Neural Volumetric Rendering: NeRF, etc.

ML Collective DLCT

June 17, 2021

Ben Mildenhall
Google Research
bmild.github.io

Neural Volumetric Rendering

Neural Volumetric Rendering

querying the radiance value
along rays through 3D space

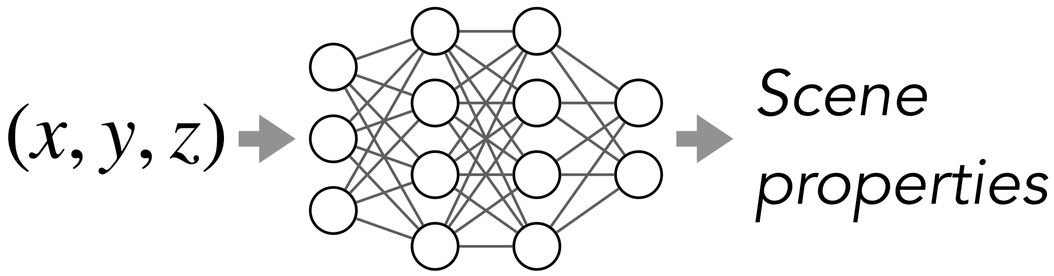
What colour?

Neural Volumetric Rendering

continuous, differentiable
rendering model without
concrete ray/surface intersections

Neural Volumetric Rendering

using a neural network as a scene representation, rather than a voxel grid of data

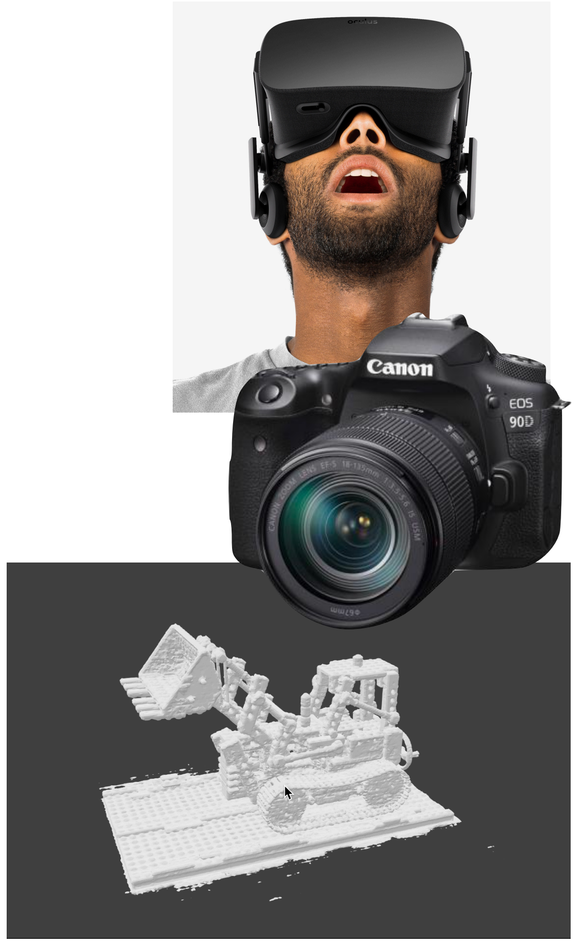


Motivation: novel view synthesis

Inputs: sparse, unstructured photographs of a scene

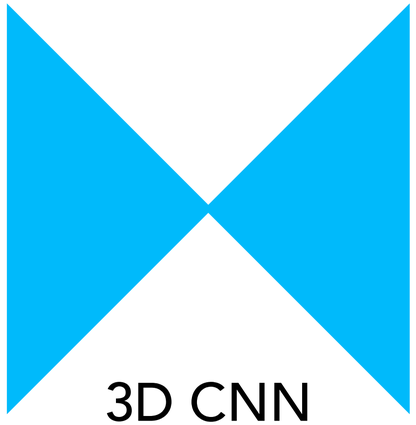
Outputs: representation allowing us to render *new* views of that scene

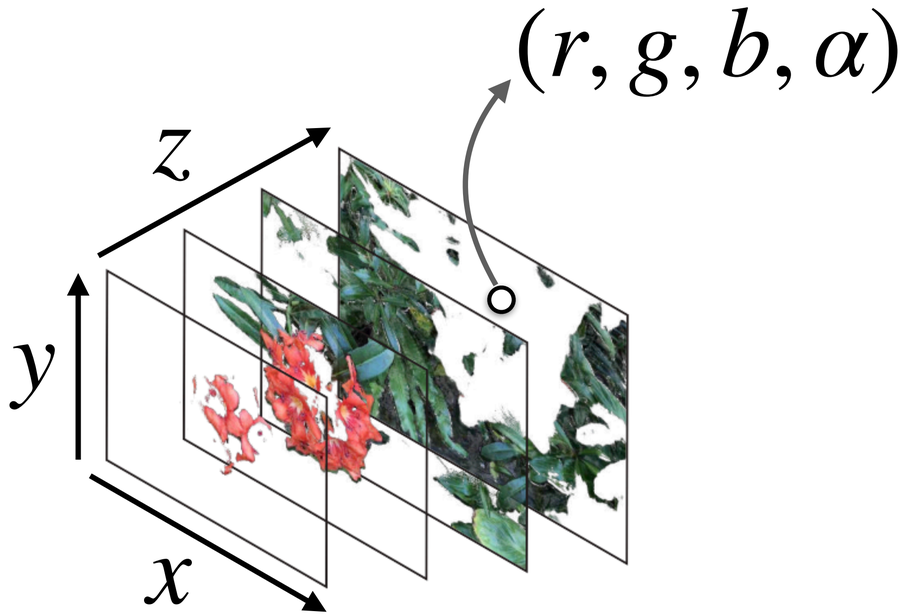
Why care about view synthesis?



- Key to enabling great virtual reality experiences
- Only requires 2D images for training and evaluation
- Research progress can inform many other 3D vision tasks (3D reconstruction, material estimation, relighting, etc)

Inspiration from view synthesis:
Predicting a 3D voxel grid of RGB-alpha values

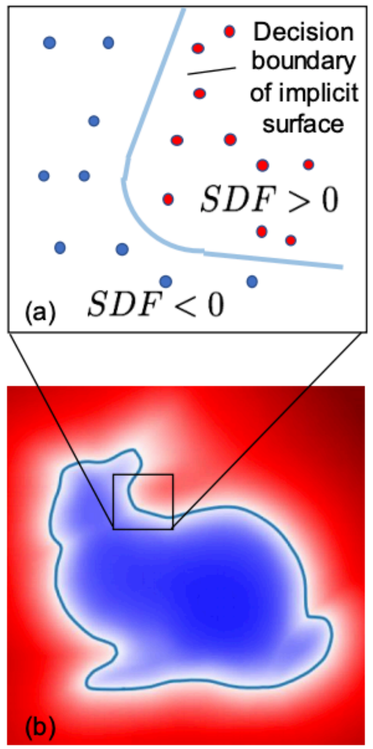




Soft3D [Penner and Zhang 2017],
Stereo Magnification [Zhou et al. 2018],
MPI Extrapolation [Srinivasan et al. 2019],
Local Light Field Fusion [Mildenhall and Srinivasan et al. 2019],

DeepView [Flynn et al. 2019],
Single-View MPI [Tucker and Snavely, 2020],
DeepVoxels [Sitzmann et al. 2019],
Neural Volumes [Lombardi et al. 2019]

Inspiration from 3D computer vision: Using neural networks as a shape representation



DeepSDF, Park et al. 2019

Supervised with 3D:
DeepSDF [Park et al. 2019],
Occupancy Networks [Mescheder et al. 2019],
Local Deep Implicit Functions [Genova et al. 2020],
Local Implicit Grids [Jiang et al. 2020]

Supervised with images:
Scene Representation Networks [Sitzmann et al. 2019],
Differentiable Volumetric Rendering [Niemeyer et al. 2020],
DIST [Liu et al. 2020]

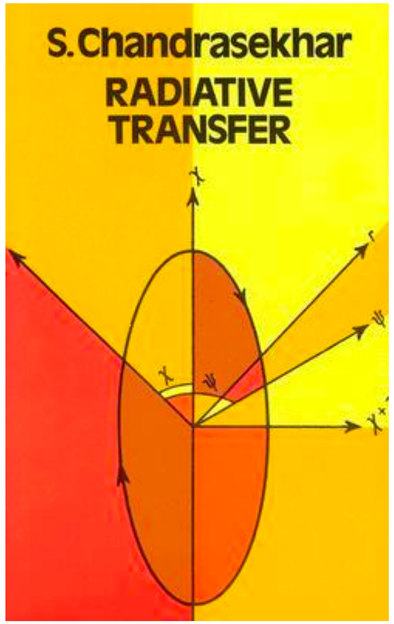
Overview

- Volumetric rendering math
- Neural networks as representations for spatial data
- Neural Radiance Fields (NeRF)
- NeRF improvements and extensions

Overview

- Volumetric rendering math
- Neural networks as representations for spatial data
- Neural Radiance Fields (NeRF)
- NeRF improvements and extensions

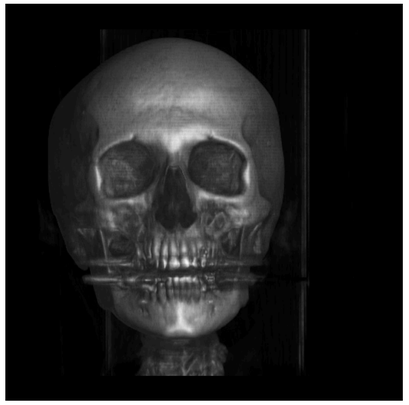
Traditional volumetric rendering



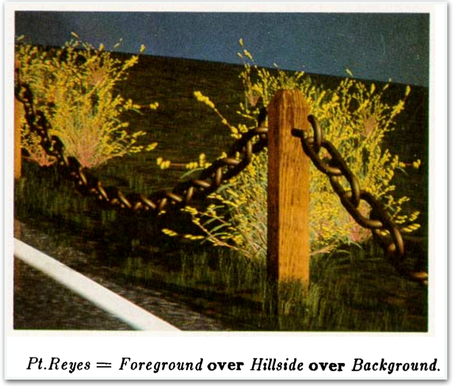
- Theory of volume rendering co-opted from physics in the 1980s: absorption, emission, out-scattering/in-scattering
- Adapted for visualising medical data and linked with alpha compositing
- Modern path tracers use sophisticated Monte Carlo methods to render volumetric effects

Chandrasekhar 1950, *Radiative Transfer*
Kajia 1984, *Ray Tracing Volume Densities*
Levoy 1988, *Display of Surfaces from Volume Data*
Max 1995, *Optical Models for Direct Volume Rendering*
Porter and Duff 1984, *Compositing Digital Images*
Novak et al 2018, *Monte Carlo methods for physically based volume rendering*

