.

$$L = ||I' - I||$$
$$I' = f(x\,;\theta)$$

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \mathrm{I}'}\frac{\partial \mathrm{I}'}{\partial \theta}$$

Chain rule, aka Back propagation

“Neural Radiance Fields”

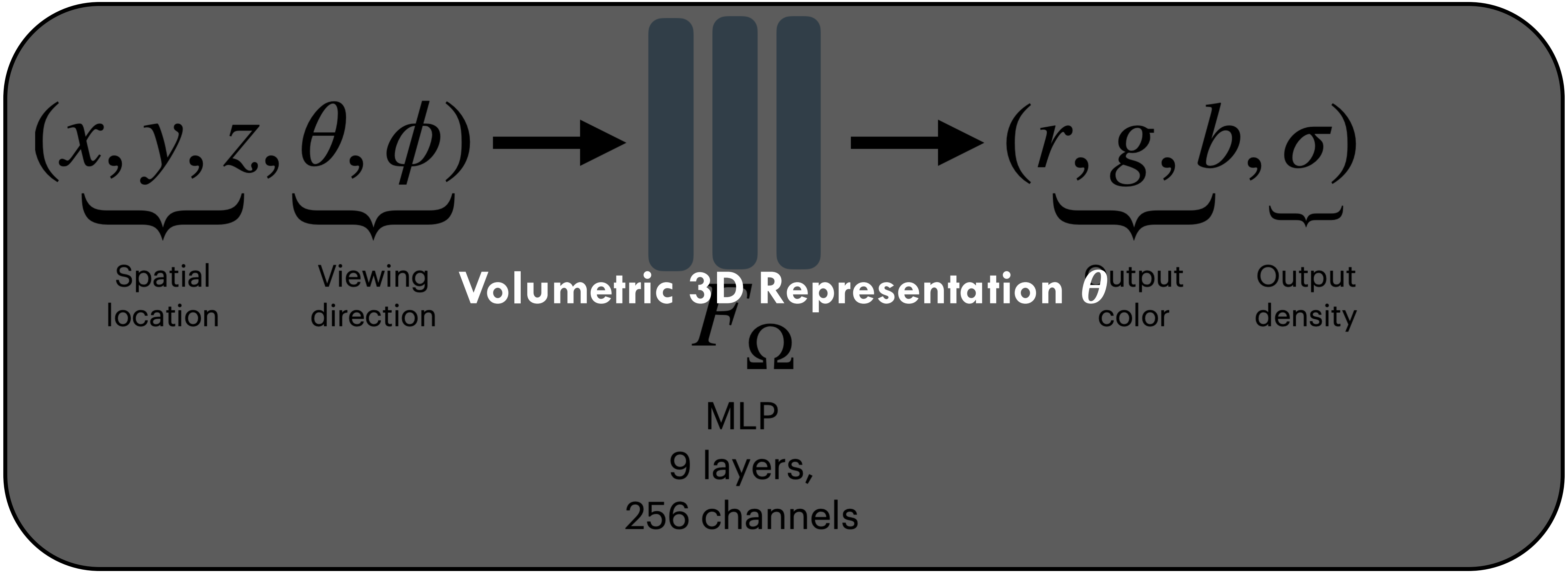


Multiplied together

“How to change $\boldsymbol{\theta}$ to reduce Loss”

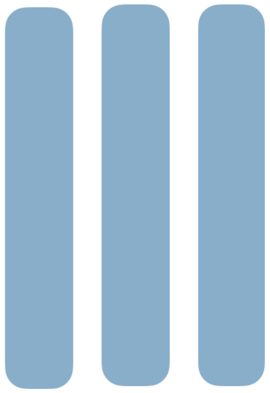
$$\frac{\partial L}{\partial \theta}$$

“Neural Radiance Fields”



"Neural Radiance Fields"

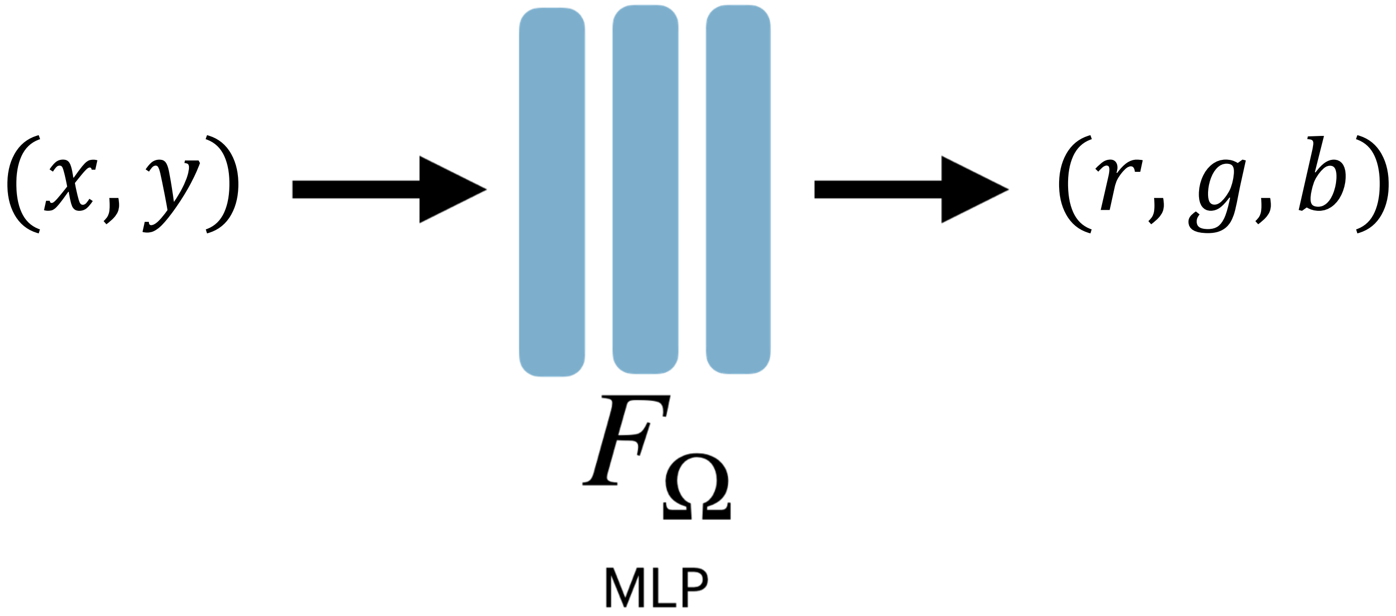
$$(\underbrace{x, y, z}_{\text{Spatial location}}, \underbrace{\theta, \phi}_{\text{Viewing direction}}) \longrightarrow F_{\Omega} \longrightarrow (\underbrace{r, g, b}_{\text{Output color}}, \underbrace{\sigma}_{\text{Output density}})$$



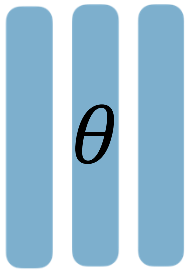
F_{Ω}

MLP
9 layers,
256 channels

Let’s simplify, do this in 2D:



Let's simplify, do this in 2D:

$(x, y) \rightarrow$  $\rightarrow (r, g, b)$

θ

F_Ω

MLP

Retrieve color from this network for every pixel $\longrightarrow$ Rendered Image: I'

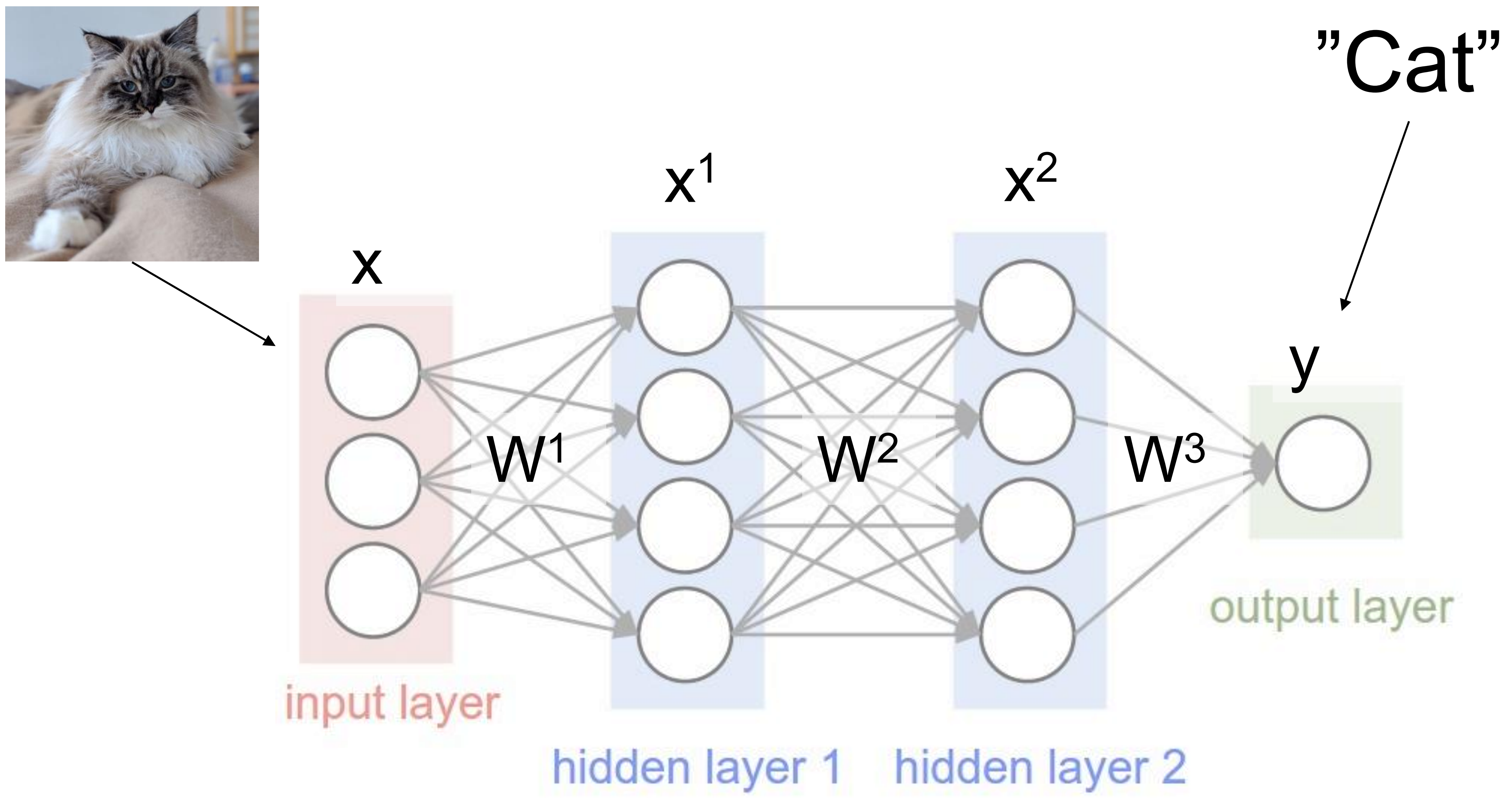
Optimize with "Training" Objective (aka Analysis-by-Synthesis):

$$\frac{\partial L}{\partial \theta} = \frac{\partial (rgb - rgb')}{\partial \theta}$$

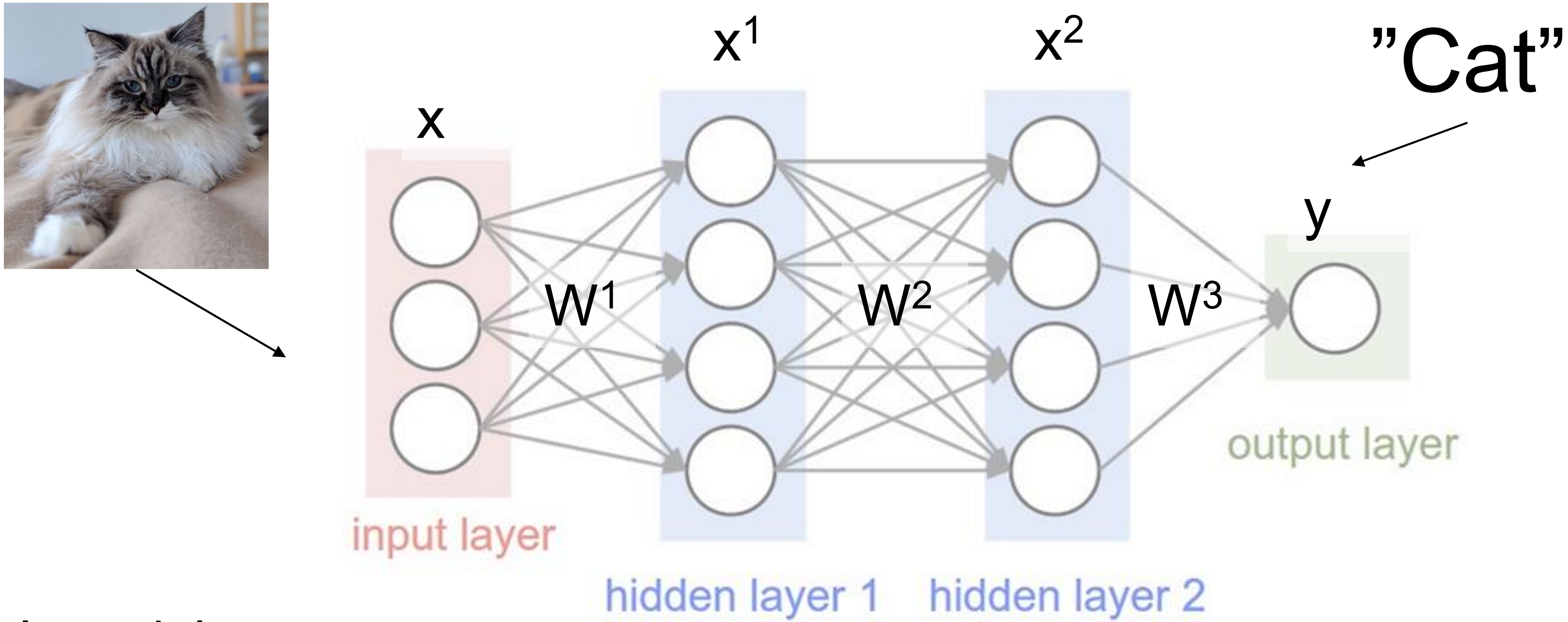
$$\min_{\theta} \left\| \boxed{\text{Rendered Image: I'}} - \boxed{\text{Observed Image: I}} \right\|_2$$

Straight forward to implement with Pytorch

ML Recap: Multi-layer perceptrons / Fully-Connected Layer



Multi-layer perceptrons / Fully-Connected Layer



In each layer:

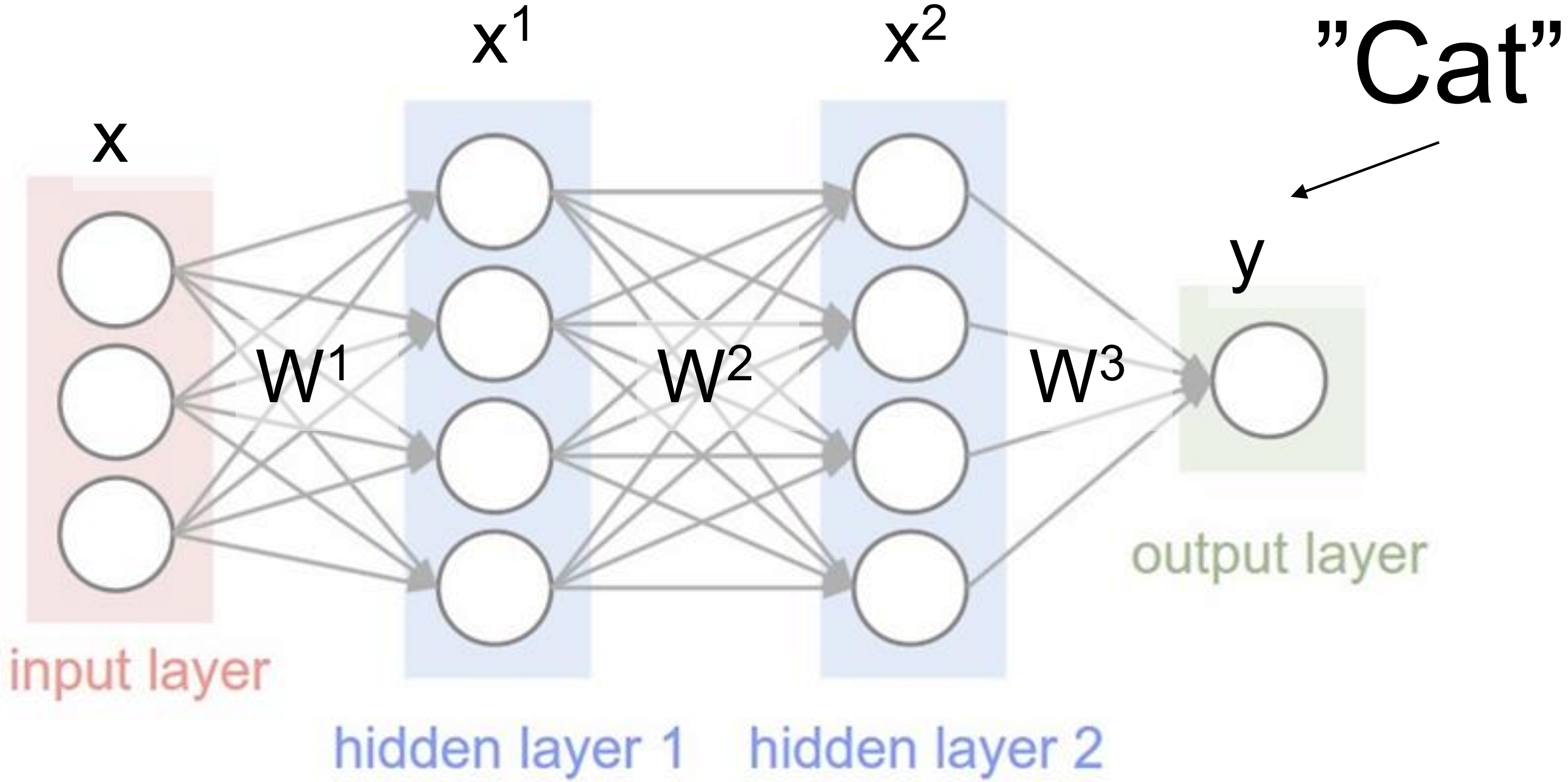
1. Linear Transform $z = W^l x^{l-1} + b$
2. Apply Non-Linearity $x^l = f(z)$

Usually

$$f = RELU(z) = \max(0, z)$$

what happens if f is identity?