Traditional volumetric rendering



Medical data visualisation [Levoy]



Alpha compositing [Porter and Duff]

- 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*

Kajia 1984, *Ray Tracing Volume Densities*

Levoy 1988, *Display of Surfaces from Volume Data*

Max 1995, *Optical Models for Direct Volume Rendering*

Porter and Duff 1984, *Compositing Digital Images*

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

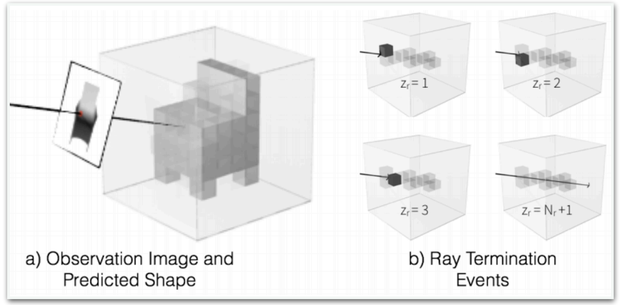
Traditional volumetric rendering

Physically-based Monte Carlo rendering [Novak et al]

- 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*
Kajia 1984, *Ray Tracing Volume Densities*
Levoy 1988, *Display of Surfaces from Volume Data*
Max 1995, *Optical Models for Direct Volume Rendering*
Porter and Duff 1984, *Compositing Digital Images*
Novak et al 2018, *Monte Carlo methods for physically based volume rendering*

Volumetric rendering and machine learning

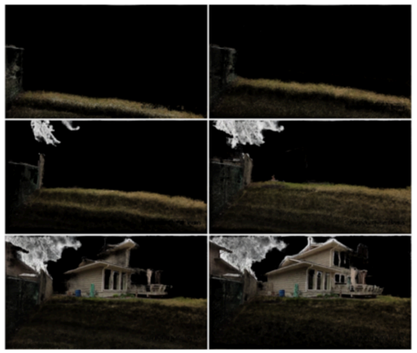


"Probabilistic" voxel grid rendering [Tulsiani et al]

- Various volume-rendering-esque methods devised for 3D shape reconstruction methods
- Scaled up to higher resolution volumes to achieve excellent view synthesis results

Tulsiani et al 2017, *Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency*
Henzler et al 2019, *Escaping Plato's Cave: 3D Shape From Adversarial Rendering*
Zhou et al 2018, *Stereo Magnification: Learning View Synthesis using Multiplane Images*
Lombardi et al 2019, *Neural Volumes: Learning Dynamic Renderable Volumes from Images*

Volumetric rendering and machine learning



Slices from a volumetric scene representation [Zhou et al]

- Various volume-rendering-esque methods devised for 3D shape reconstruction methods

View synthesis from a dynamic voxel grid [Lombardi et al]

- Scaled up to higher resolution voxel grids, ML methods can achieve excellent view synthesis results

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

Henzler et al 2019, *Escaping Plato's Cave: 3D Shape From Adversarial Rendering*

Zhou et al 2018, *Stereo Magnification: Learning View Synthesis using Multiplane Images*

Lombardi et al 2019, *Neural Volumes: Learning Dynamic Renderable Volumes from Images*

Volumetric formulation for NeRF

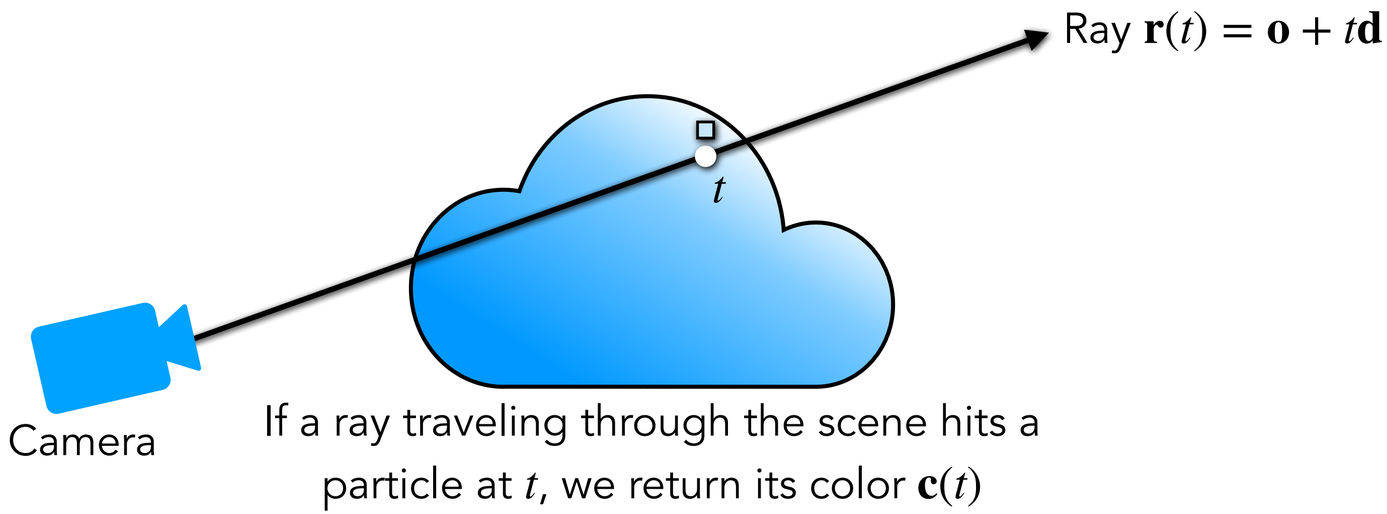
Max and Chen 2010, *Local and Global Illumination in the Volume Rendering Integral*

Volumetric formulation for NeRF

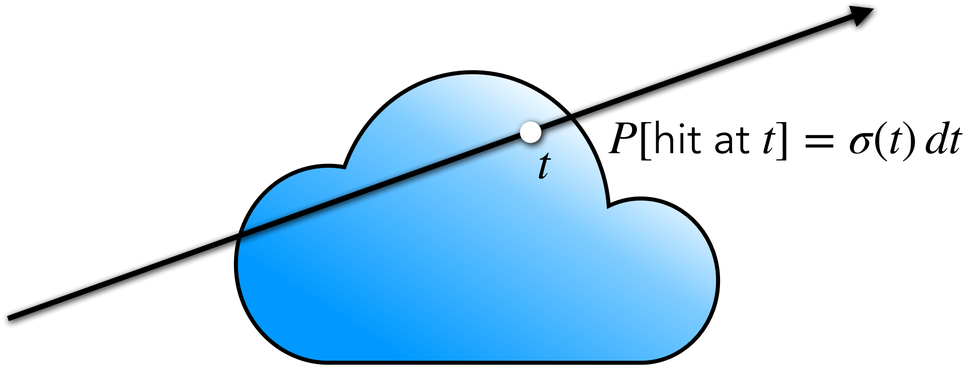
Scene is a cloud of tiny colored particles

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

Volumetric formulation for NeRF



Volumetric formulation for NeRF

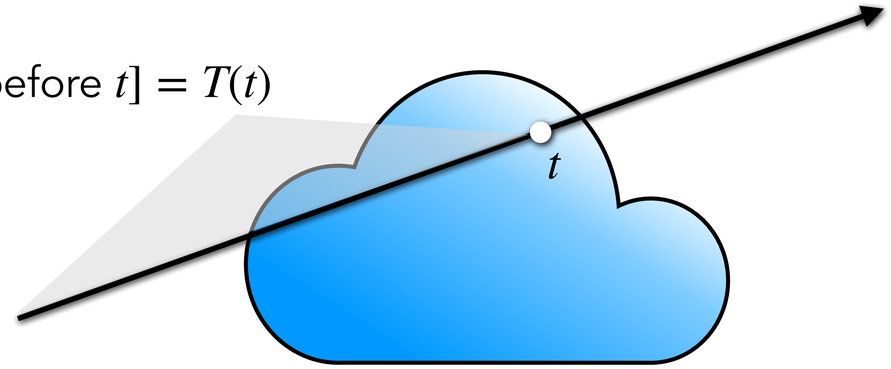


This notion is *probabilistic:* chance that ray stops in a small interval around t is $\sigma(t)\,dt$.

σ is known as the "volume density"

Volumetric formulation for NeRF

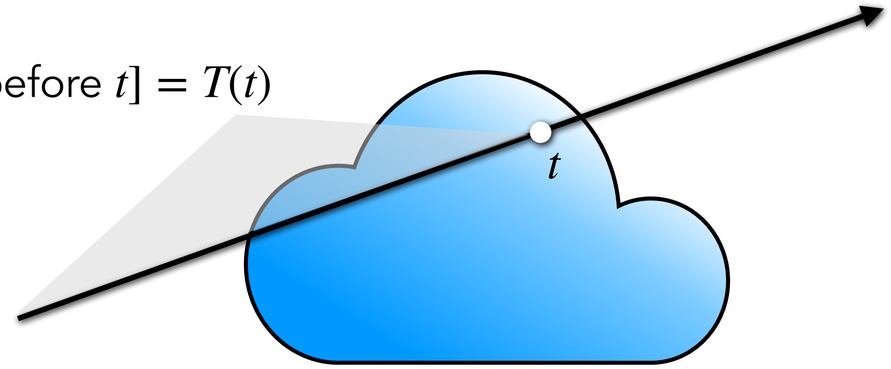
$P[\text{no hits before } t] = T(t)$



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

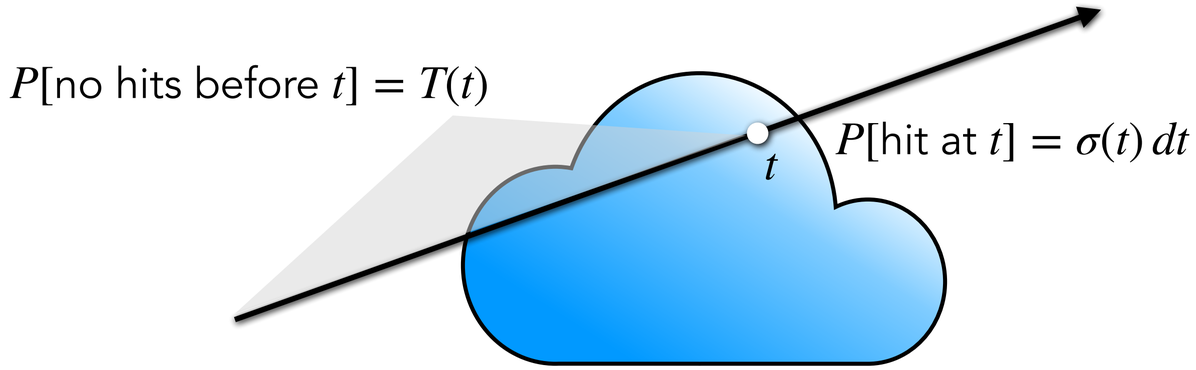
Volumetric formulation for NeRF

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



To determine if t is the *first* hit, need to know $T(t)$:
probability that the ray didn't hit any particles earlier.
$T(t)$ is called "transmittance"
We assume σ is known and want to use it to calculate T

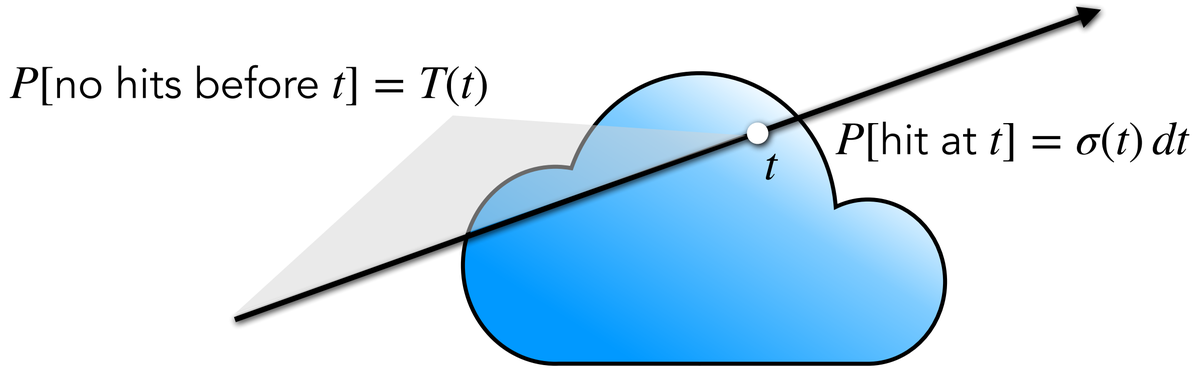
Volumetric formulation for NeRF



σ and T are related by the probability fact that

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

Volumetric formulation for NeRF



σ and T are related by the probability fact that

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

Volumetric formulation for NeRF

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

Volumetric formulation for NeRF

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

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

Volumetric formulation for NeRF

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

Split up differential ⇒ $T(t) + T'(t)dt = T(t) - T(t)\sigma(t)dt$

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

Volumetric formulation for NeRF

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

Split up differential ⇒ $T(t) + T'(t)dt = T(t) - T(t)\sigma(t)dt$

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

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

Volumetric formulation for NeRF

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

Volumetric formulation for NeRF

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

Food for thought #1: for a constant density medium and $t_0 = 0$, we have

$$1 - T(t) = 1 - \exp(-\sigma t),$$

which is the exponential distribution CDF

Volumetric formulation for NeRF

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

Food for thought #2: From our derivation,

$$\nabla \log T(t) = -\sigma(t),$$

so $-\sigma$ acts as the score function of transmittance.
Interesting: T depends on the current camera ray, but σ does not.

Volumetric formulation for NeRF

Finally, what we want to know: the probability that a ray first hits a particle at t is

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

Volumetric formulation for NeRF

Finally, what we want to know: the probability that a ray first hits a particle at t is

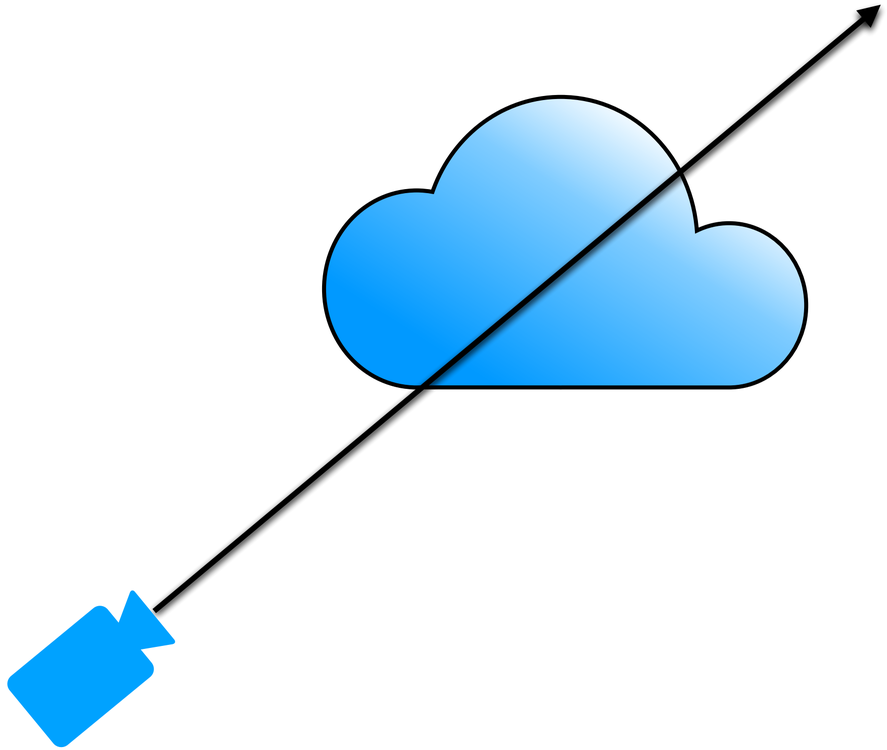
$$T(t)\sigma(t)\,dt = \exp\left(-\int_{t_0}^{t}\sigma(s)\,ds\right)\sigma(t)\,dt$$

So the expected color returned by the ray will be

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

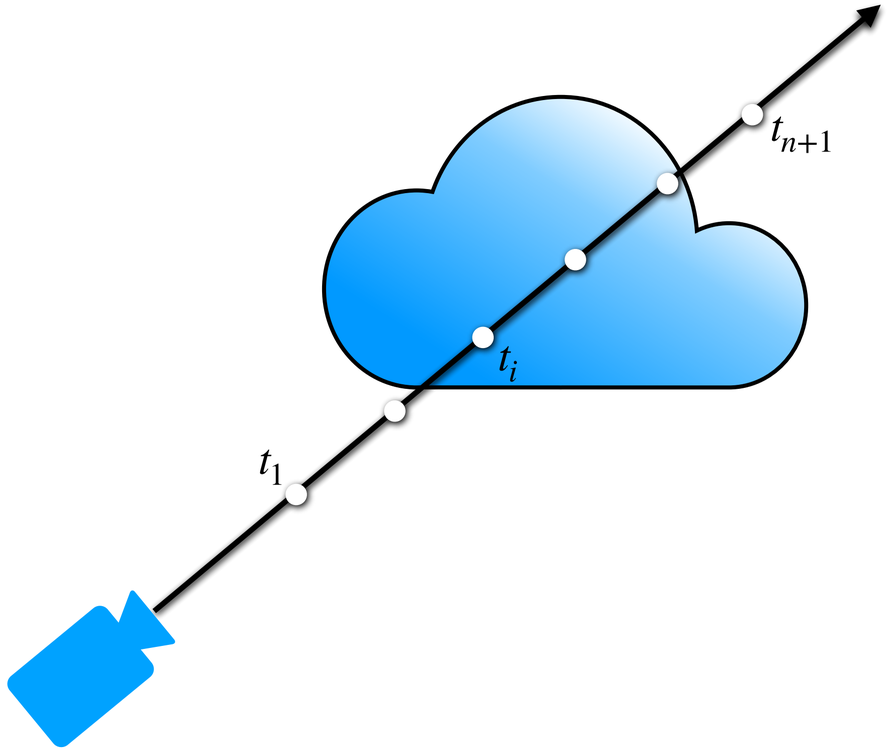
Note the nested integral!

Approximating the nested integral



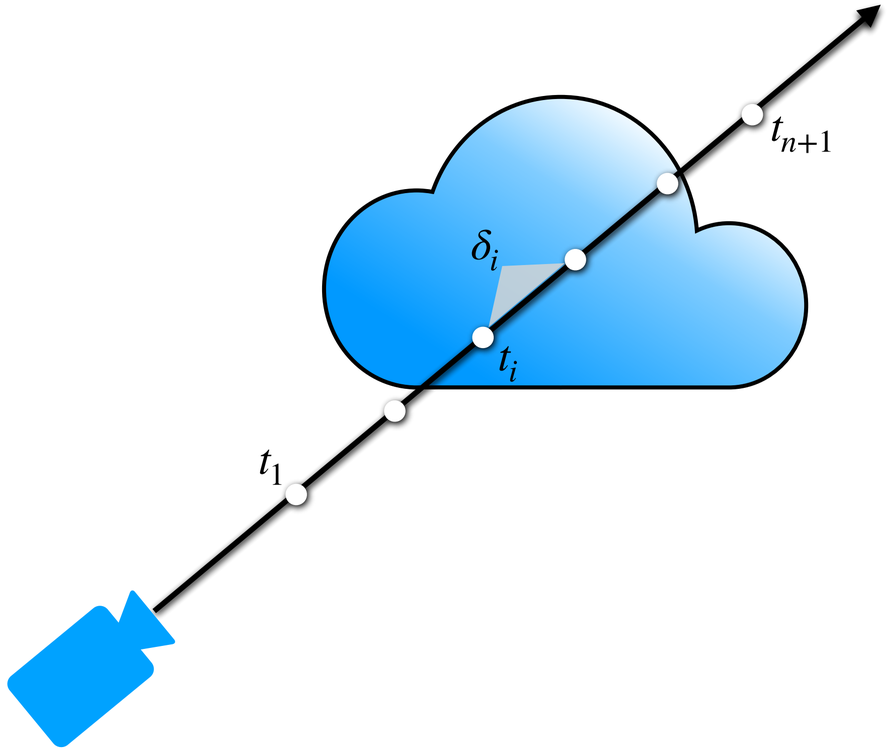
We use quadrature to approximate the nested integral,

Approximating the nested integral



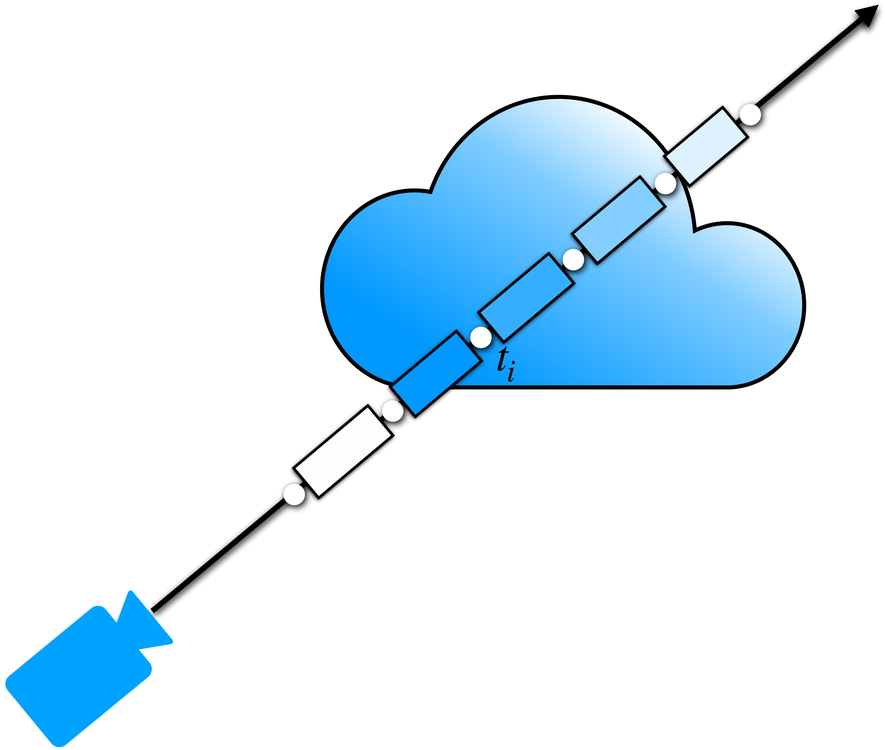
We use quadrature to approximate the nested integral,
splitting the ray up into n segments with endpoints $\{t_1, t_2, \ldots, t_{n+1}\}$

Approximating the nested integral



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

Approximating the nested integral



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

Approximating the nested integral

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

This allows us to break the outer integral

Approximating the nested integral

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

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

Approximating the nested integral

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

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

Approximating the nested integral

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

Catch: piecewise constant density and color **do not** imply constant transmittance!