Multi-layer perceptrons / Fully-Connected Layer



In each layer:

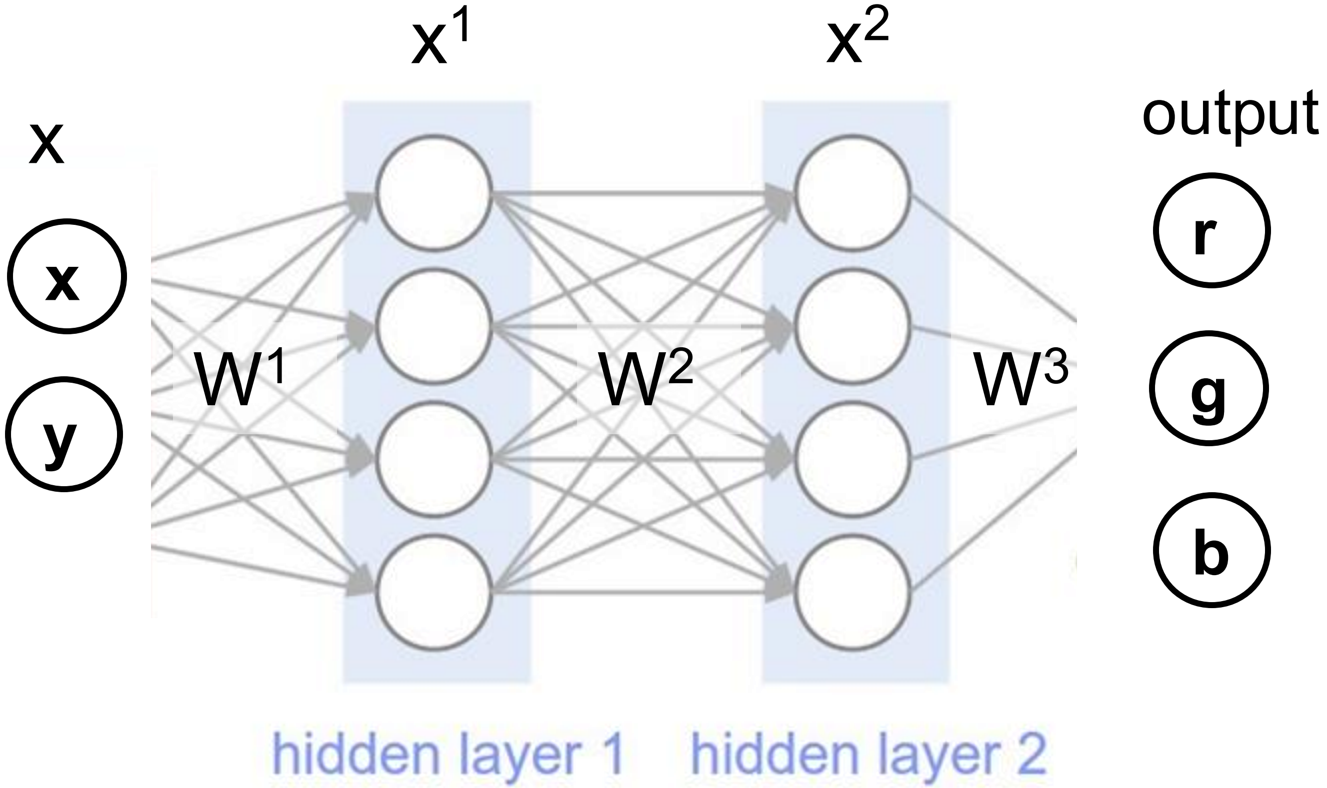
1. Linear Transform $z = W^l x^{l-1} + b$
2. Apply Non-Linearity $x^l = f(z)$

Usually

$$f = RELU(z)$$
$$= \max(0, z)$$

What are the learnable parameters?

In our 2D case:



In each layer:

1. Linear Transform $z = W^l x^{l-1} + b$
2. Apply Non-Linearity $x^l = f(z)$

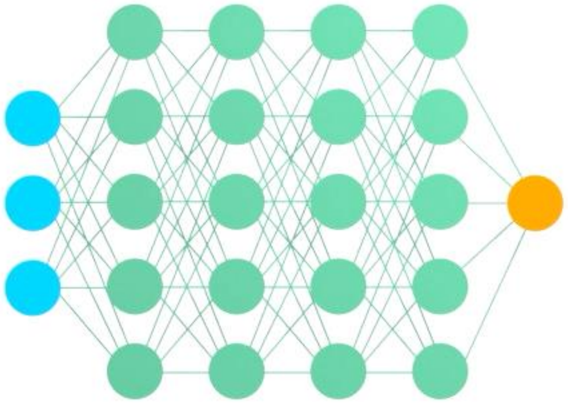
Usually

$$f = RELU(z) = \max(0, z)$$

What are the learnable parameters?

Coordinate Based Neural Network

Input
Coordinate



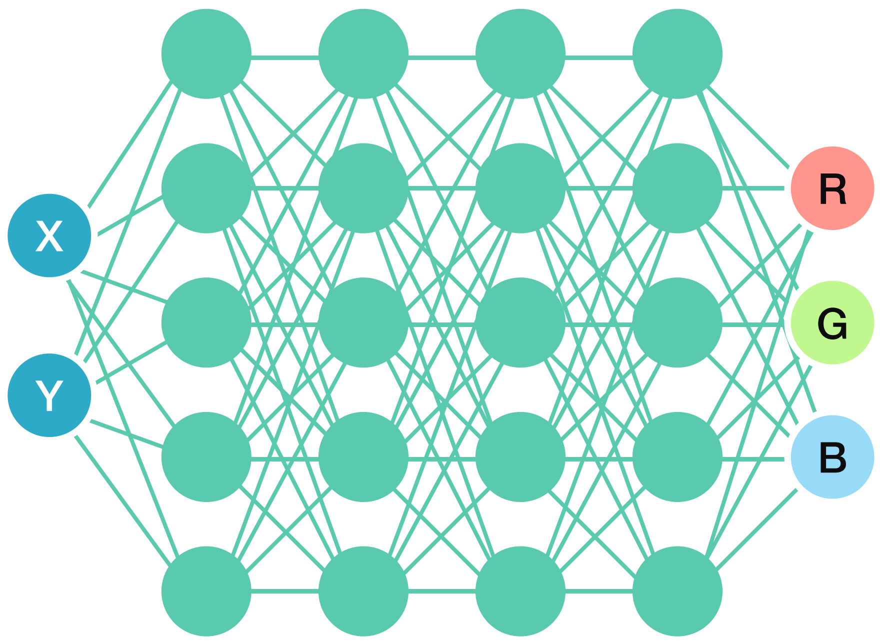
Value at
Coordinate

Multi Layer Perceptron

MLP

Slide credit: Matt Tancik

Image Representation



Slide credit: Matt Tancik

Challenge:

How to get MLPs to represent higher frequency functions?

Rahaman et al. 2019, Basri et al. 2020

Slide credit: Matt Tancik

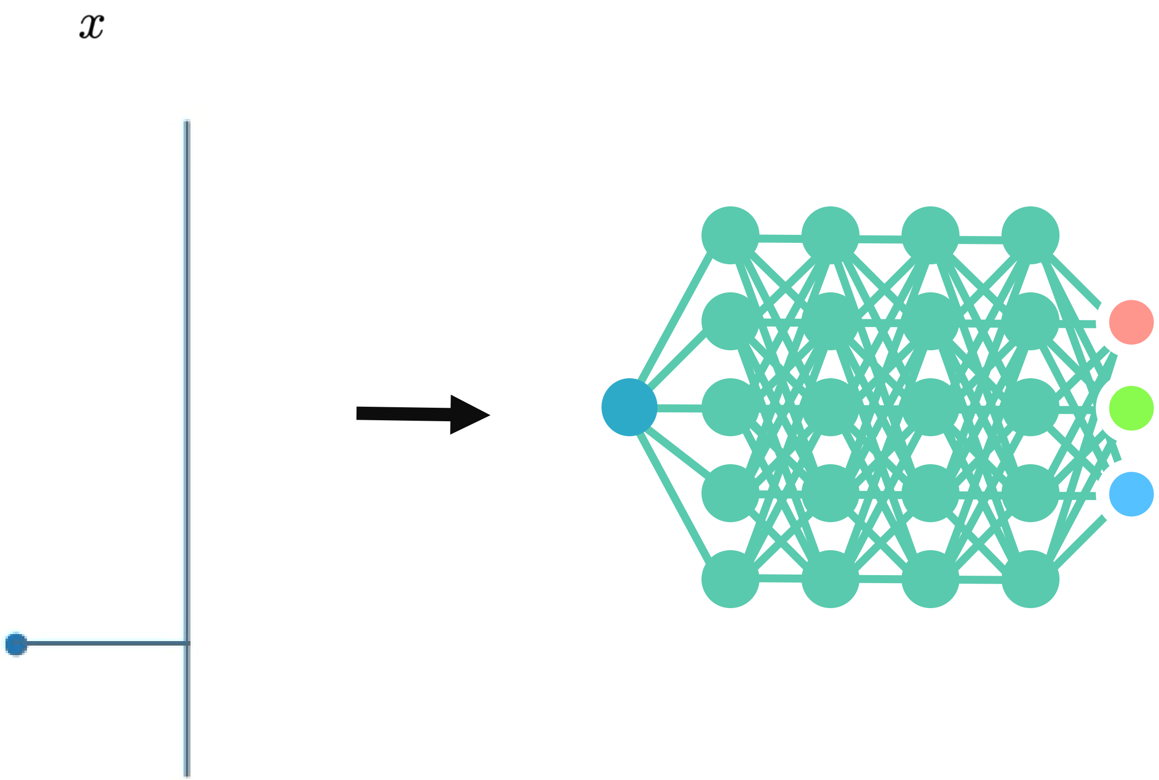
what happens if you naively optimize this network