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

Approximating the nested integral

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

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

Approximating the nested integral

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

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

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

"How much is blocked by all previous segments?"

Approximating the nested integral

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

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

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

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

Approximating the nested integral

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

Approximating the nested integral

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

Substitute

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

Approximating the nested integral

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

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

Integrate

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

Approximating the nested integral

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

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

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

Cancel σ_i

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

Connection to alpha compositing

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

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

Connection to alpha compositing

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

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

$$\text{color} = \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i = \sum_{i=1}^{n} T_i \mathbf{c}_i \left(1 - \exp(-\sigma_i \delta_i)\right)$$

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

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

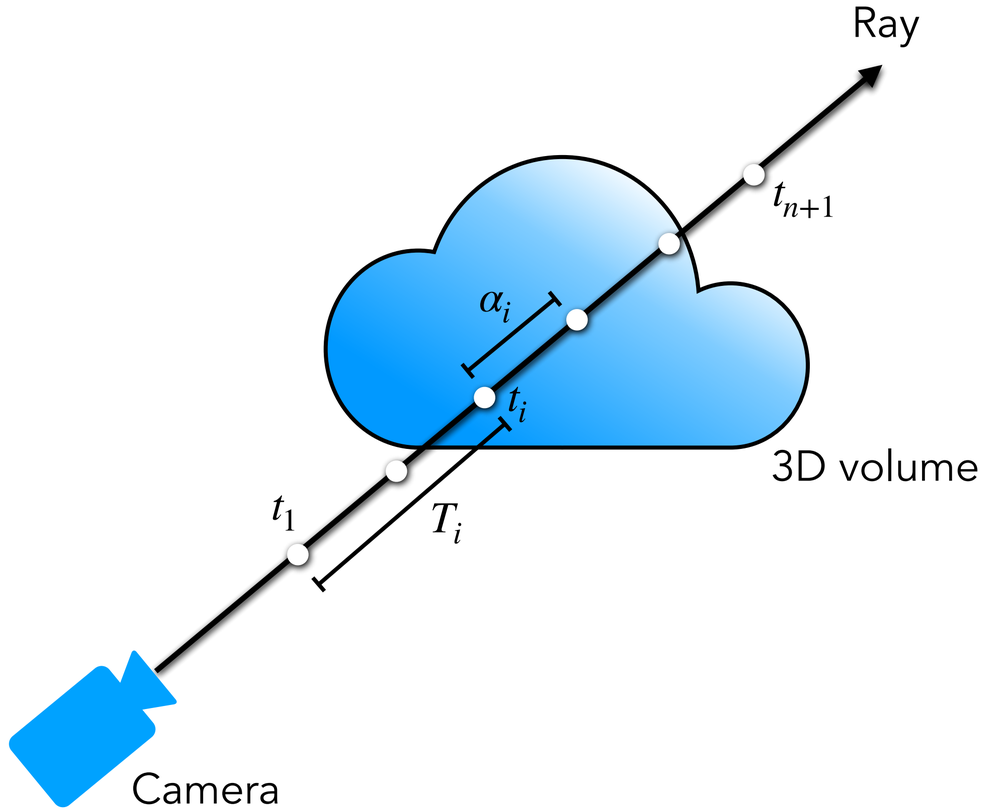
weights

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*:

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



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

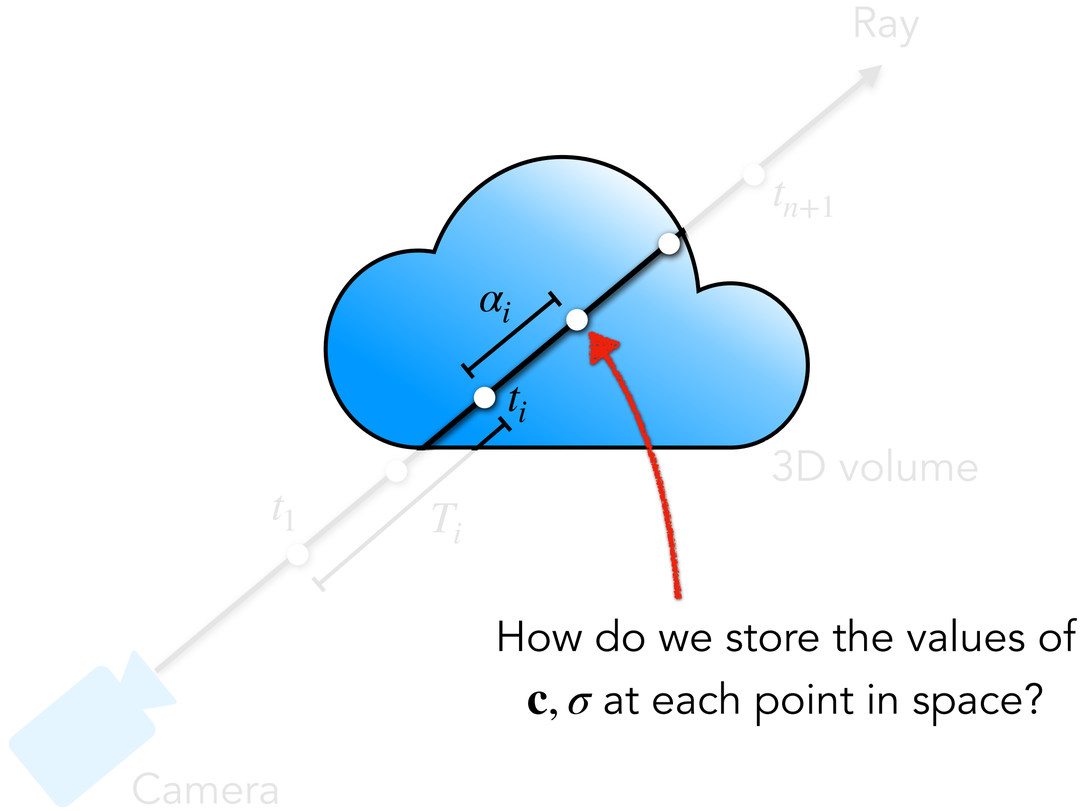
weights

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:

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

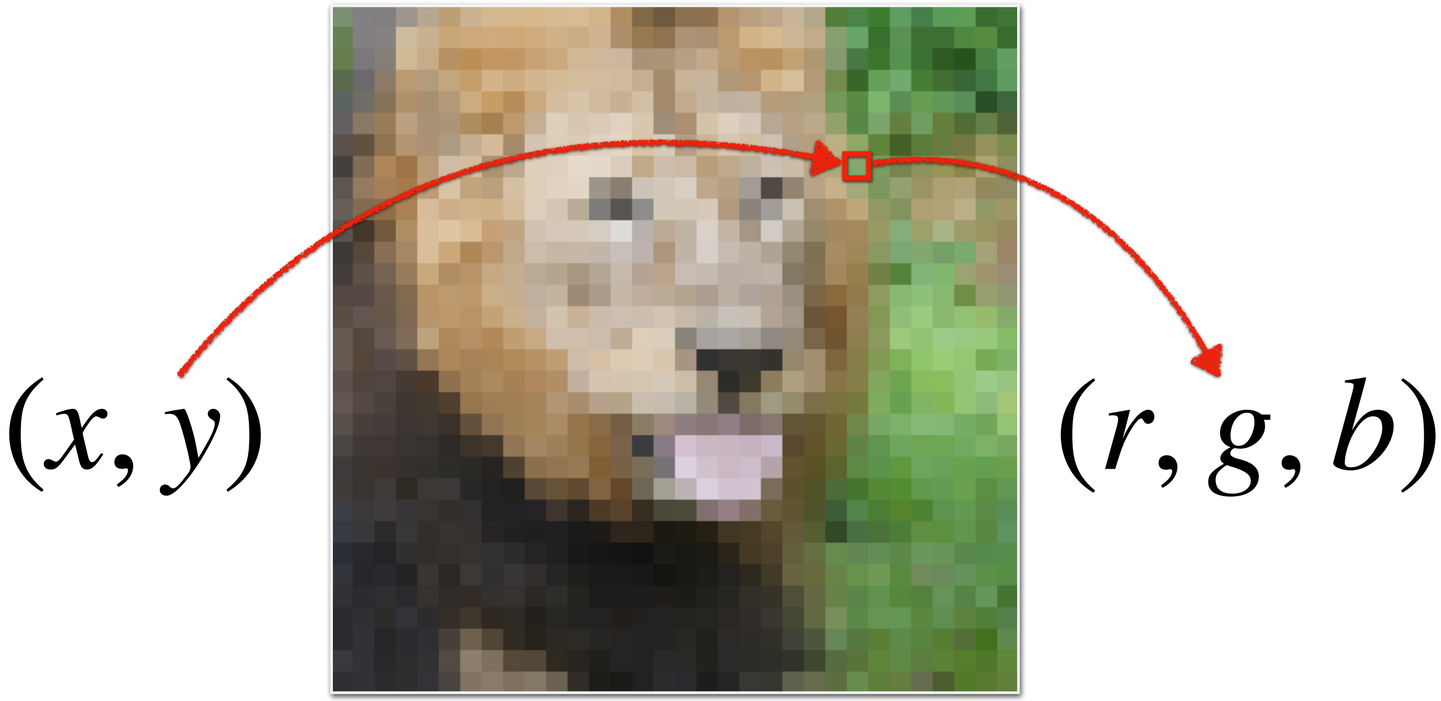


How do we store the values of $\mathbf{c}, \sigma$ at each point in space?

Overview

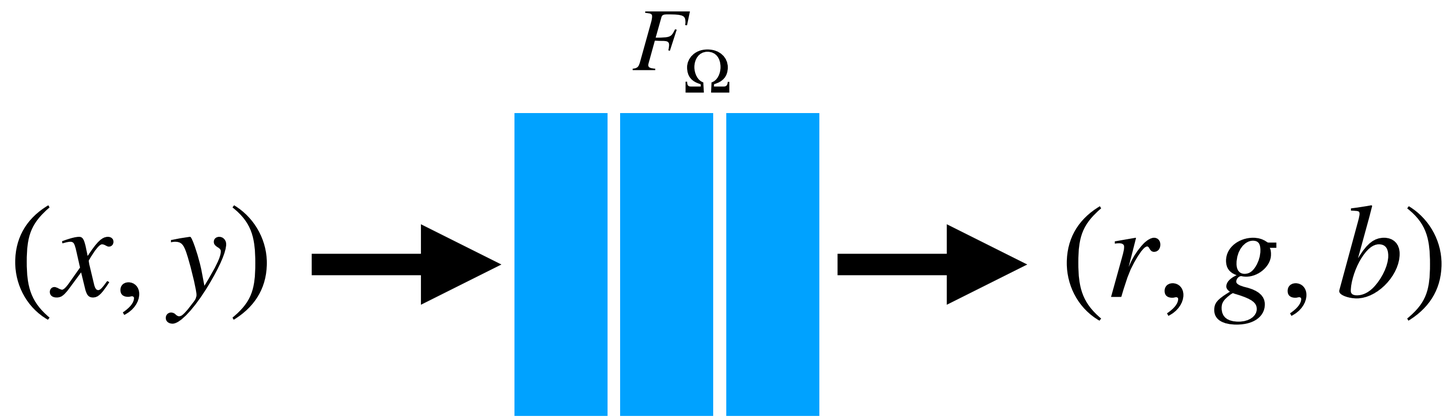
- Volumetric rendering math
- Neural networks as representations for spatial data
- Neural Radiance Fields (NeRF)
- NeRF improvements and extensions

Toy problem: storing 2D image data



Usually we store an image as a
2D grid of RGB color values

Toy problem: storing 2D image data



What if we train a simple fully-connected network (MLP) to do this instead?

Naive approach fails!

Ground truth image

Neural network output

Problem:

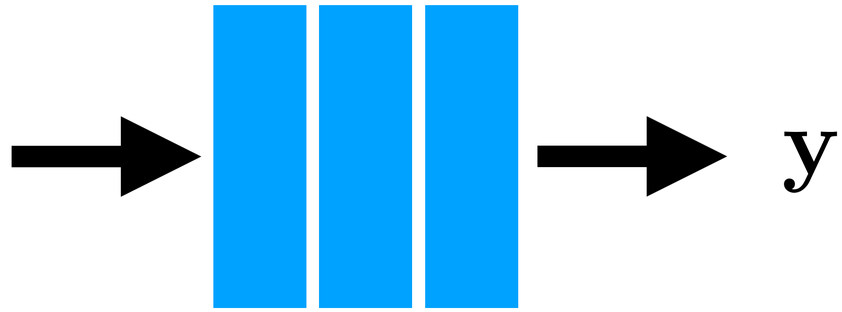
"Standard" coordinate-based MLPs cannot represent high frequency functions

Solution:

Pass input coordinates through a high frequency mapping first

Example mapping: “positional encoding”

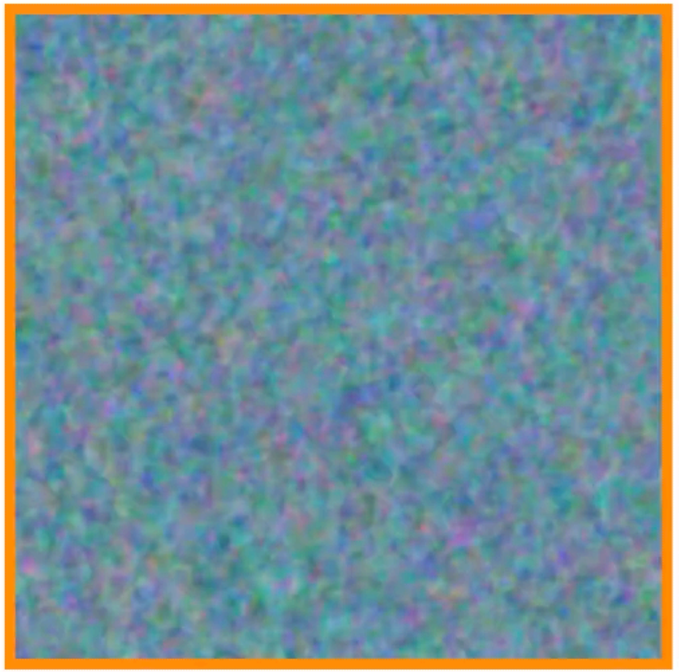
$$\begin{pmatrix} \sin(\mathbf{v}), \cos(\mathbf{v}) \\ \sin(2\mathbf{v}), \cos(2\mathbf{v}) \\ \sin(4\mathbf{v}), \cos(4\mathbf{v}) \\ \dots \\ \sin(2^{L-1}\mathbf{v}), \cos(2^{L-1}\mathbf{v}) \end{pmatrix}$$



Problem solved… but why?

Ground truth image

Neural network output **without** high frequency mapping



Neural network output **with** high frequency mapping

Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains

NeurIPS 2020

Training networks ≈ kernel regression

- Recent ML theory work shows that training neural network with gradient descent becomes the same as performing kernel regression as the width of each layer goes to infinity

- Can examine corresponding kernel function (the *neural tangent kernel*) to see why adding Fourier feature mapping allows MLPs to represent high frequency functions

Jacot et al., *Neural Tangent Kernel: Convergence and generalization in neural networks*, NeurIPS 2018

Kernel regression

- Method for fitting a continuous function to a set of data points $\{(x_i, y_i)\}$
- High level: add up a set of blobs (kernel functions), one centered at each input point, each with its own weight
- Weights are optimal in a least-squares sense: $\min_w \sum_i \|y_i - \hat{f}_w(x_i)\|^2$

$$\hat{f}_w(x) = \sum_{i=1}^{n} w_i \, k(x - x_i)$$

Blob centered at training input point x_i

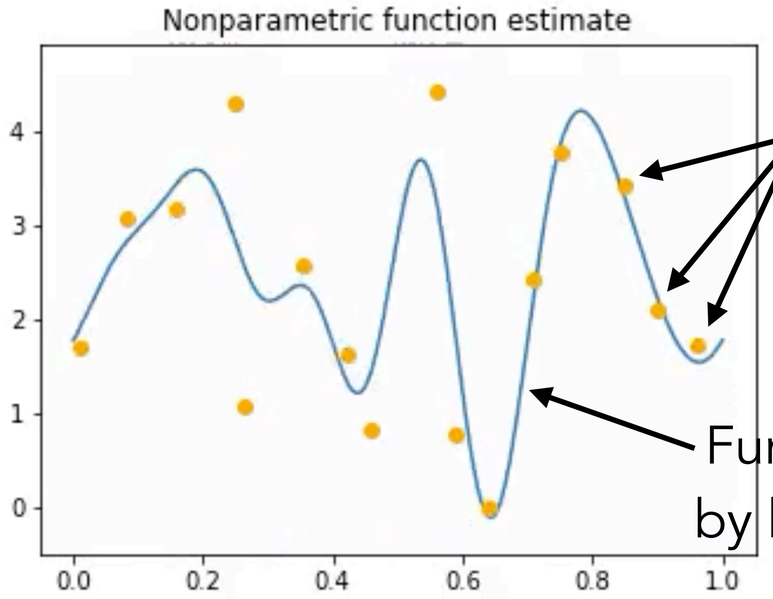
Estimated function

Weight corresponding to blob centered at x_i

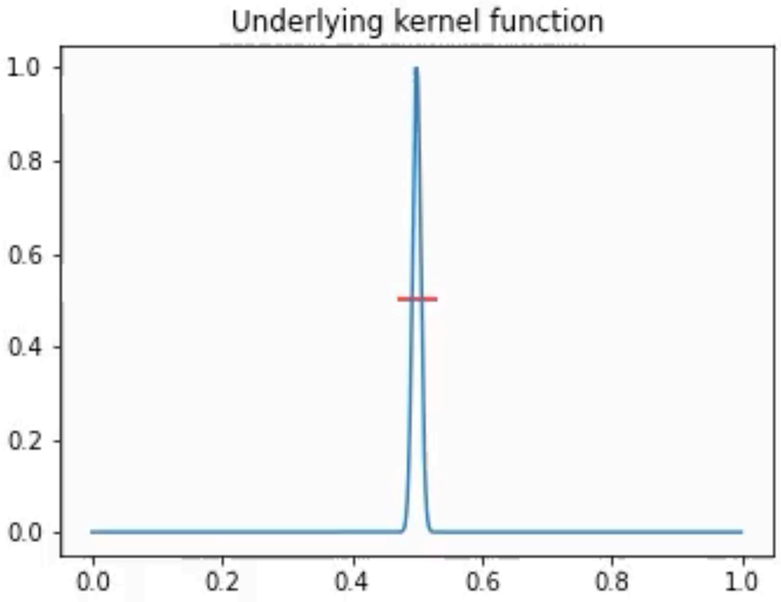
“Width” of kernel function is critical

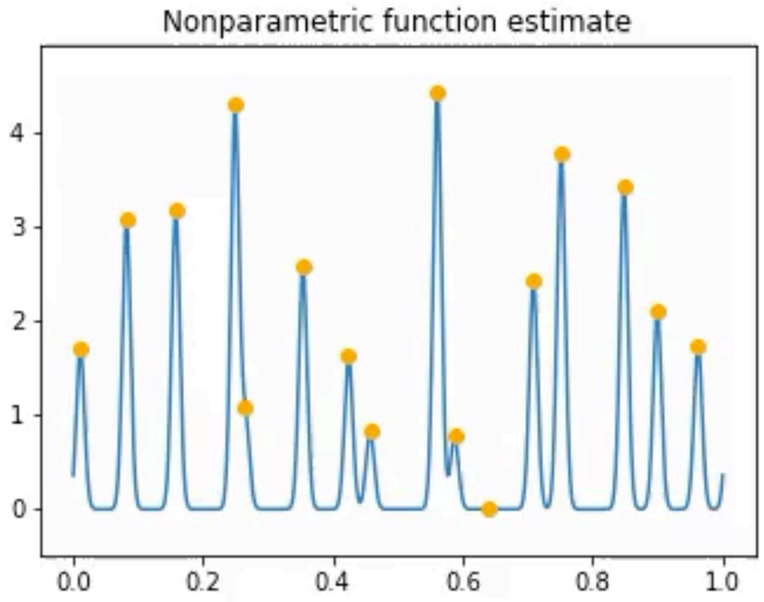
- If the kernel function is too wide, reconstruction is too smooth. If it’s too skinny, reconstruction does not interpolate correctly.
- Similar to picking the right reconstruction filter bandwidth in signal processing to avoid either blurring or aliasing.