MLP output

Supervision image

Slide credit: Matt Tancik

Standard input

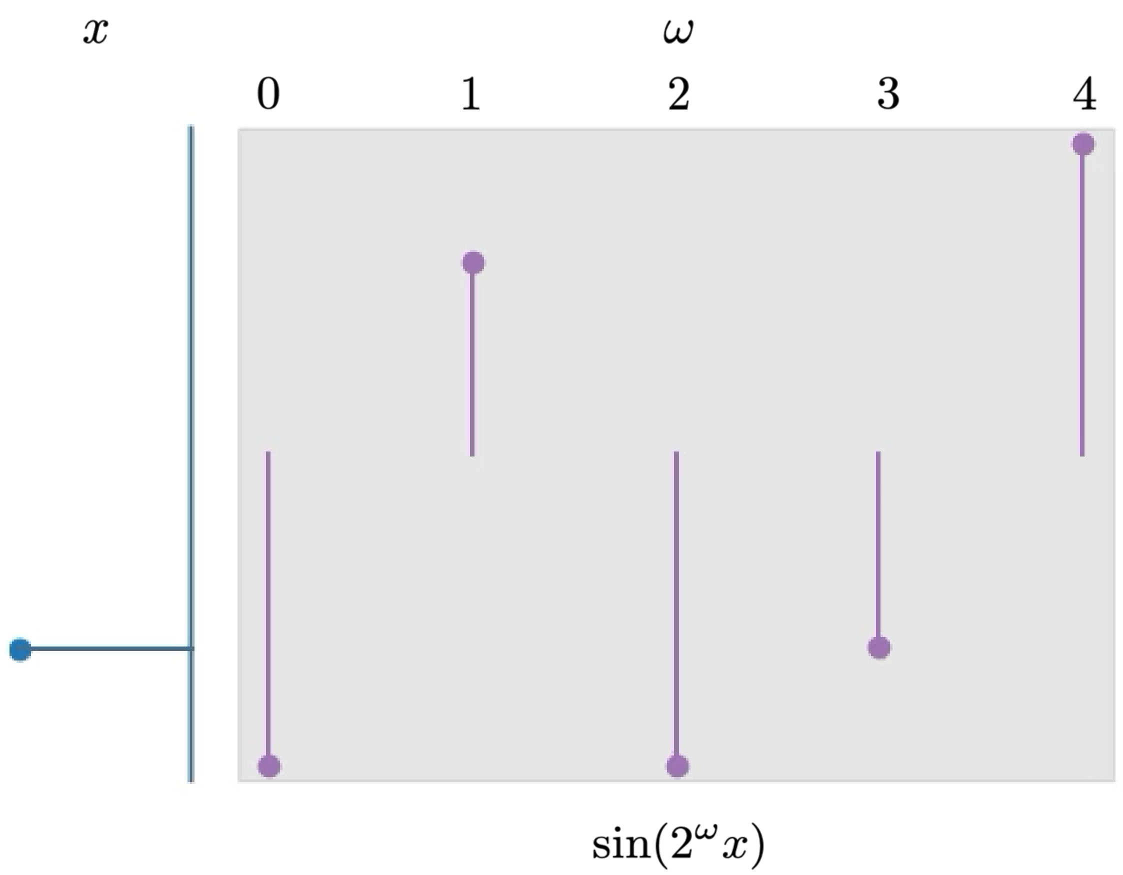


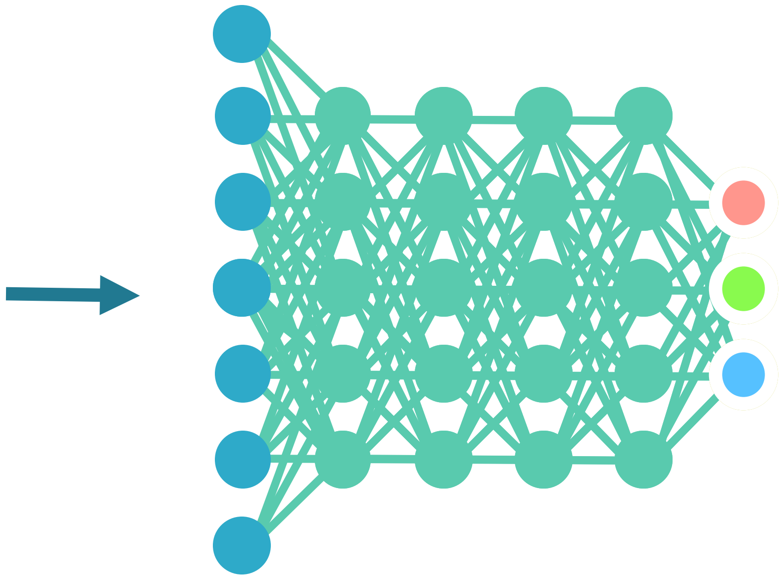
Slide credit: Matt Tancik

Positional Encoding

Standard input

Positionally Encoded input

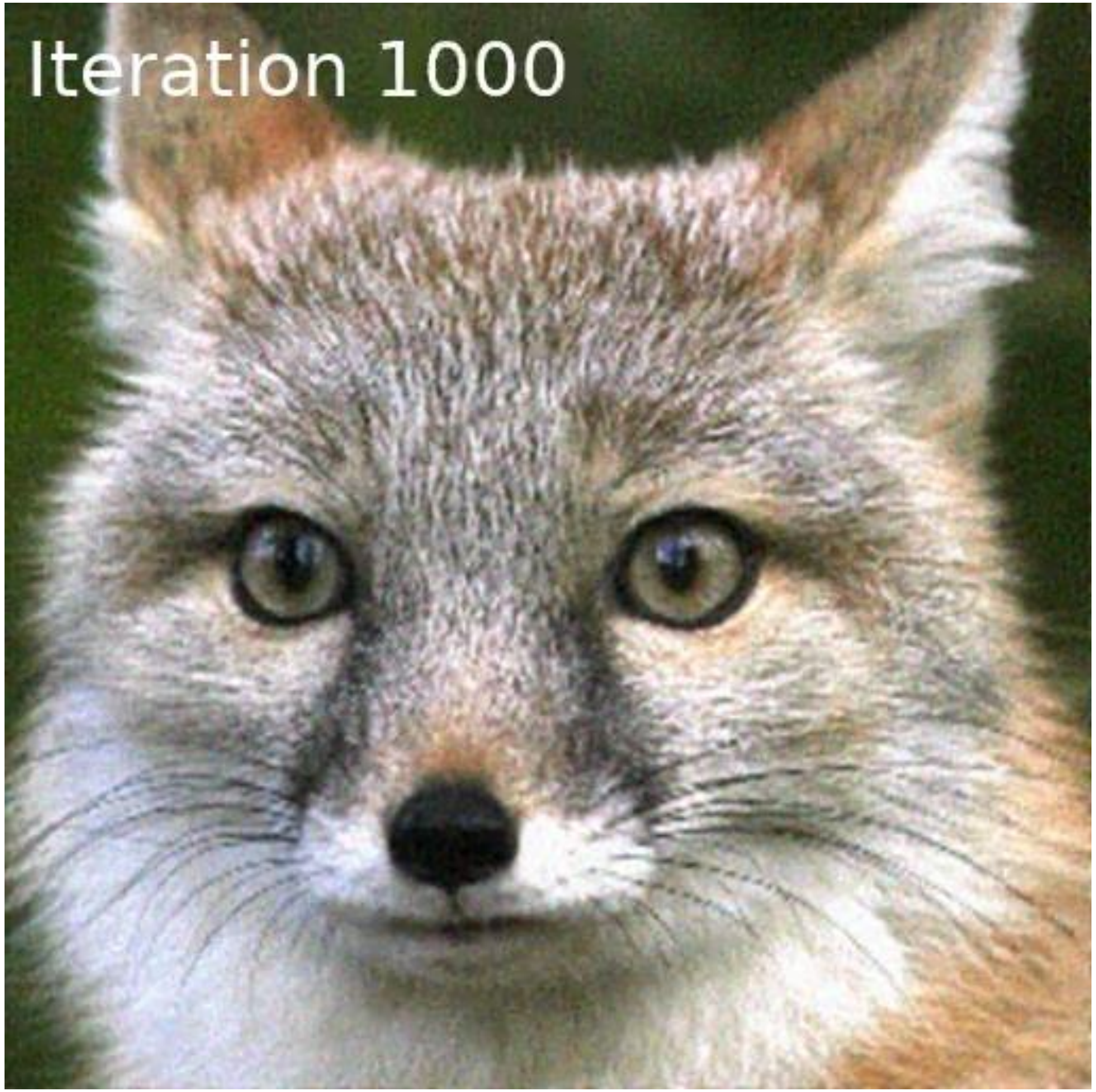




Fourier Features $\gamma(p) = \left(\sin(2^0 \pi p), \cos(2^0 \pi p), \cdots, \sin(2^{L-1} \pi p), \cos(2^{L-1} \pi p) \right)$

Slide credit: Matt Tancik

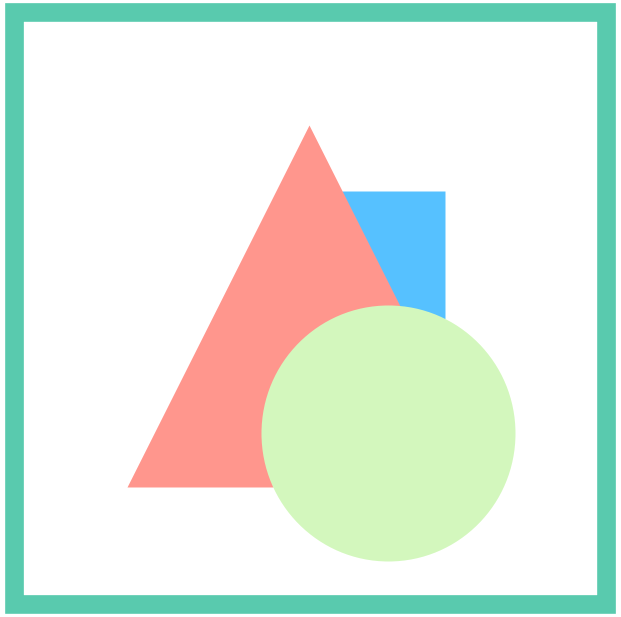
Standard MLP



MLP with Fourier features

Slide credit: Matt Tancik

Why does positional encoding help?

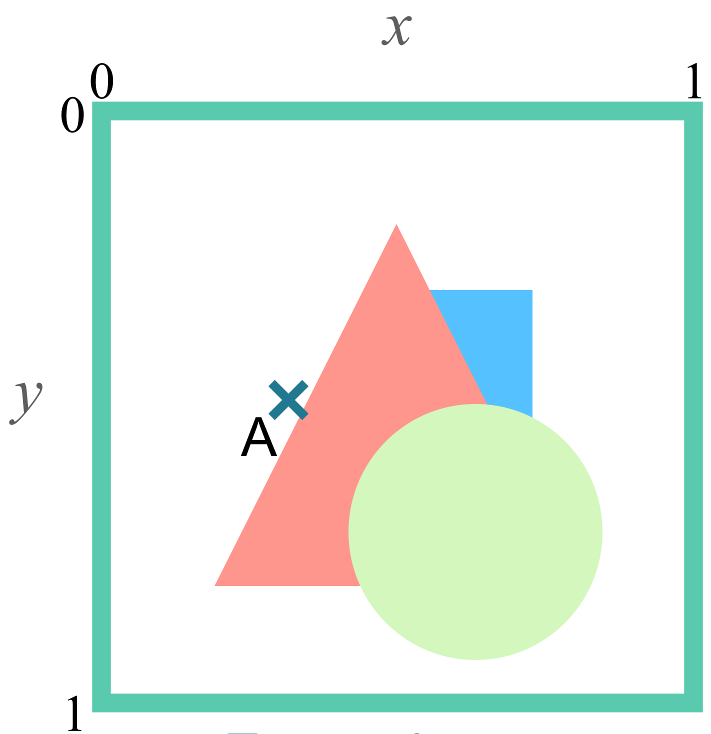


Target Image

Slide credit: Matt Tancik

Why does positional encoding help?

	Input		Target
	x	y	
A	.36	.5	□



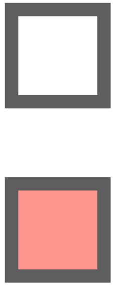
Target Image

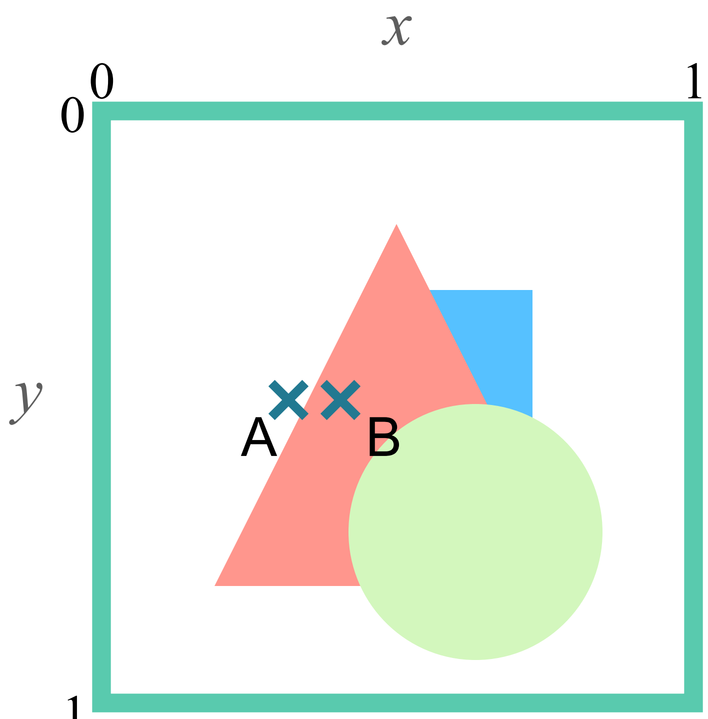
Slide credit: Matt Tancik

Why does positional encoding help?

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

Target

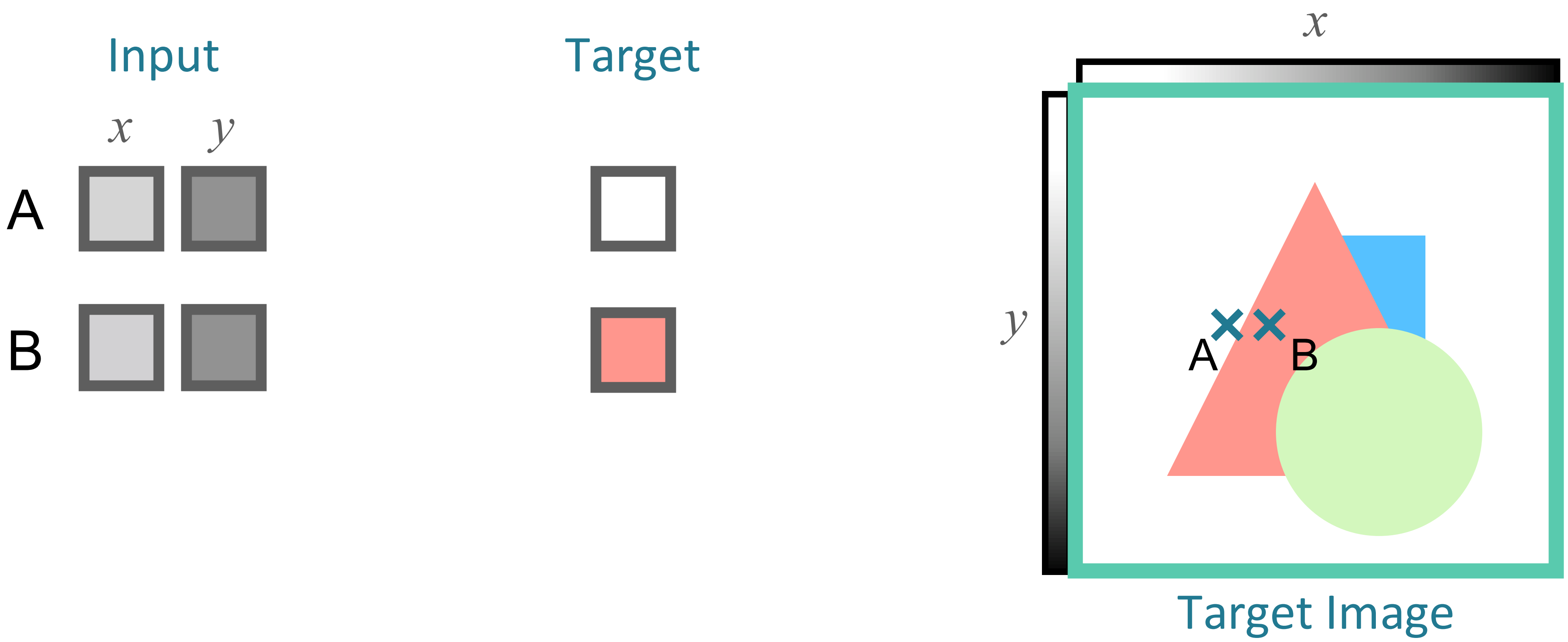




Target Image

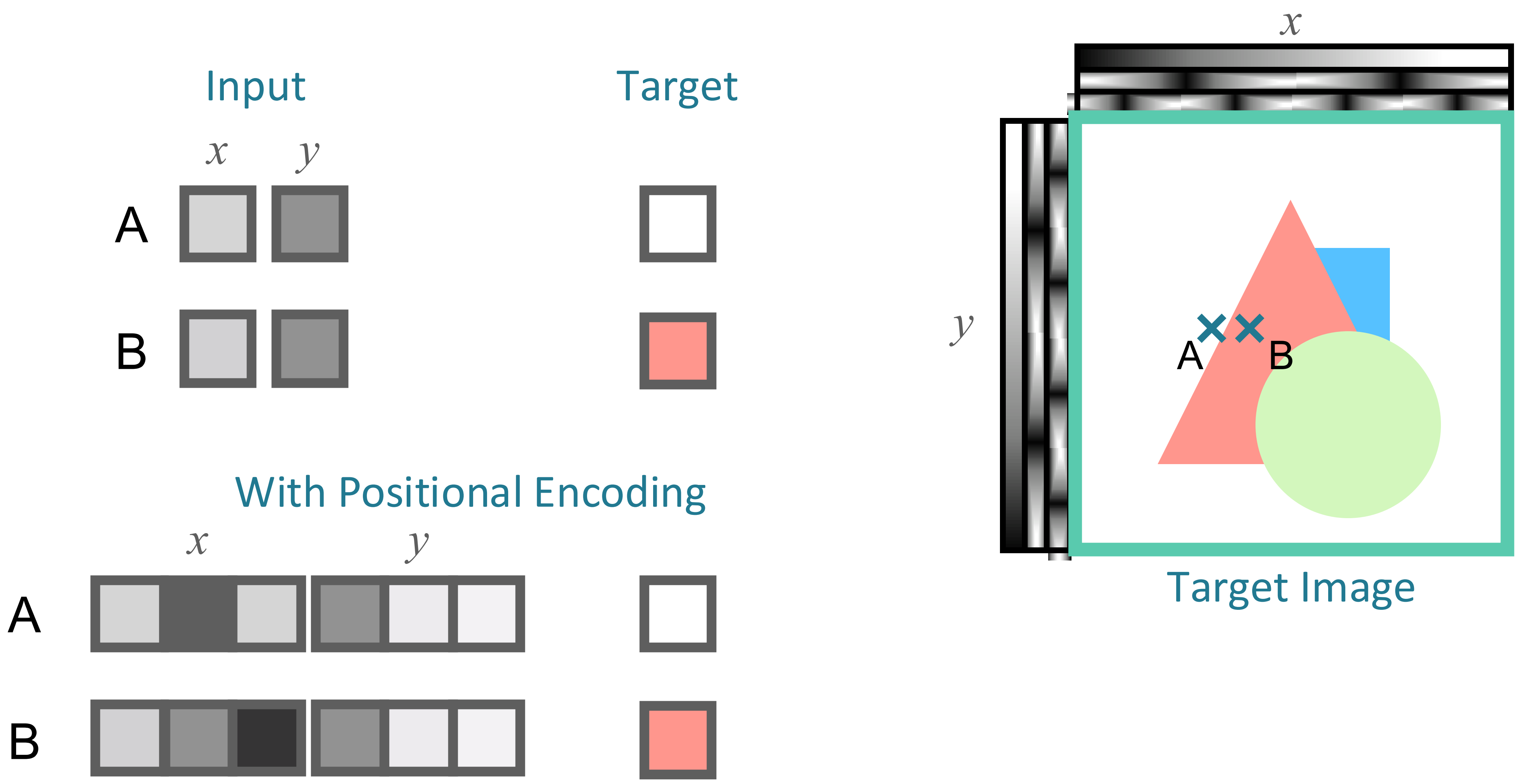
Slide credit: Matt Tancik

Why does positional encoding help?



Slide credit: Matt Tancik

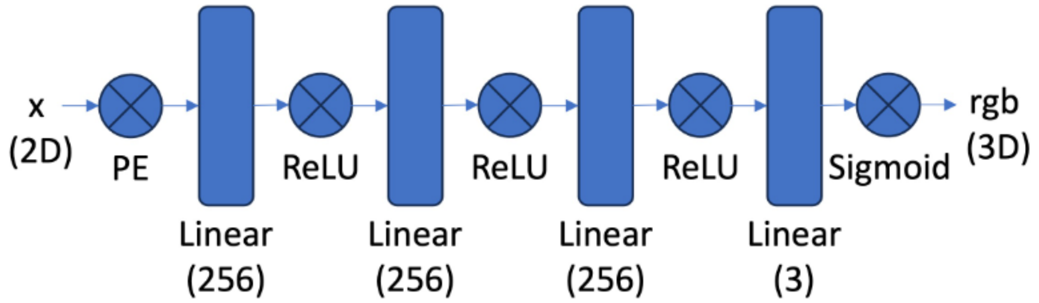
Why does positional encoding help?