“Width” of kernel function is critical



“Width” of kernel function is critical





Training networks ≈ kernel regression

- Recent ML theory work shows that training neural network with gradient descent becomes the same as performing kernel regression as the width of each layer goes to infinity

- Using a Fourier feature mapping changes the corresponding kernel function (the *neural tangent kernel*), allowing MLPs to represent higher frequency functions

Jacot et al., *Neural Tangent Kernel: Convergence and generalization in neural networks*, NeurIPS 2018

Fourier feature mapping

- Mapping procedure = sample a random tall skinny matrix **B** and apply it to the input coordinate vectors **x**, then apply *sin* and *cos*:

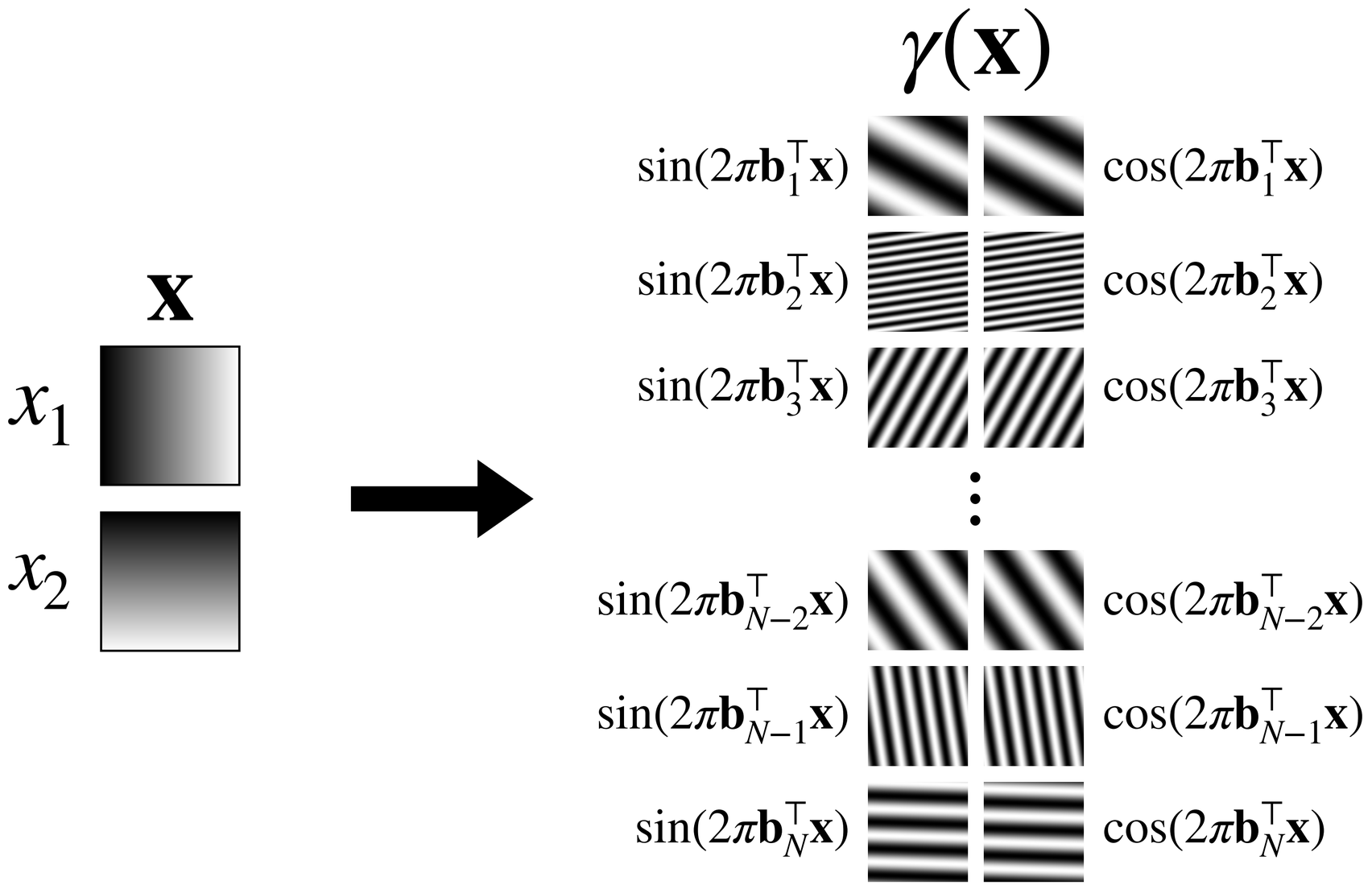
$$\gamma(\mathbf{x}) = (\sin(2\pi\mathbf{B}\mathbf{x}), \cos(2\pi\mathbf{B}\mathbf{x}))$$

- Same as the *Random Fourier Feature* mapping proposed by [Rahimi and Recht 2008]
- "Positional encoding" is a special case with deterministic **B**:

$$\mathbf{B} = [2^0\mathbf{I} \quad 2^1\mathbf{I} \quad \cdots \quad 2^{L-1}\mathbf{I}]^\top$$

Rahimi and Recht, *Random features for large-scale kernel machines*, NeurIPS 2008
Vaswani et al., *Attention is all you need*, NeurIPS 2017

Fourier feature mapping: simple 2D example



“Neural tangent kernel” is a scalar function of dot product

- Can express the kernel corresponding to the network as a scalar function of the inner product of two input vectors:

$$\mathrm{NTK}(\mathbf{x}, \mathbf{y}) = h(\mathbf{x}^\top \mathbf{y})$$

- Dot product of Fourier feature mapping is simple:

$$\begin{aligned}\gamma(\mathbf{x})^\top \gamma(\mathbf{y}) &= \sin(2\pi\mathbf{B}\mathbf{x})^\top \sin(2\pi\mathbf{B}\mathbf{y}) + \cos(2\pi\mathbf{B}\mathbf{x})^\top \cos(2\pi\mathbf{B}\mathbf{y}) \\ &= \cos(2\pi\mathbf{B}(\mathbf{x} - \mathbf{y}))\end{aligned}$$

- Hence adding Fourier features changes the effective kernel to:

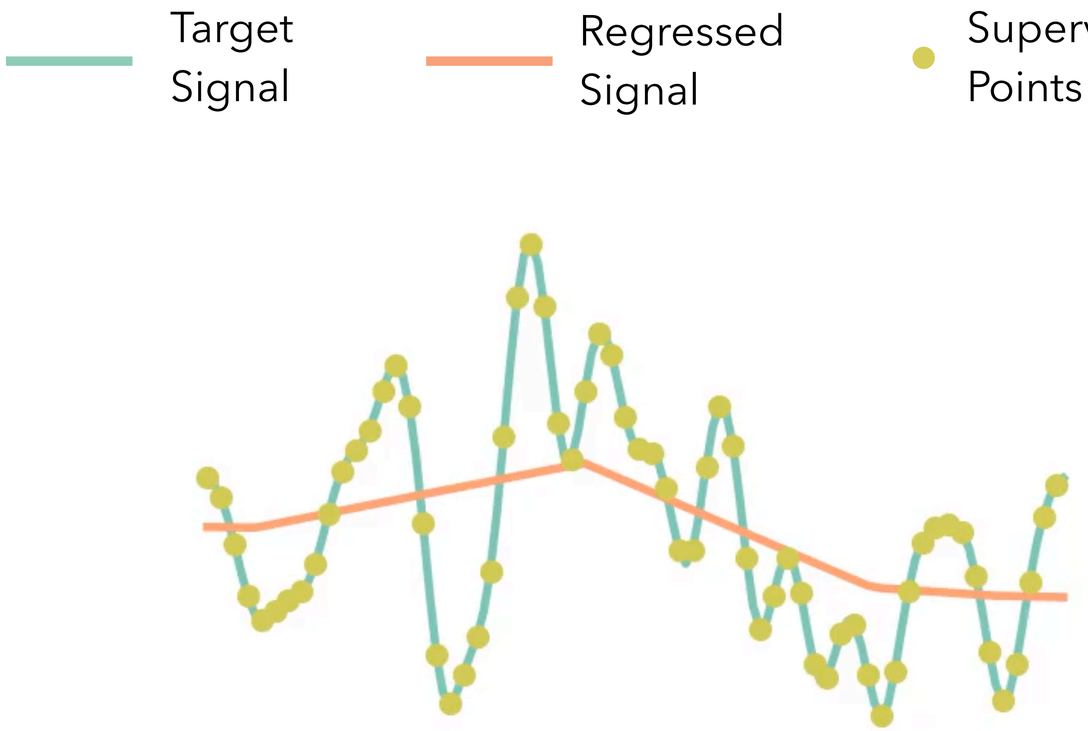
$$\mathrm{NTK}(\gamma(\mathbf{x}), \gamma(\mathbf{y})) = h(\cos(2\pi\mathbf{B}(\mathbf{x} - \mathbf{y})))$$

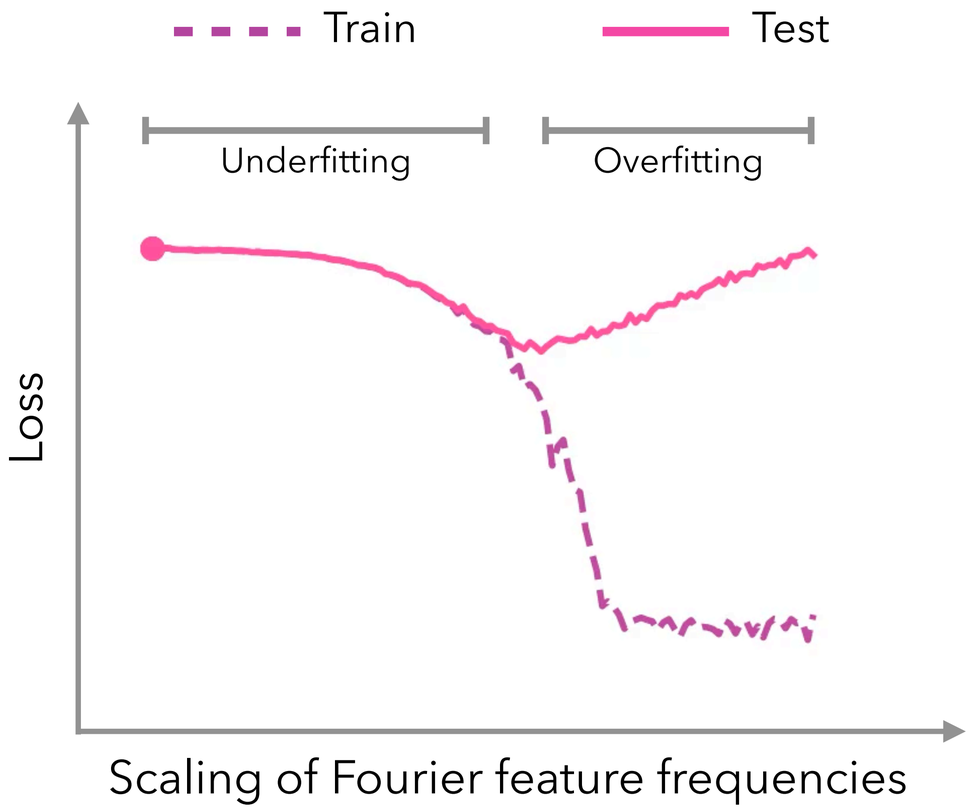
Fourier feature mapping lets us control width of *neural tangent kernel*

If we simply scale **B**, we can manipulate the width of the kernel!