Slide credit: Matt Tancik

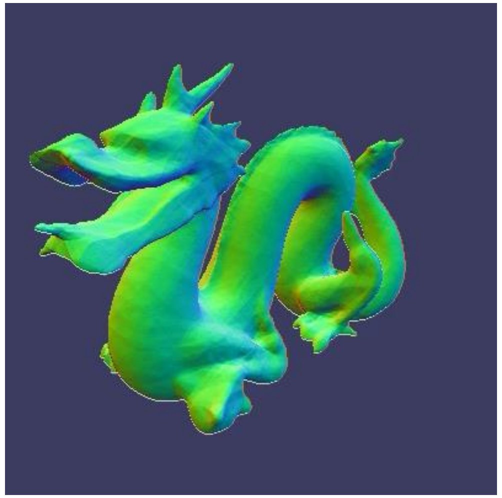
NeRF Project Part 1

- Fit a Neural Network to a single image
- Implement this network, and Positional Embedding (PE) and reconstruct an image:

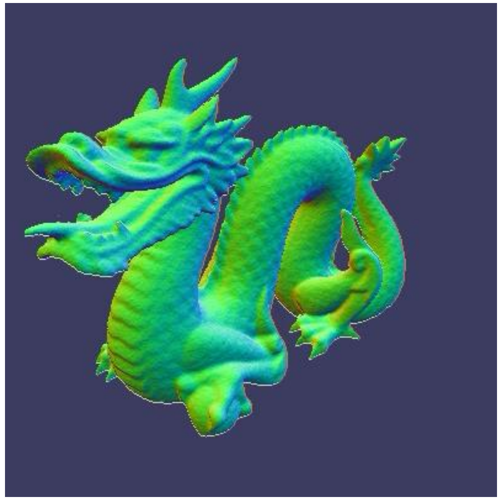


Coordinate-based MLPs can replace any low-dimensional array

Without Encoding



With Encoding



3D Shape

Slide credit: Matt Tancik

NeRF with and without positional encoding

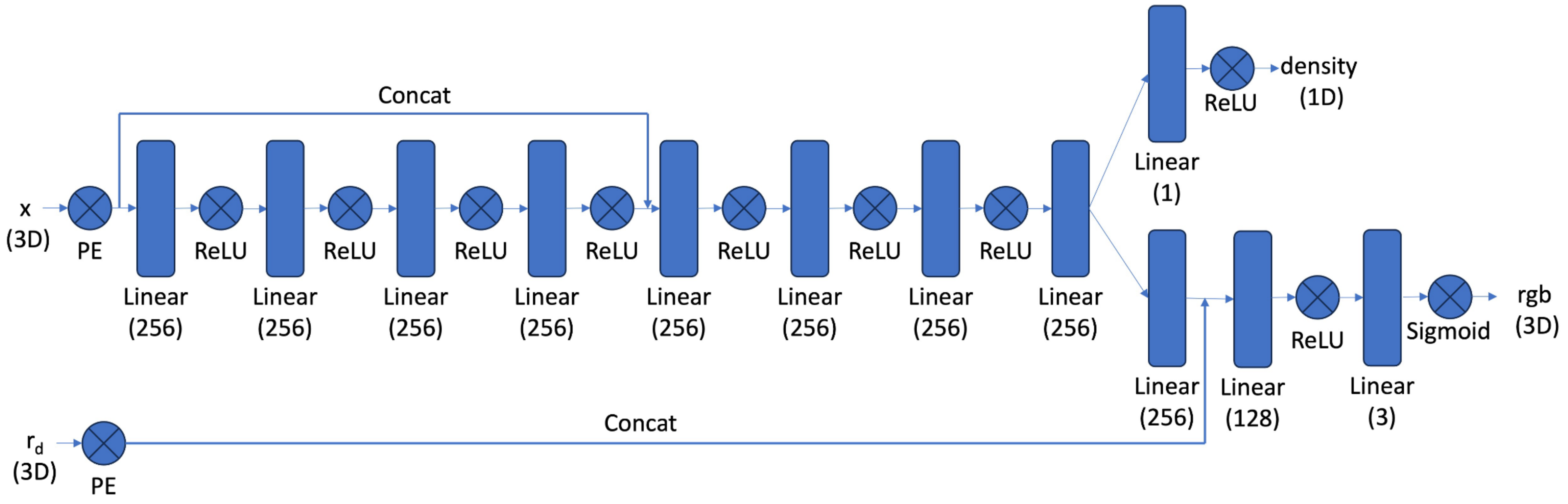
NeRF (Naive)

NeRF (with positional encoding)

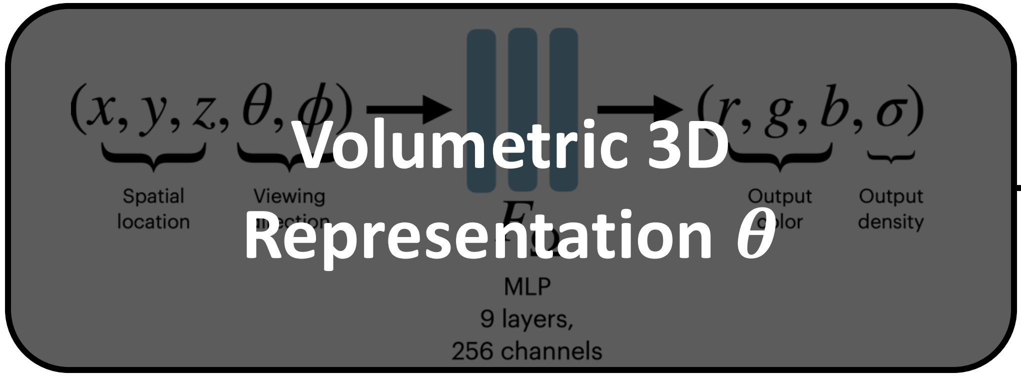
Slide credit: Matt Tancik

NeRF Network Architecture

Next section you will implement this:



Let’s go back to 3D



Now we need to render an image from this 3D representation in a differentiable manner

Differentiable
Volumetric Rendering

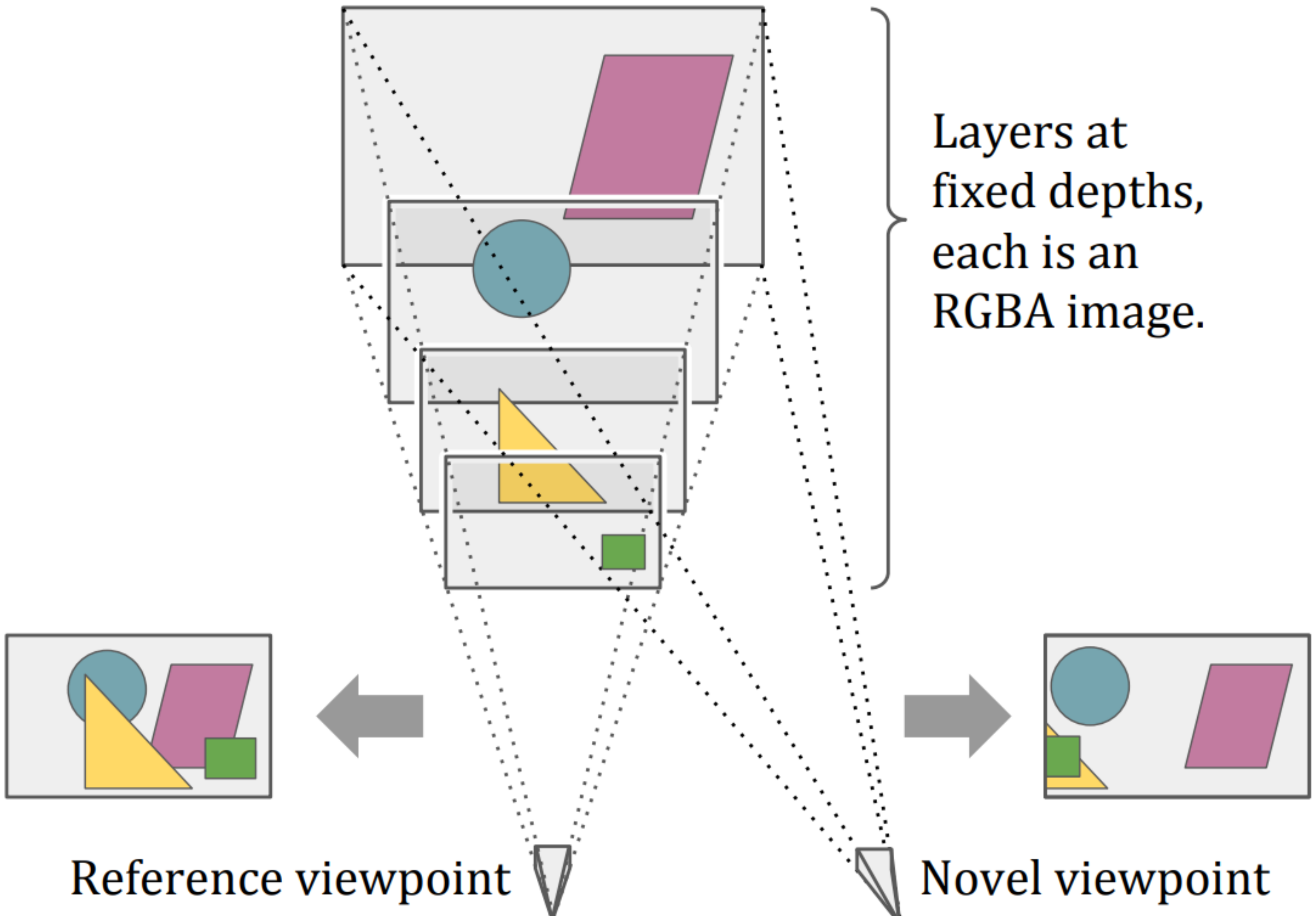
Rendered Image:
I’

“Training” Objective (aka Analysis-by-Synthesis):

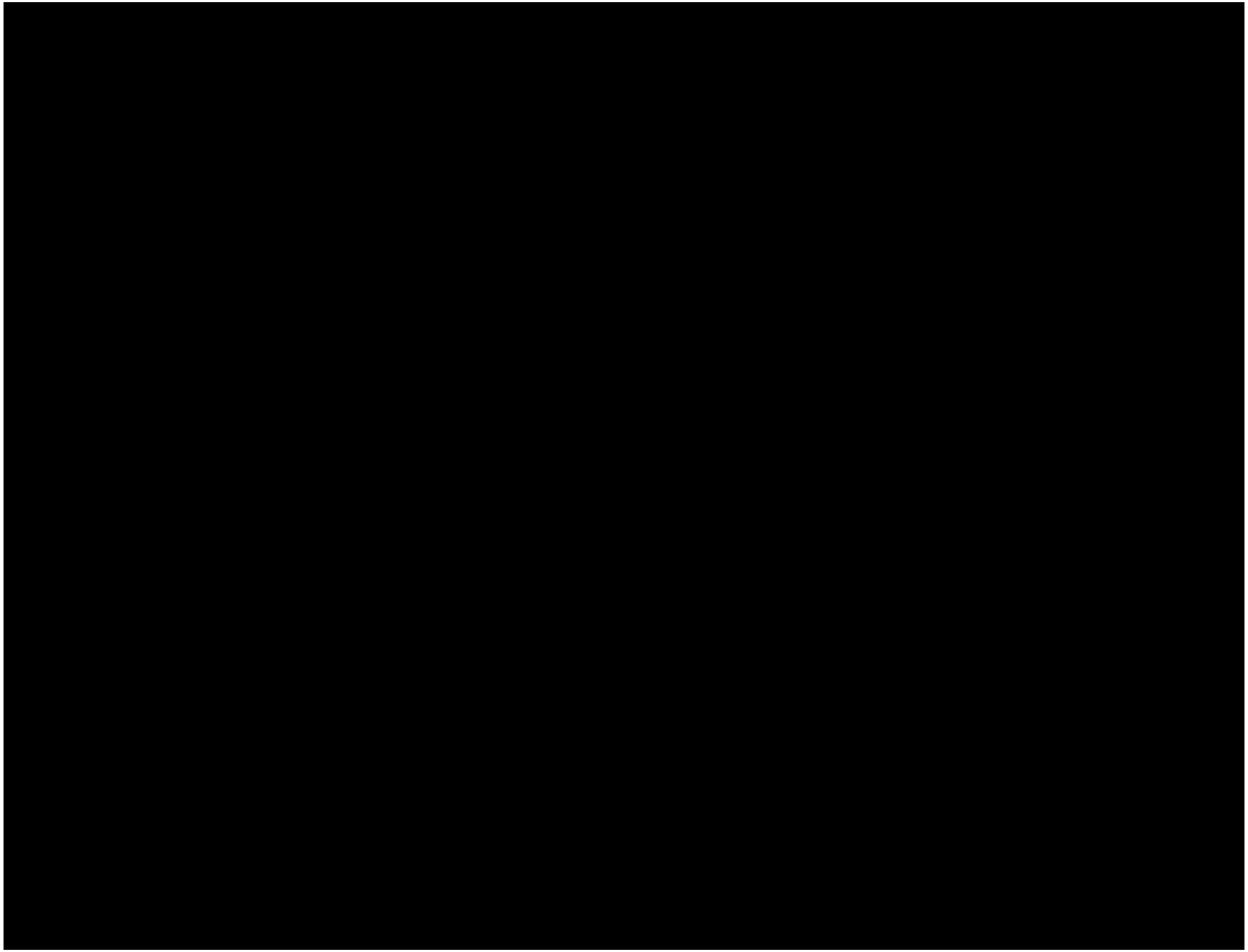
$$\min_{\theta} \| \text{Rendered Image: } I' - \text{Observed Image: } I \|_2$$

Differentiable Volumetric Rendering

A Precursor: Multi-plane Images



Zou et al. Stereo Magnification, SIGGRAPH 2018



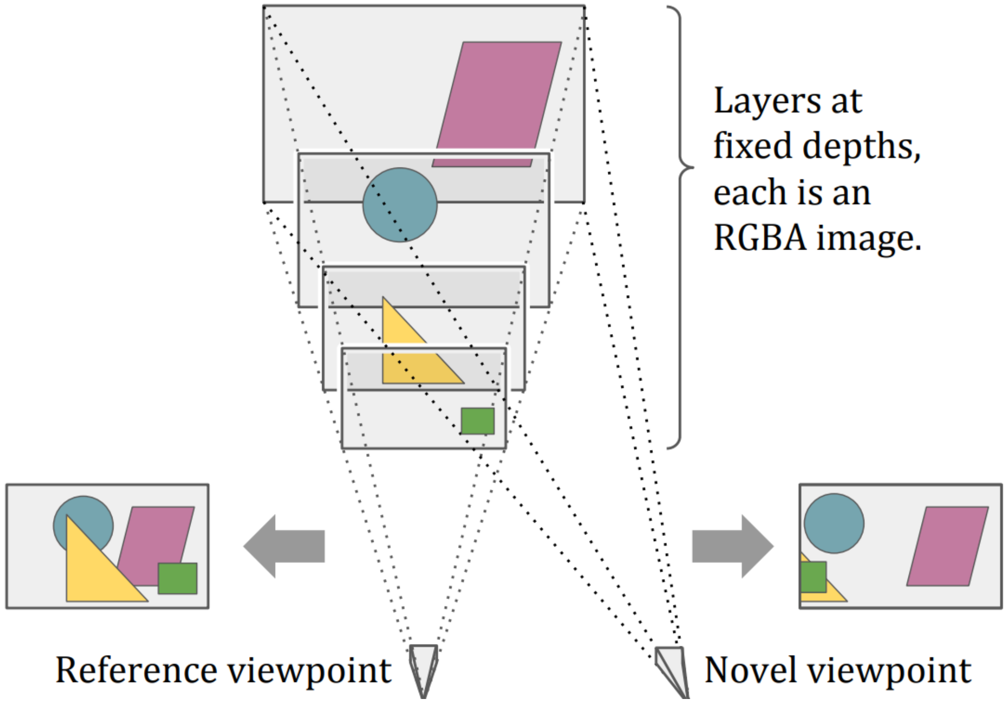
Multi-plane Camera at Disney

https://www.youtube.com/watch?v=YdHTlUGN1zw

Generating an Image MPI

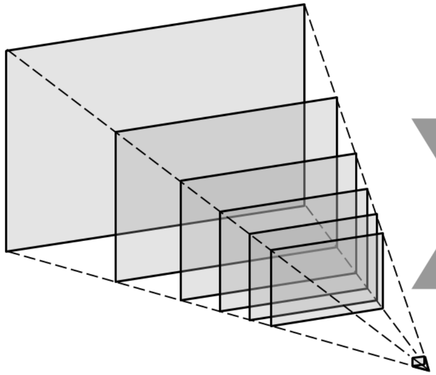
To render a novel view:

1. Homography warp the image from the new viewpoint
2. Alpha Blend each layer



Zou et al. Stereo Magnification, SIGGRAPH 2018

Sample Novel View Synthesis with a MPI

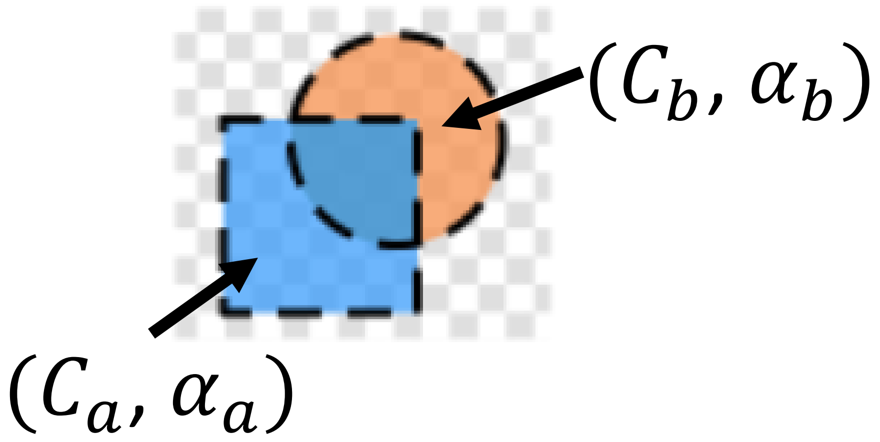


Single-view view synthesis with multiplane images, Tucker and Snavely CVPR 2020

Also called front-to-back compositing or “over” operation

Alpha Blending

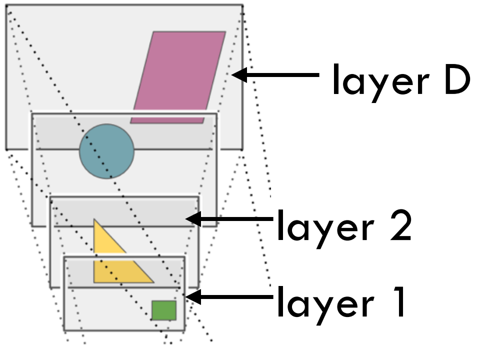
for two image case, A and B, both partially transparent:



$$I = C_a\,\alpha_a + C_b\,\alpha_b \underbrace{(1-\alpha_a)}_{\text{How much light is the previous layer lettina through?}}$$

General D layer case:

$$I = \sum_{i=1}^{D} C_i \alpha_i \prod_{j=1}^{i-1} (1-\alpha_j)$$



What is missing in MPIs?