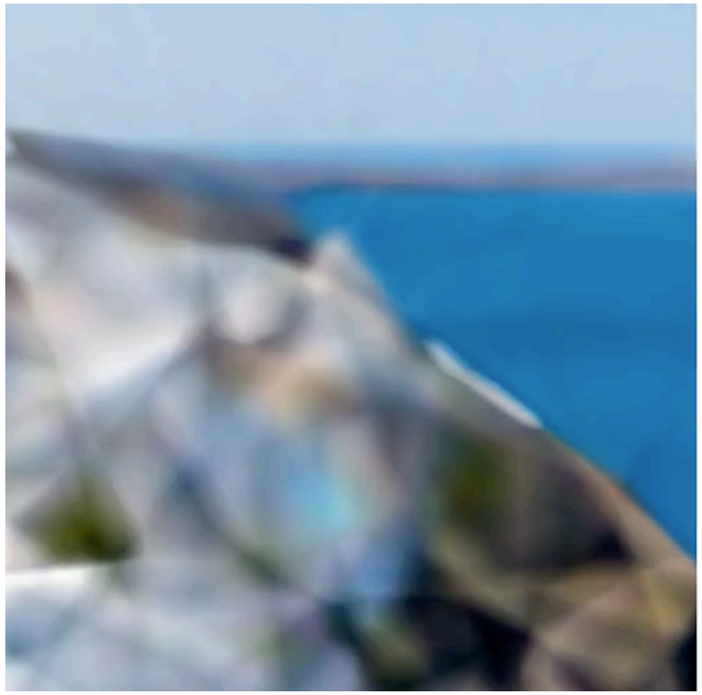
$$\mathrm{NTK}(\gamma(\mathbf{x}), \gamma(\mathbf{y})) = h(\cos(2\pi \mathbf{B}(\mathbf{x} - \mathbf{y})))$$

Changing feature scale σ traverses an underfitting-overfitting curve

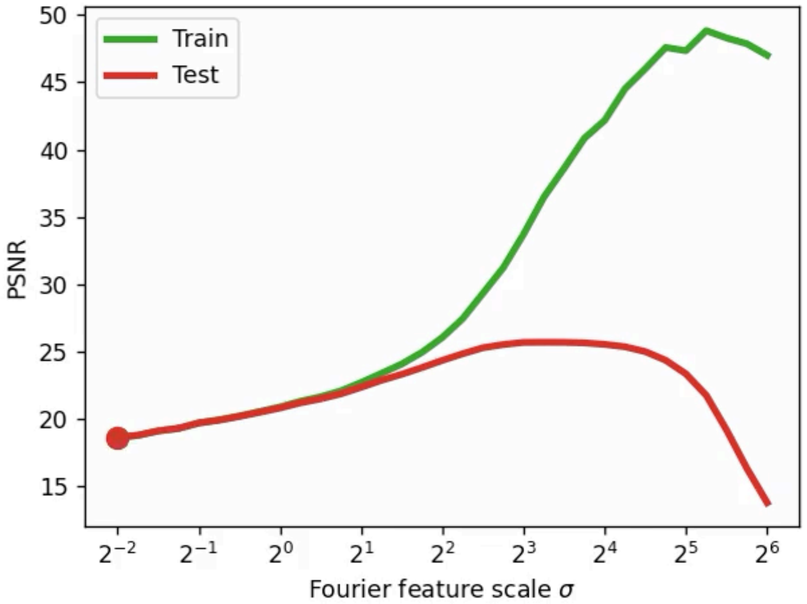




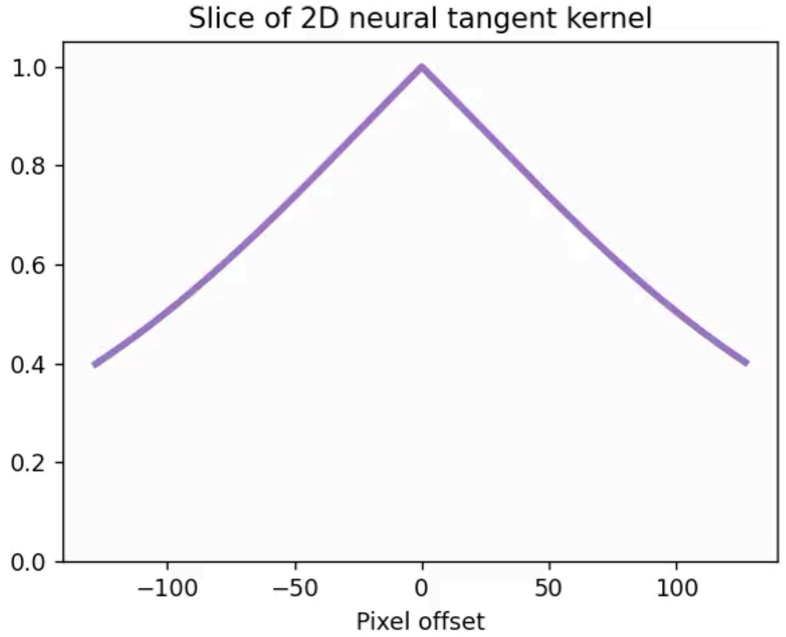
Changing feature scale σ traverses an underfitting-overfitting curve



Network output



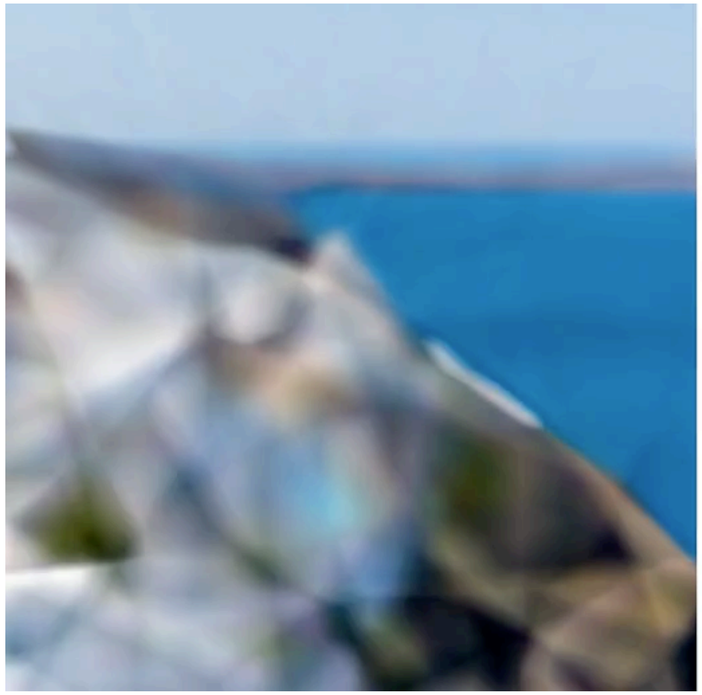
Performance vs. scale value



Network kernel shape

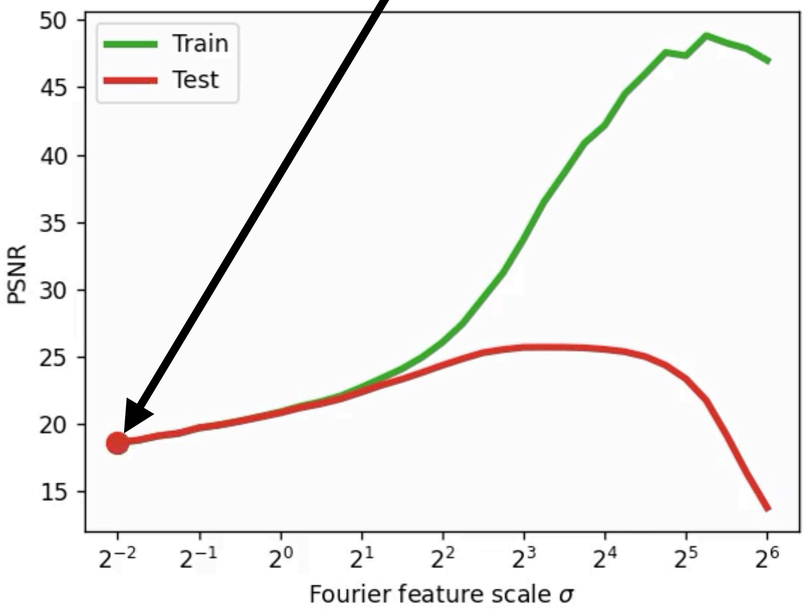
Changing feature scale σ traverses an underfitting-overfitting curve

Learned output too smooth



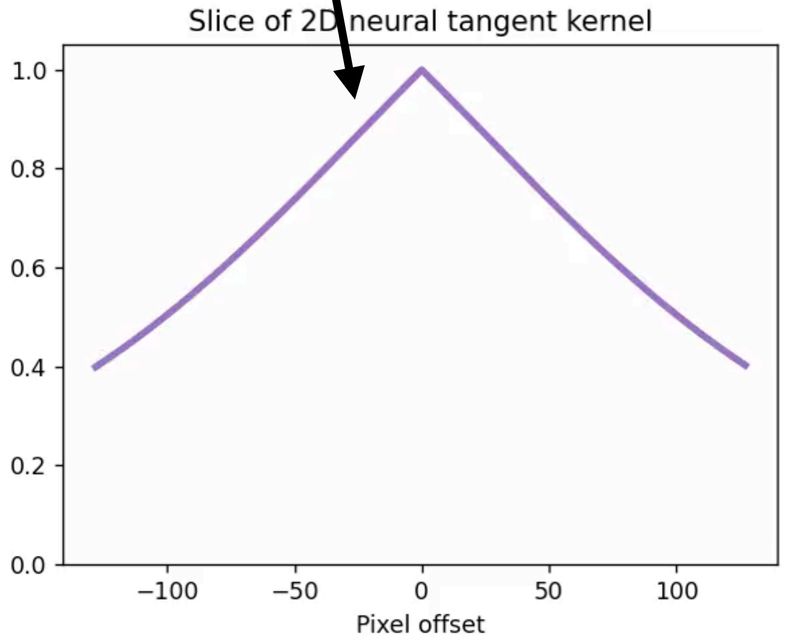
Network output

Underfitting — poor performance on both training points and test points (interpolation behavior)