- Look at it from the side??
- You'll see all the edges!!

➔ Limited camera mobility

NeRF overcomes this problem, because it's defined everywhere

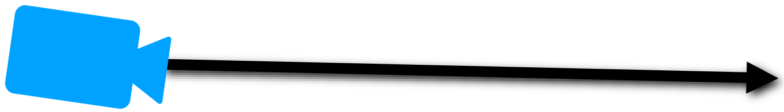
Volumetric Rendering behaves similarly to alpha compositing

Back to NeRFs

Neural Volumetric Rendering

Through Volumetric Representation (No surfaces)!

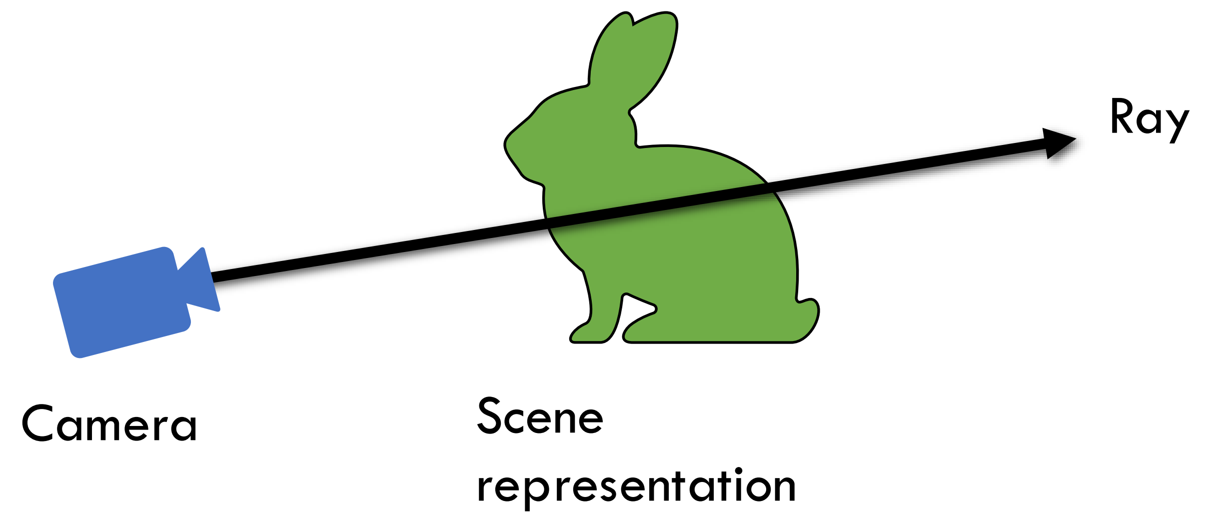
computing color along rays through 3D space



What color is this pixel?

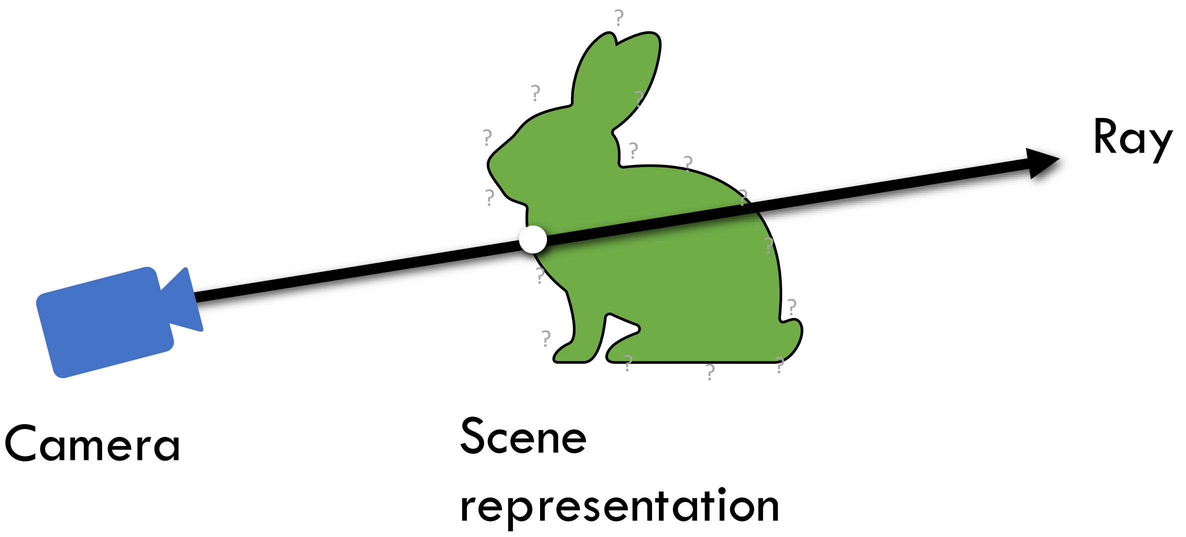
Lots of slide from here on modified from Ben Mildenhall

Surface vs. volume rendering



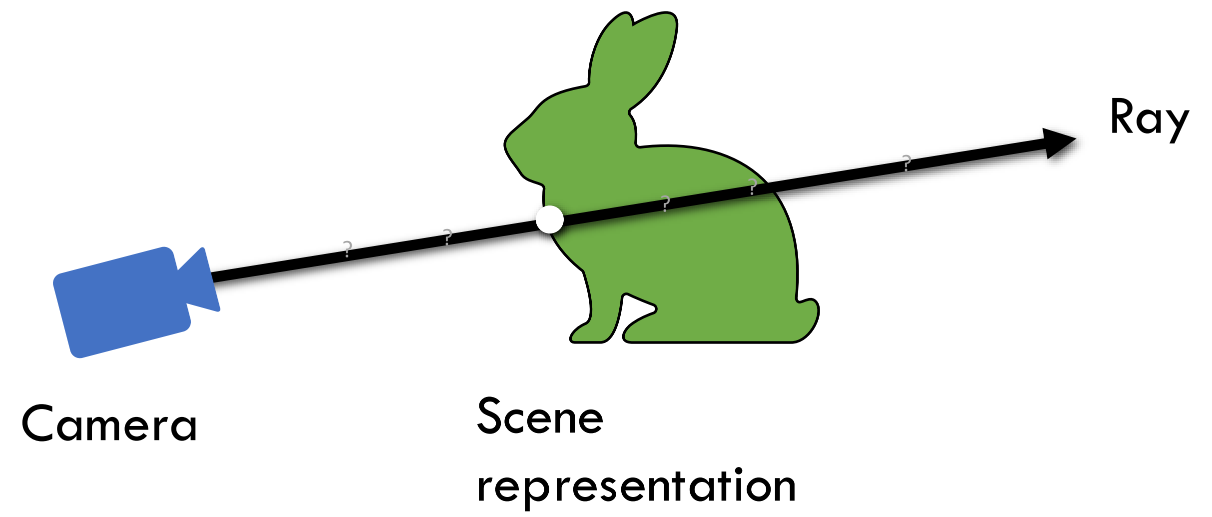
Want to know how ray interacts with scene

Surface vs. volume rendering



Surface rendering — loop over geometry, check for ray hits

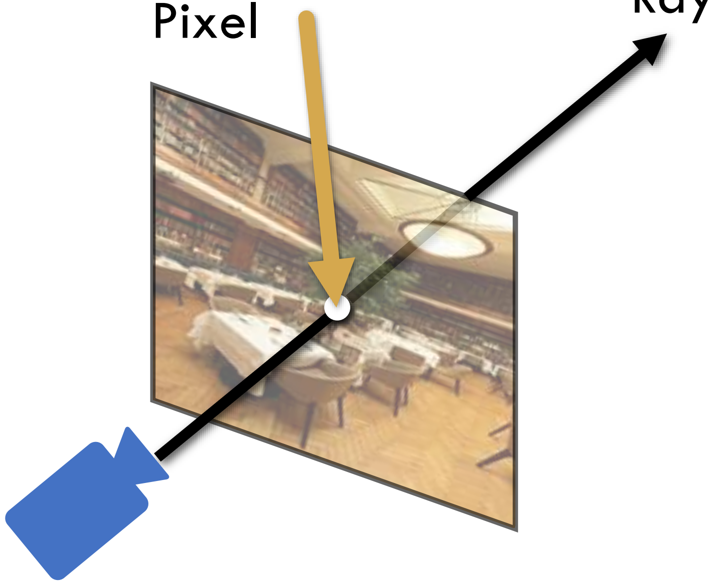
Surface vs. volume rendering



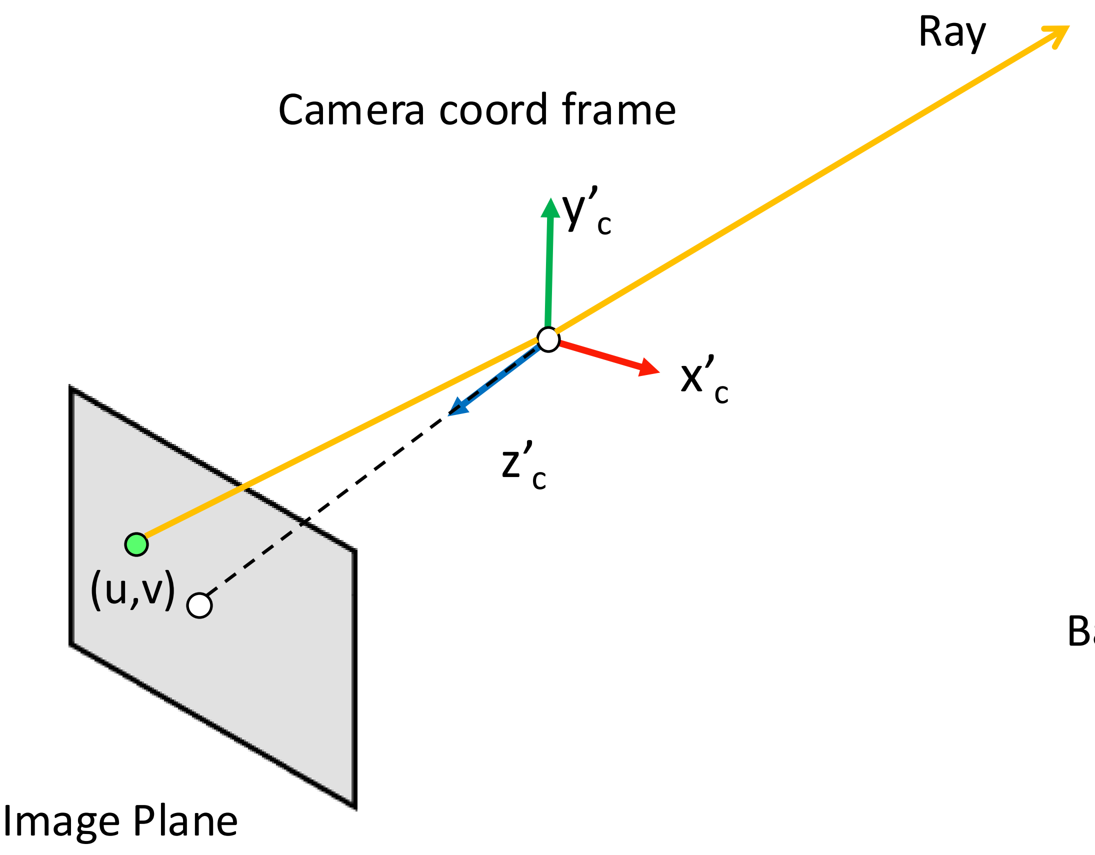
Volume rendering — loop over ray points, query geometry

Recap: Cameras and rays

- We need the mathematical mapping from (*camera*, *pixel*) → *ray*
- Then can abstract underlying problem as learning the function *ray* → *color*



Compute the Ray



3D to 2D:
(point)

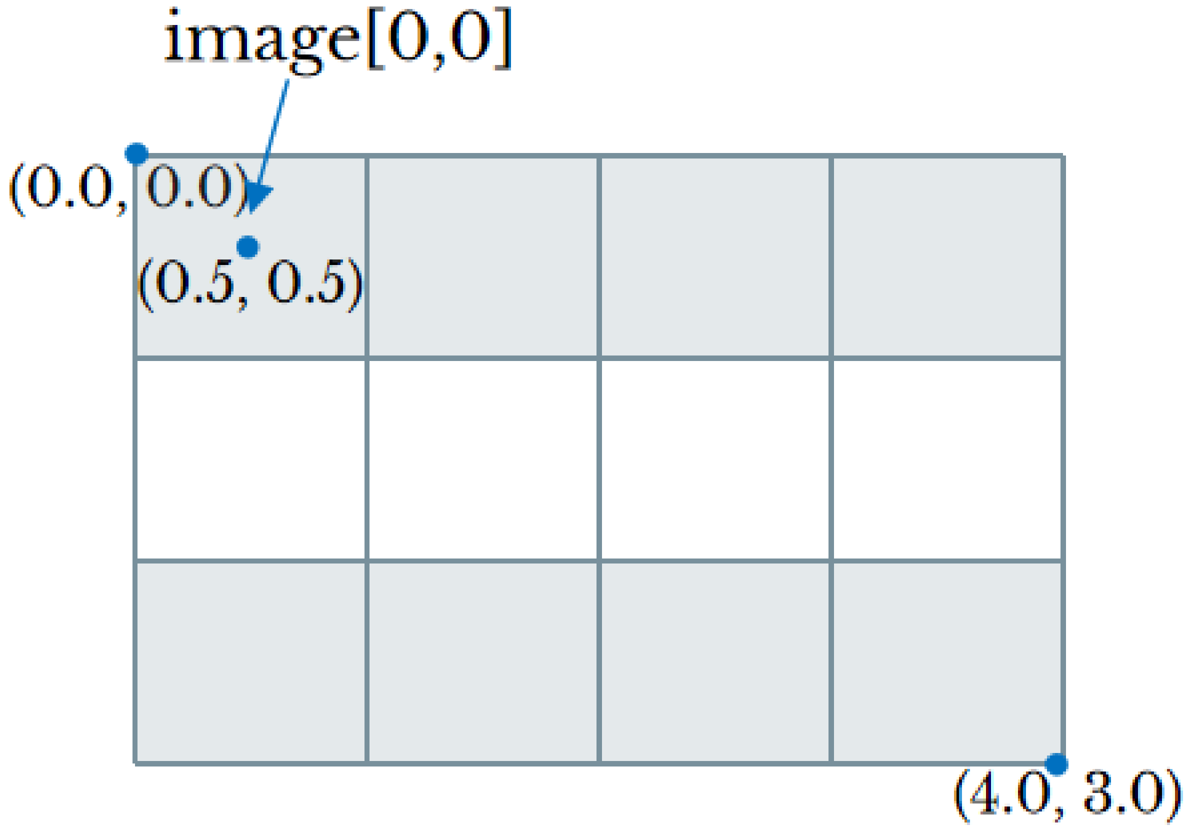
$$u = f_x \frac{x_c}{z_c} + o_x$$
$$v = f_y \frac{y_c}{z_c} + o_y$$

2D to 3D:
(ray)
Back projection

$$x = \frac{z}{f_x}(u - o_x)$$
$$y = \frac{z}{f_y}(v - o_y)$$
$$z > 0$$

Details:

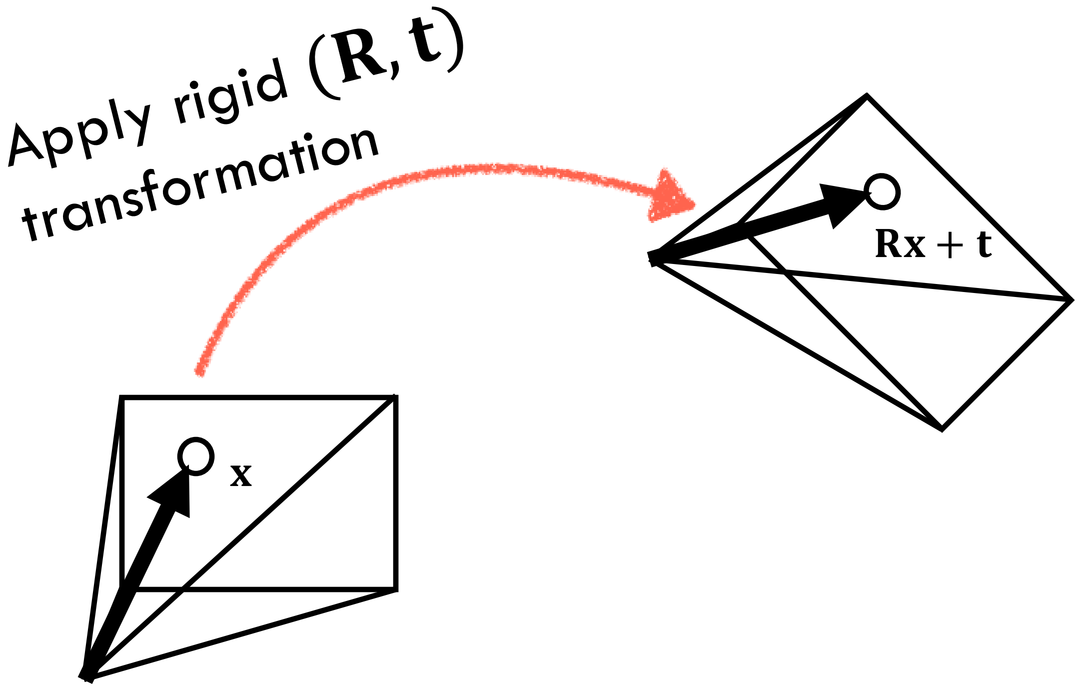
A half-pixel offset — add 0.5 to *i* and *j* so ray precisely hits pixel center



Want: Ray in the World

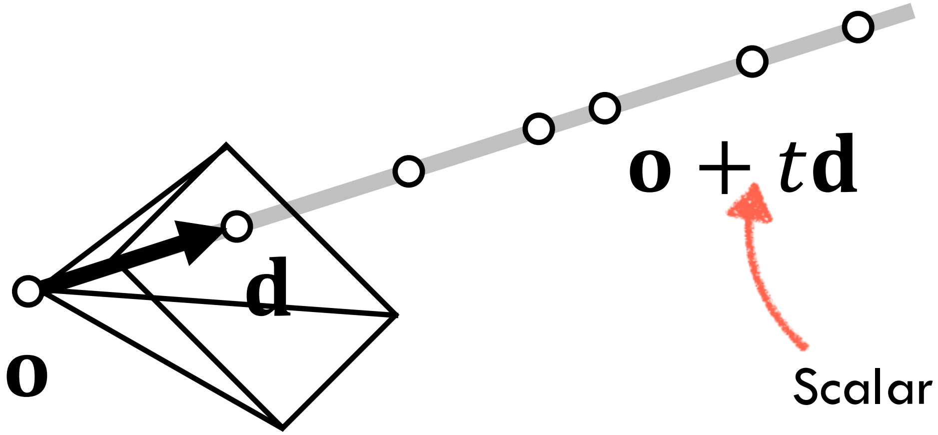
- What coordinate space is the current ray in?

- Convert it to World!



Calculating points along a ray

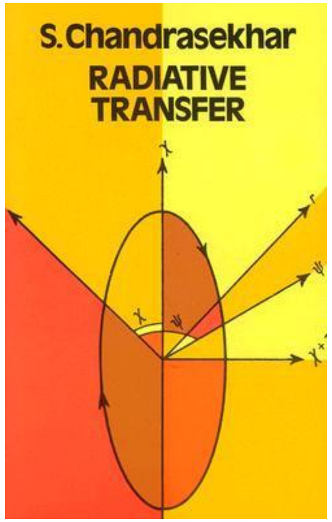
In the world coordinate frame:

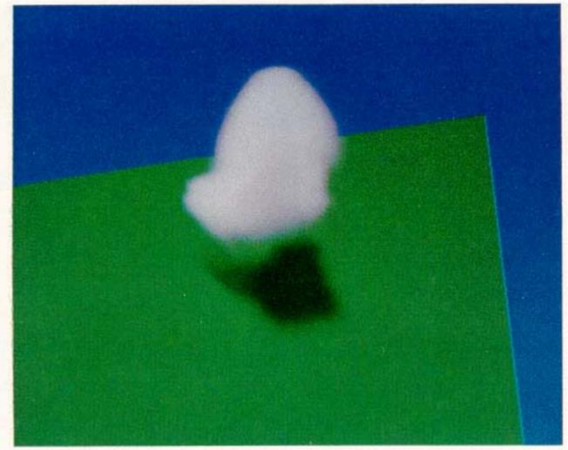


Scalar t controls distance along the ray

History of volume rendering

In Early computer graphics





Ray tracing simulated cumulus cloud [Kajiya]

- Theory of volume rendering co-opted from physics in the 1980s: absorption, emission, out-scattering/in-scattering
- Adapted for visualising medical data and linked with alpha compositing
- Modern path tracers use sophisticated Monte Carlo methods to render volumetric effects

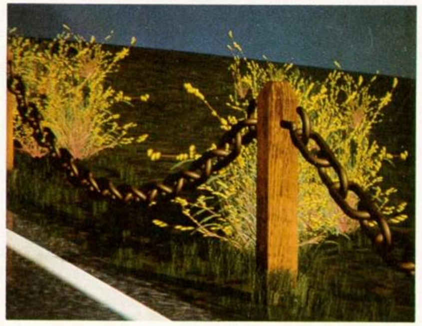
Chandrasekhar 1950, Radiative Transfer

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



Pt.Reyes = Foreground over Hillside over Background.

Alpha compositing [Porter and Duff]

Porter and Duff 1984, Compositing Digital Images

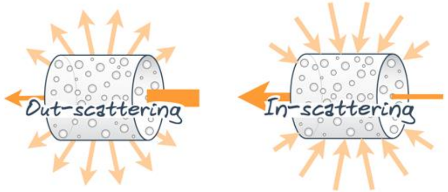
Volume rendering for visualization

Medical data visualisation [Levoy]

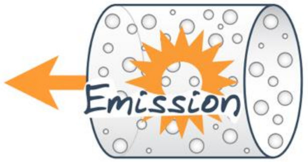
- Theory of volume rendering co-opted from physics in the 1980s: absorption, emission, out-scattering/in-scattering
- Alpha rendering developed for digital compositing in VFX movie production
- Volume rendering applied to visualise 3D medical scan data in 1990s

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

Absorption

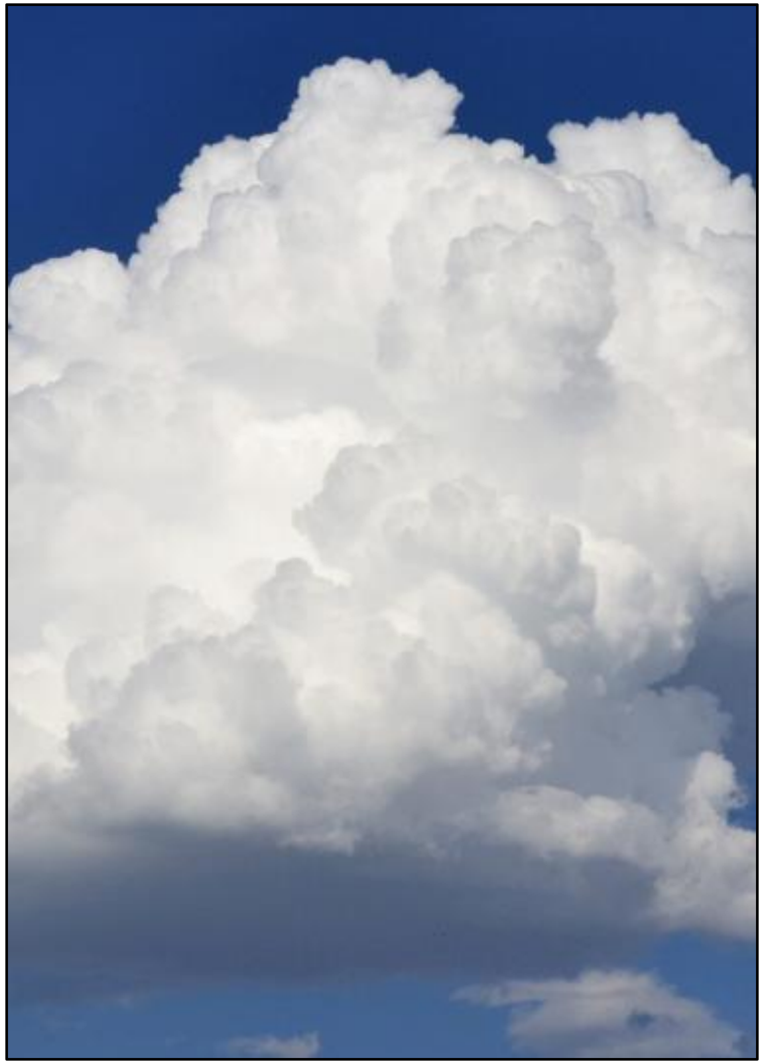


Scattering



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

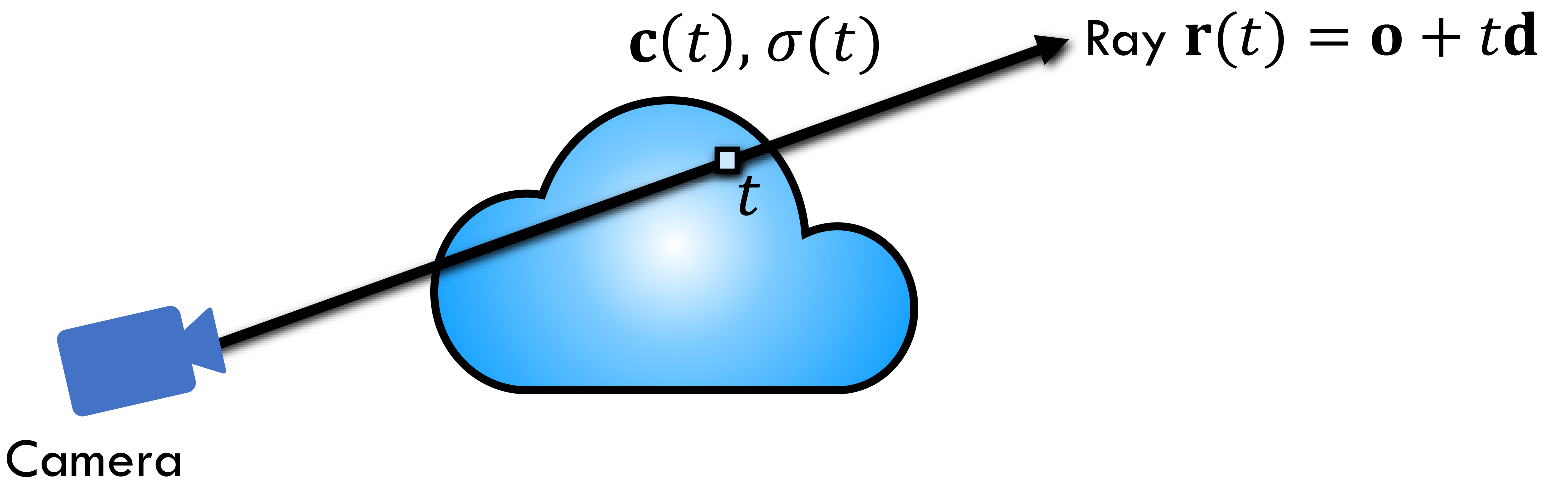
Volume rendering derivations

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



at a point on the ray $\mathbf{r}(t)$, we can query color $\boldsymbol{c}(t)$ and density $\sigma(t)$

How to integrate all the info along the ray to get a color per ray?

Idea: Expected Color

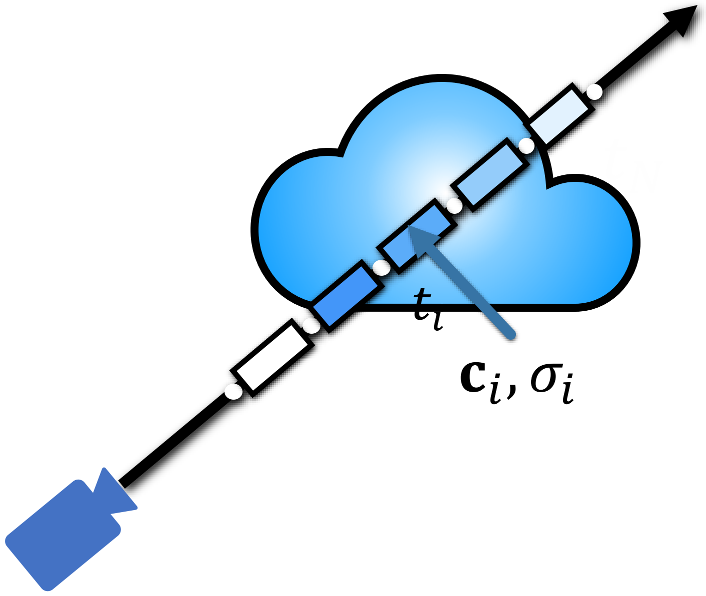
- Pose probabilistically.
- Each point on the ray has a probability to be the first "hit" : $P[first\ hit\ at\ t]$
- Color per ray = Expected value of color with this probability of first "hit"

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

$$\boldsymbol{c}(\boldsymbol{r}) = \int_{t_0}^{t_1} P[first\ hit\ at\ t]\boldsymbol{c}(t)dt$$

$$\approx \sum_{t=0}^{T} P[first\ hit\ at\ t]\boldsymbol{c}(t)$$

$$\approx \sum_{t=0}^{T} w_t\boldsymbol{c}(t)$$



Differentiable Volumetric Rendering Formula

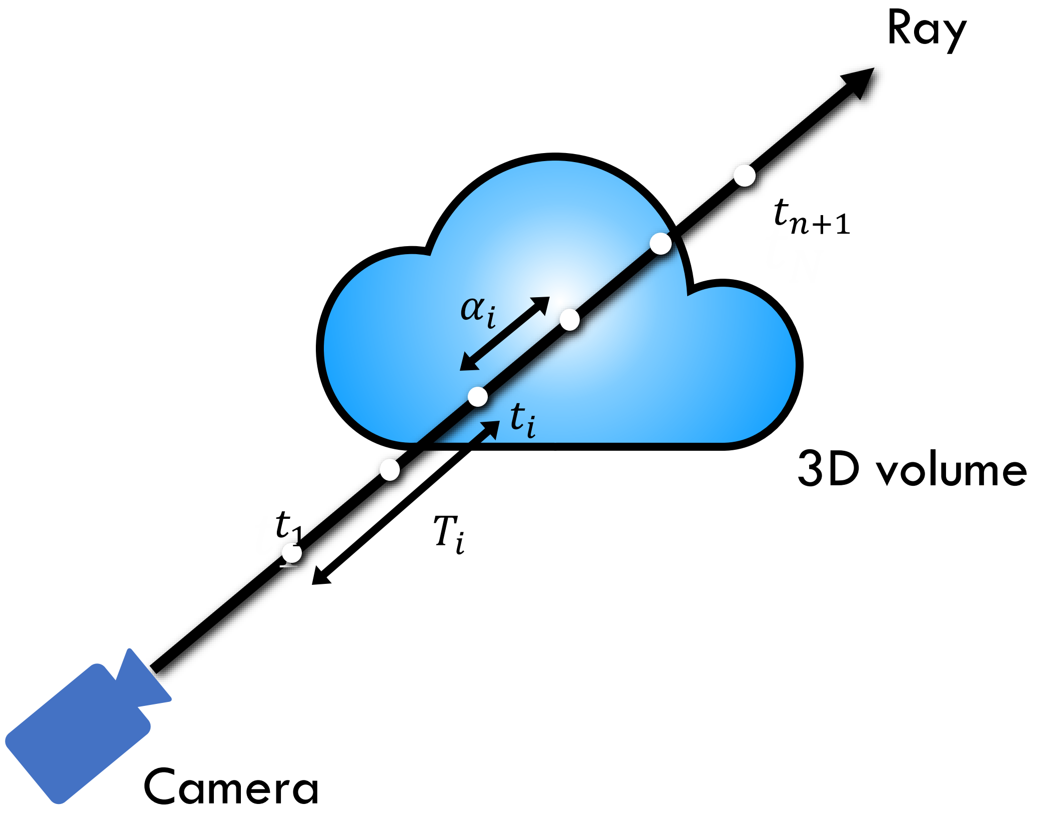
for a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

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

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

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*:

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

Summary

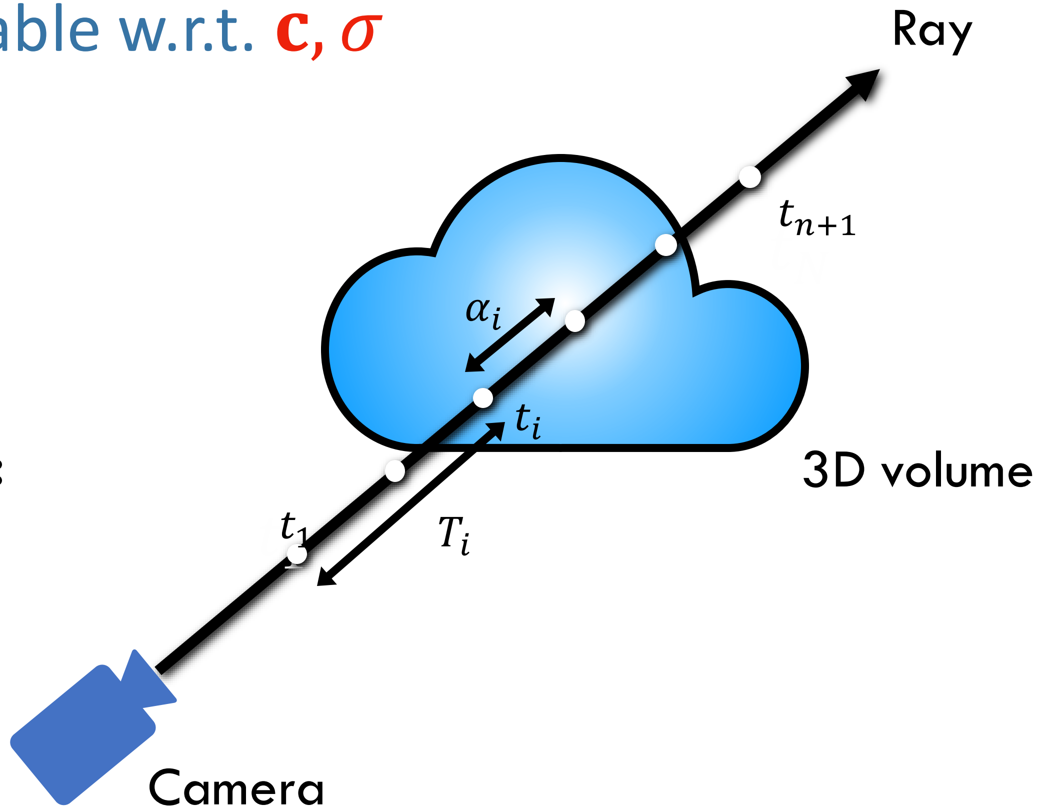
for a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

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

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

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*:

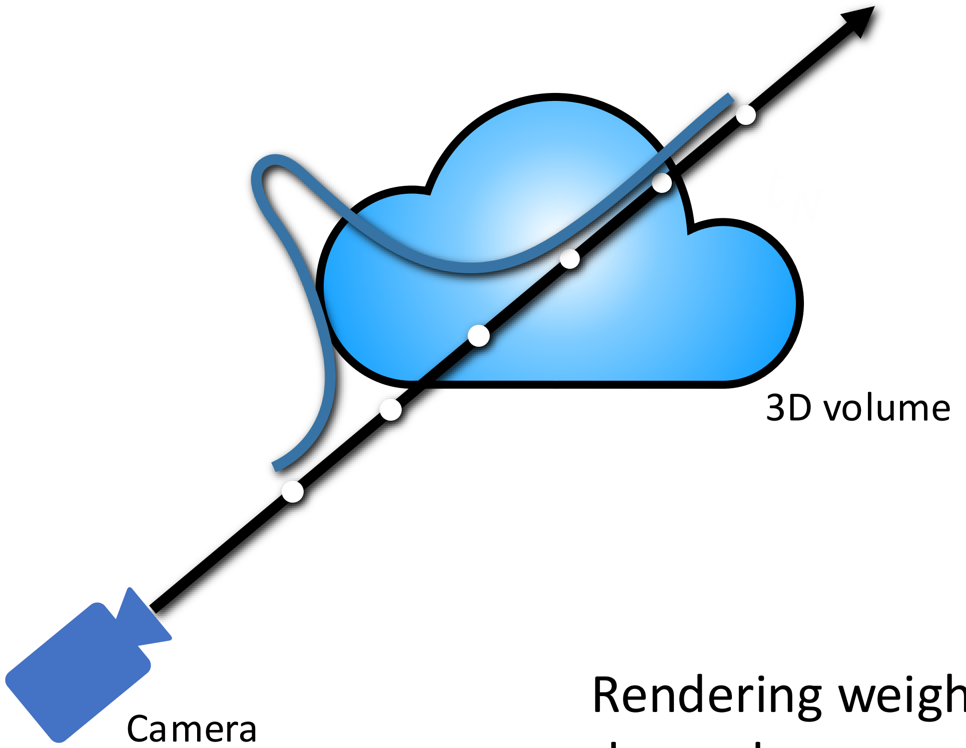
$$\alpha_i = 1 - \exp(-\sigma_i \delta_i)$$

Complete derivation

- If you want a complete derivation of the volrend equation, see this https://drive.google.com/file/d/1QsCK5V0d6DSc0QGcKsV97u83JFd-dFoz/view
- From slide 35
- Or https://arxiv.org/pdf/2209.02417

Visual intuition: rendering weights is specific to a ray

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

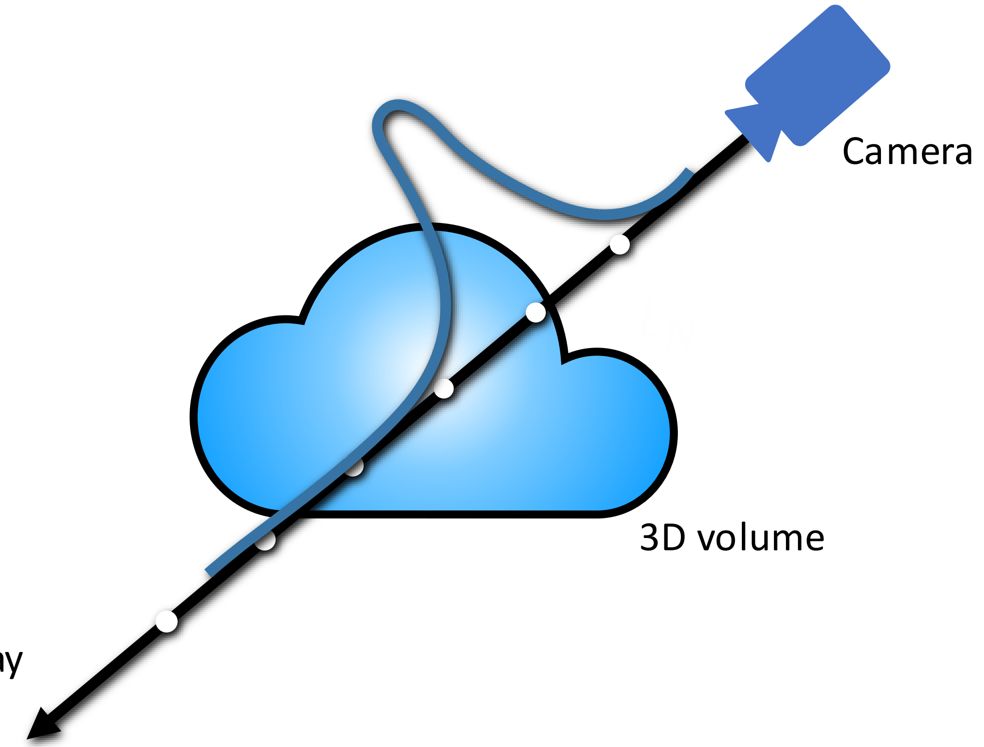


Rendering weights are not a 3D function — depends on ray, because of tranmisttance!

69

Visual intuition: rendering weights is specific to a ray

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



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

70

Rendering weight PDF is important

Remember, expected color is equal to

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

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

You can also render entities other than color in 3D, for example it's depth, or any other N-D vector $\boldsymbol{v}_i$

Volume rendered "feature" $= \sum_i w_i\boldsymbol{v}_i$

Rendering weight PDF is important — depth

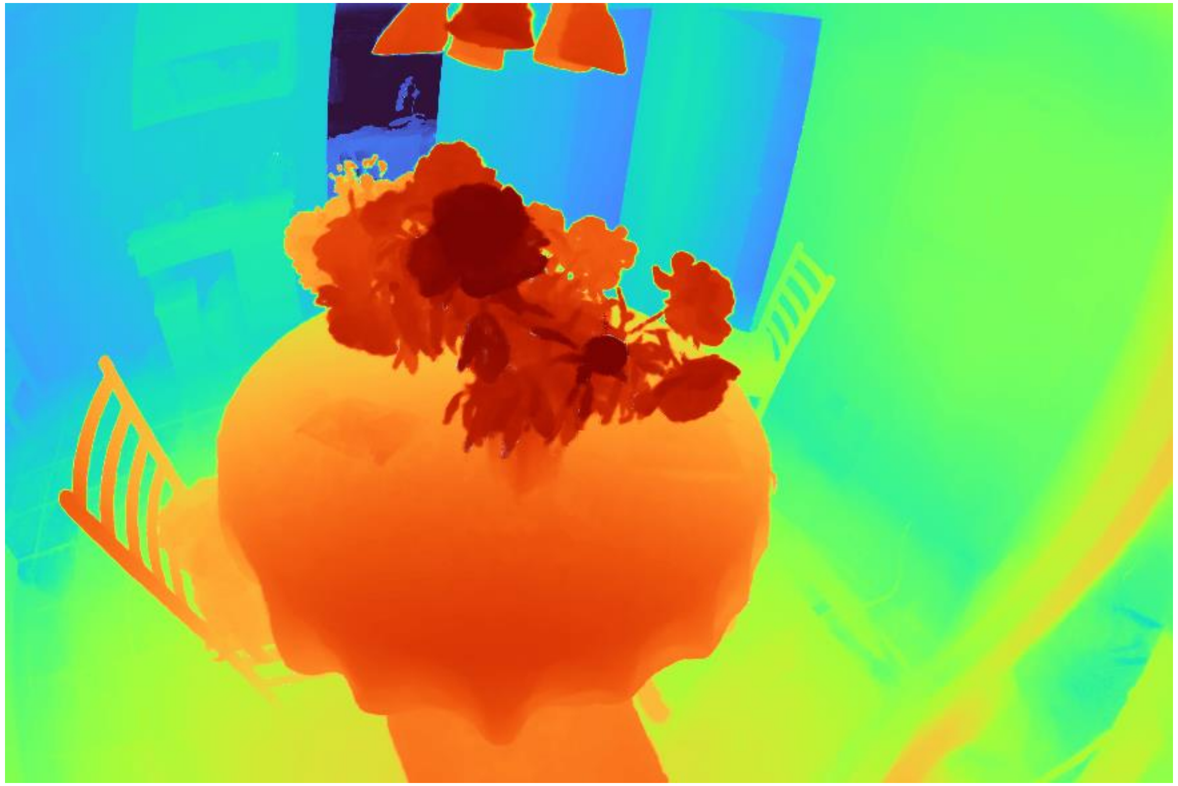
We can use this distribution to compute expectations for other quantities, e.g. "expected depth":

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

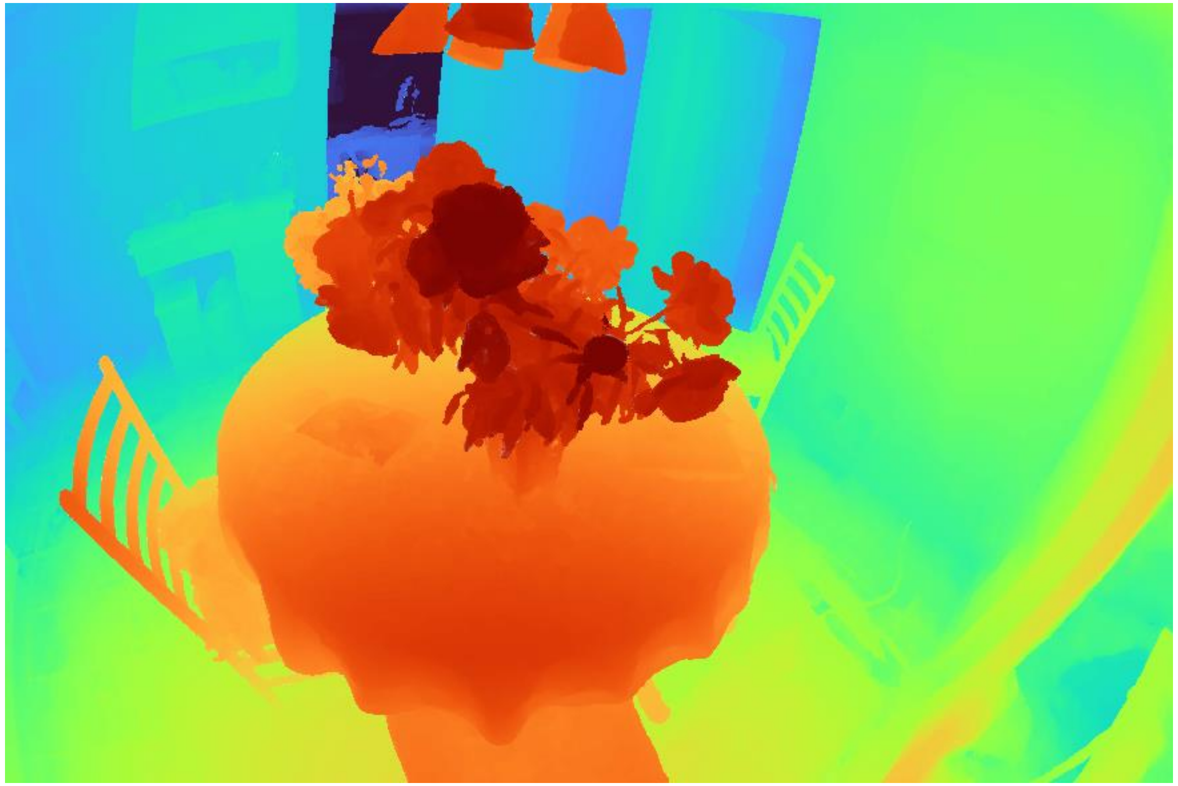
This is often how people visualise NeRF depth maps.

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

Rendering weight PDF is important — depth

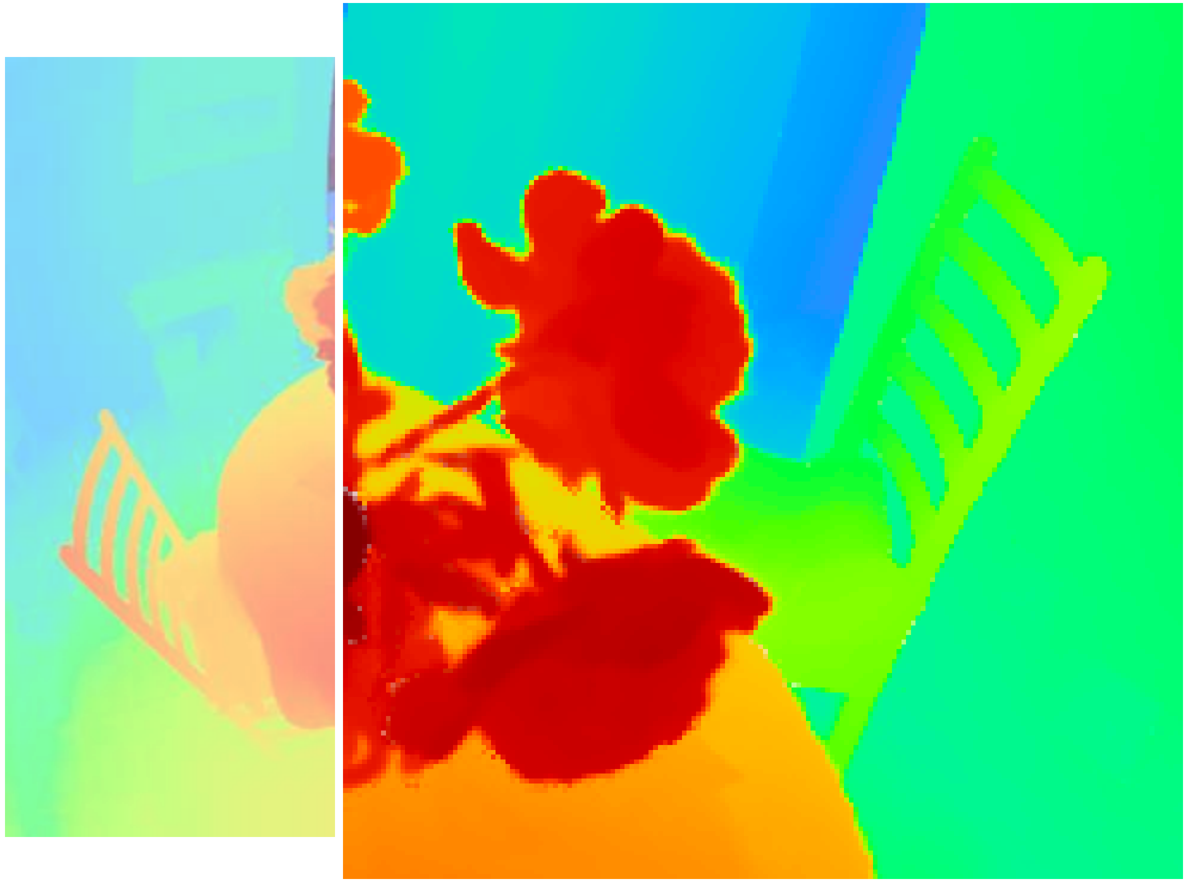


Mean depth

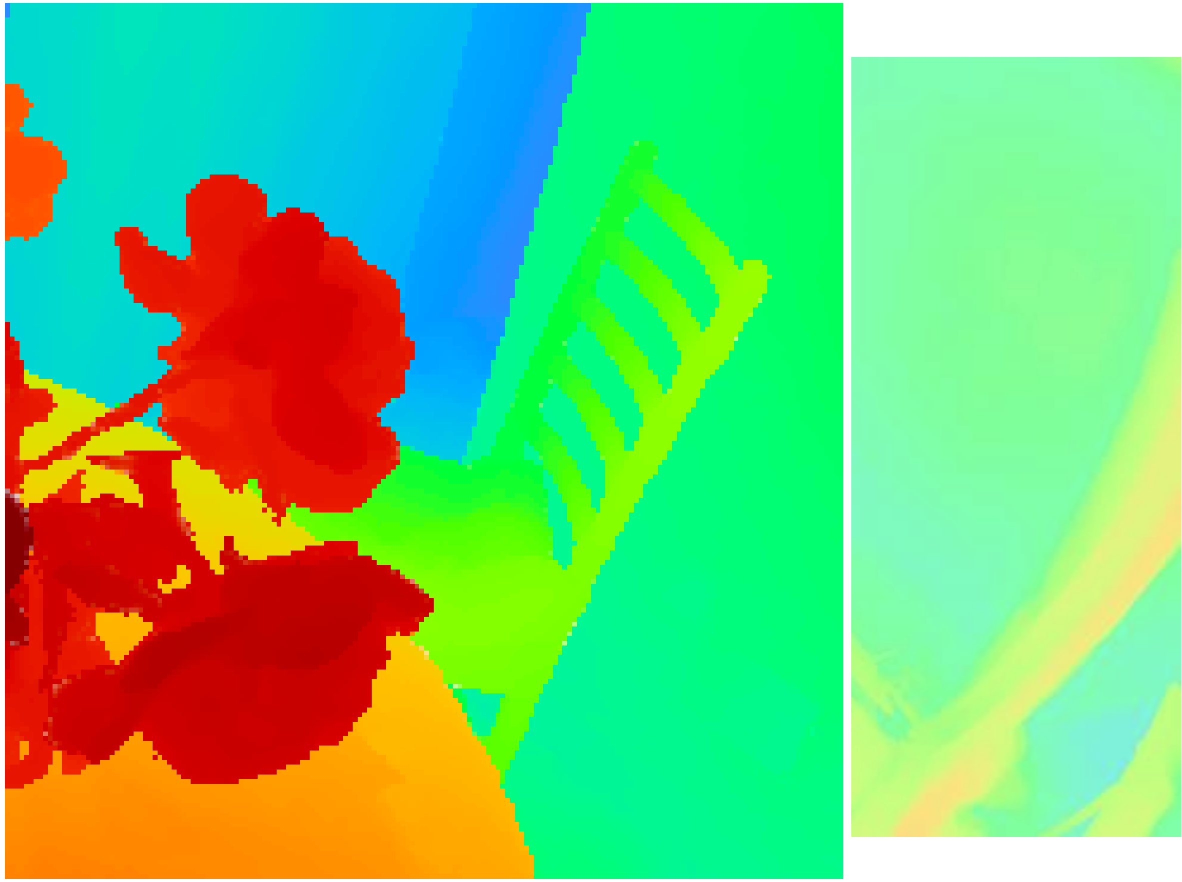


Median depth

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.)

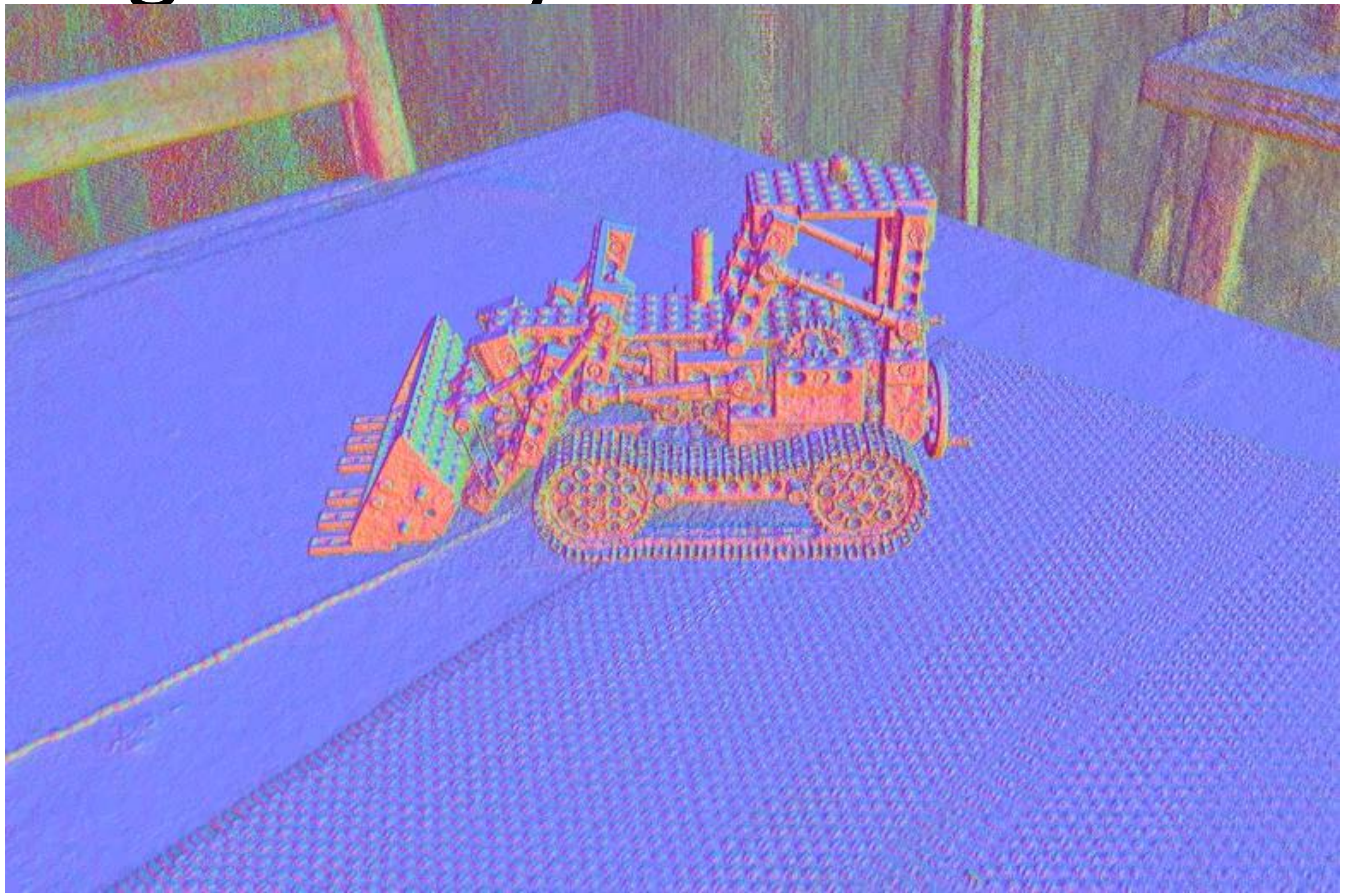
$$\sum_i T_i \alpha_i \mathbf{v}_i$$

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

Volume Rendering CLIP features

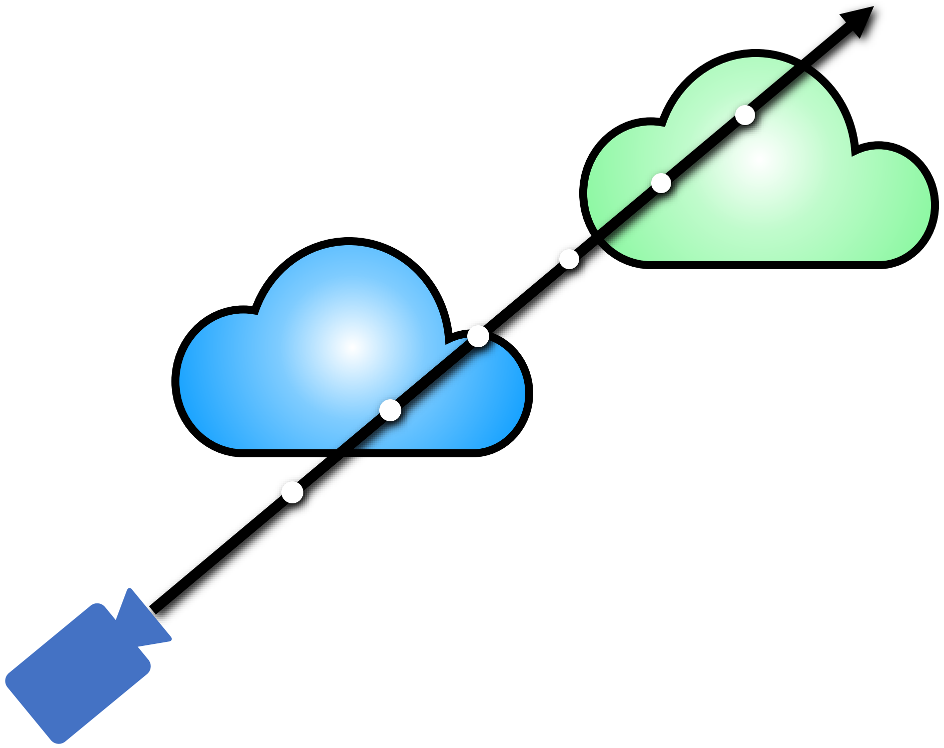
LERF: Language Embedded Radiance Fields, Kerr* and Kim* et al. ICCV 2023

Density as geometry

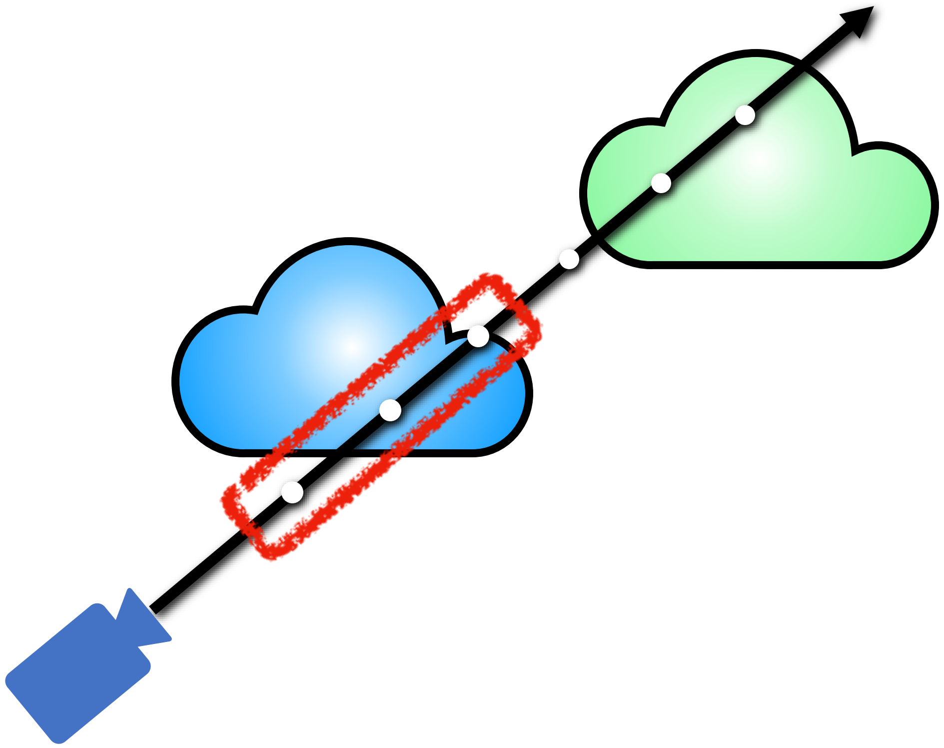


Normal vectors (from analytic gradient of density)

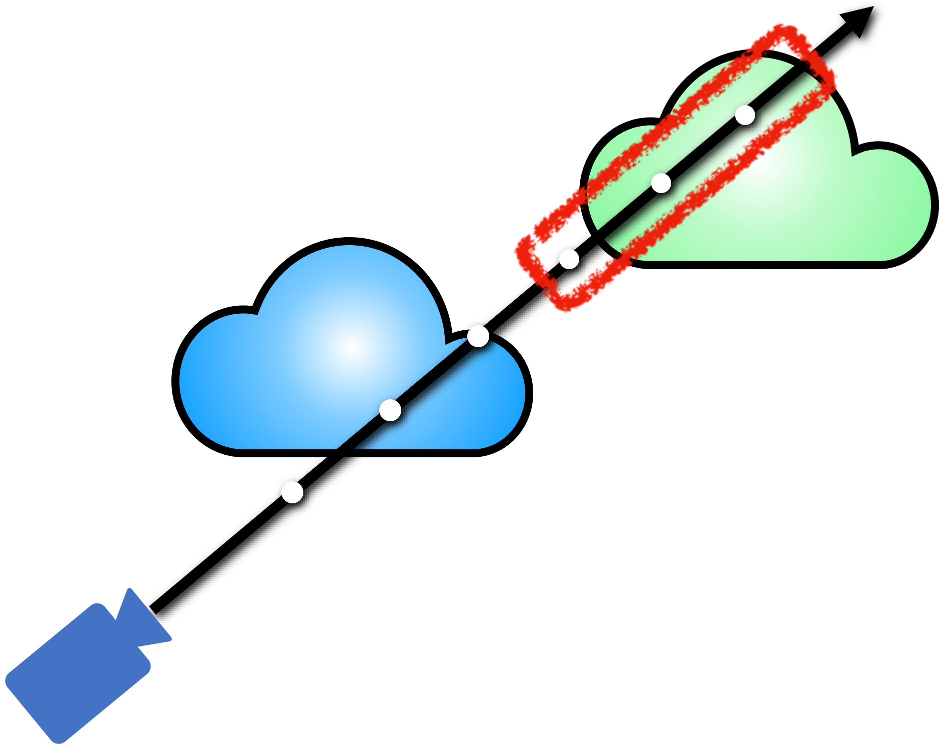
Alpha mattes and compositing



Alpha mattes and compositing



Alpha mattes and compositing



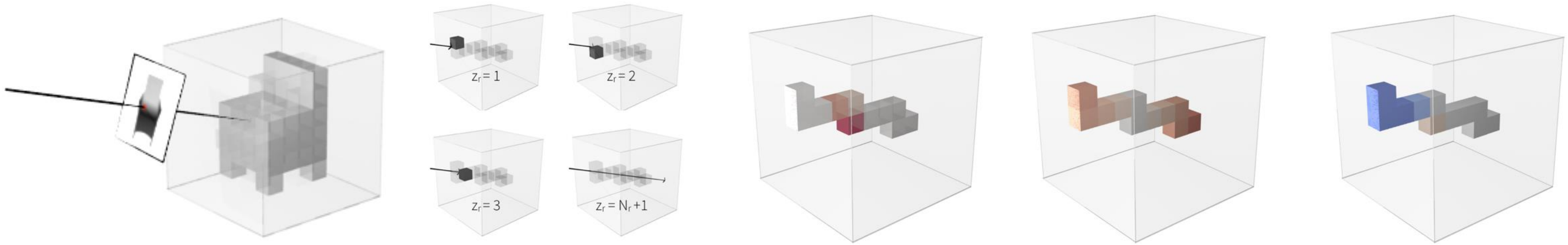
Alpha mattes and compositing

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

Poole et al 2022, DreamFusion

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

Previous Papers



Differentiable ray consistency work used a forward model with "probabilistic occupancy" to supervise 3D-from-single-image prediction.
Same rendering model as alpha compositing!

$$p(z_r = i) = \begin{cases} (1 - x_i^r) \prod_{j=1}^{i-1} x_j^r, & \text{if } i \leq N_r \\ \prod_{j=1}^{N_r} x_j^r, & \text{if } i = N_r + 1 \end{cases}$$

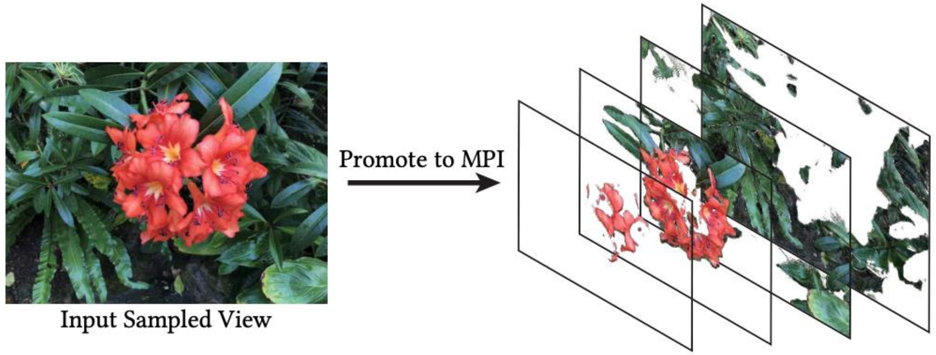
Tulsiani et al 2017, Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency

Similar Ideas before NeRF

Multiplane image methods

Stereo Magnification (Zhou et al. 2018)
Pushing the Boundaries… (Srinivasan et al. 2019)
Local Light Field Fusion (Mildenhall et al. 2019)
DeepView (Flynn et al. 2019)
Single-View… (Tucker & Snavely 2020)

Typical deep learning pipelines - images go into a 3D CNN, big RGBA 3D volume comes out



Neural Volumes
(Lombardi et al. 2019)

Direct gradient descent to optimize an RGBA volume, regularized by a 3D CNN