Performance vs. scale value

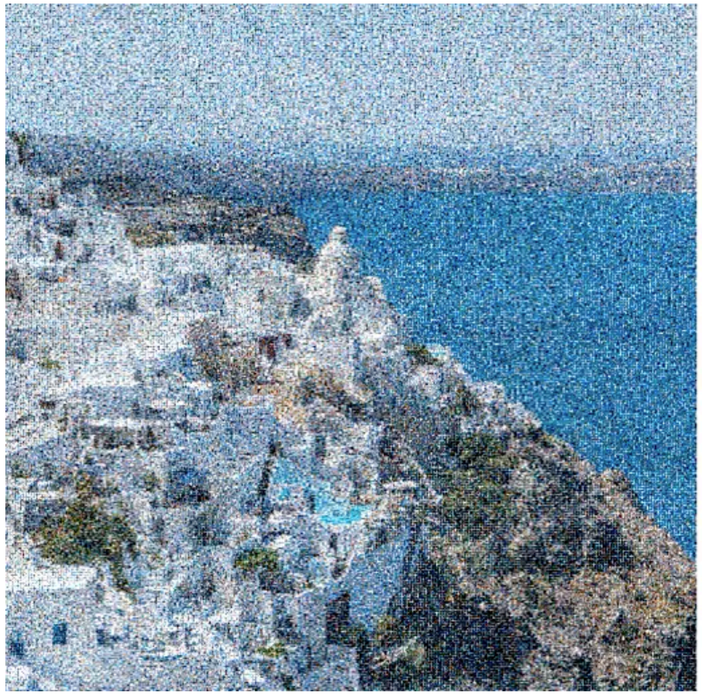
Kernel too wide



Network kernel shape

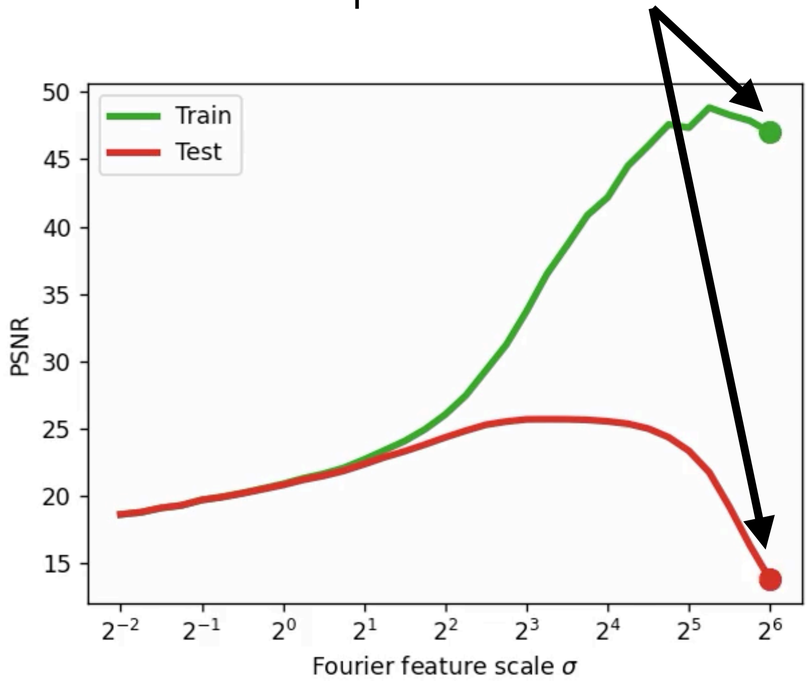
Changing feature scale σ traverses an underfitting-overfitting curve

Output exhibits aliasing



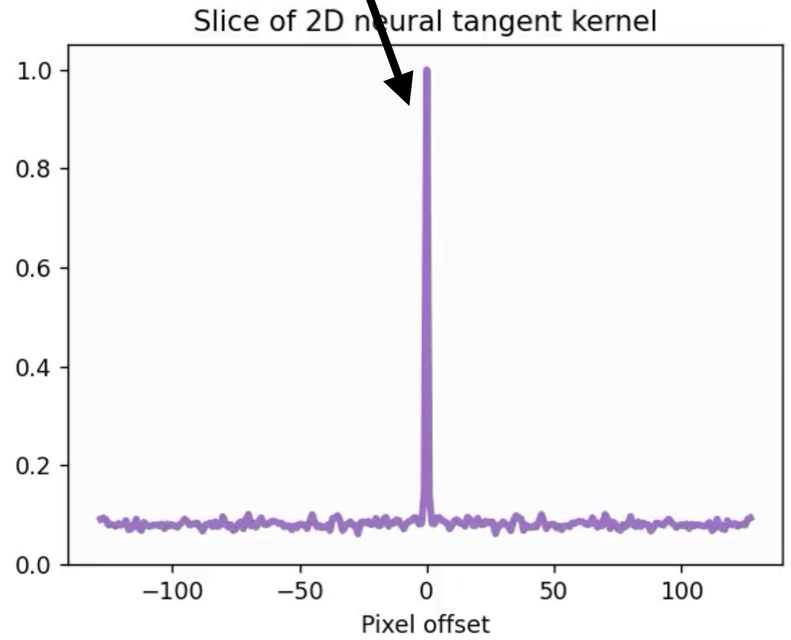
Network output

Overfitting — great performance on training points but very bad interpolation behavior



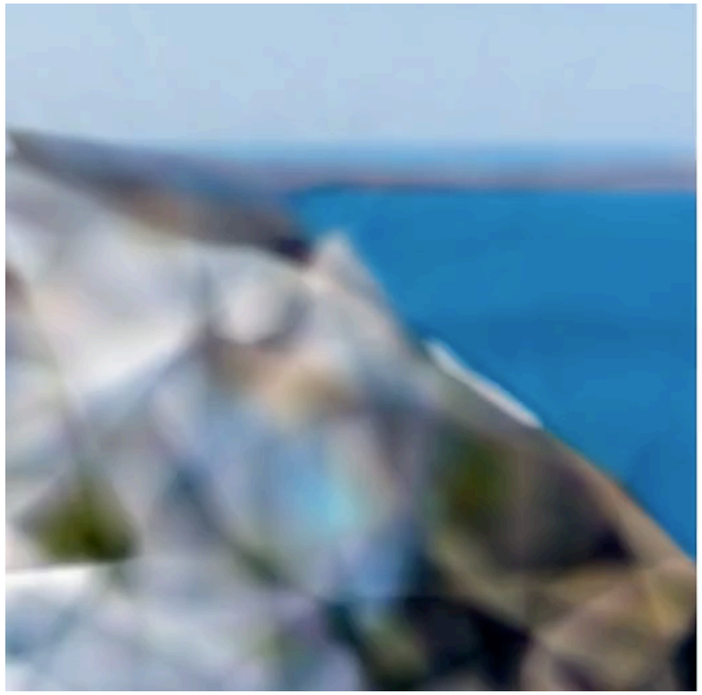
Performance vs. scale value

Kernel too narrow

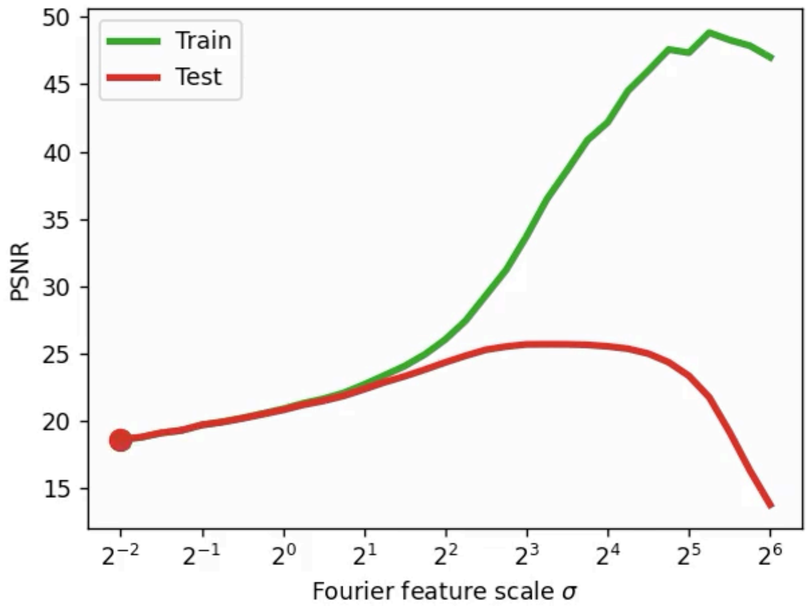


Network kernel shape

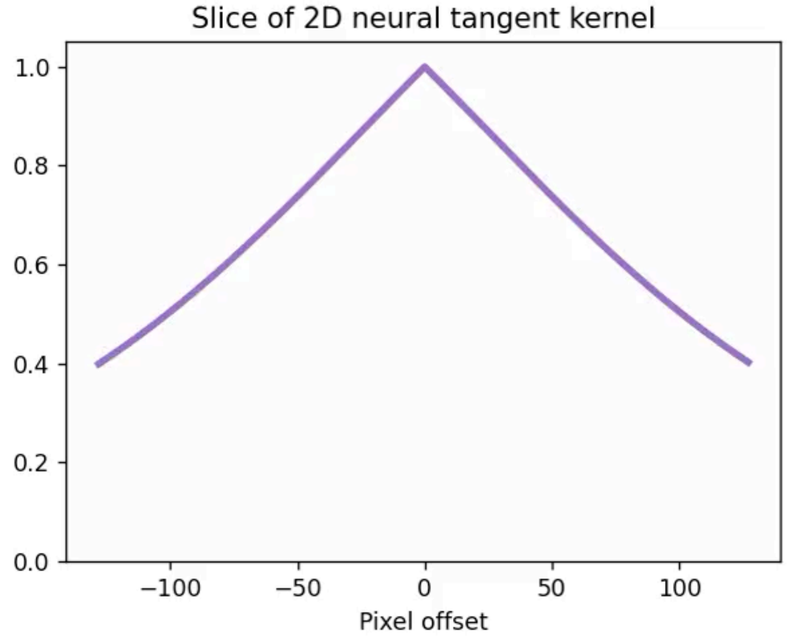
Optimal scale σ lies between the extremes



Network output



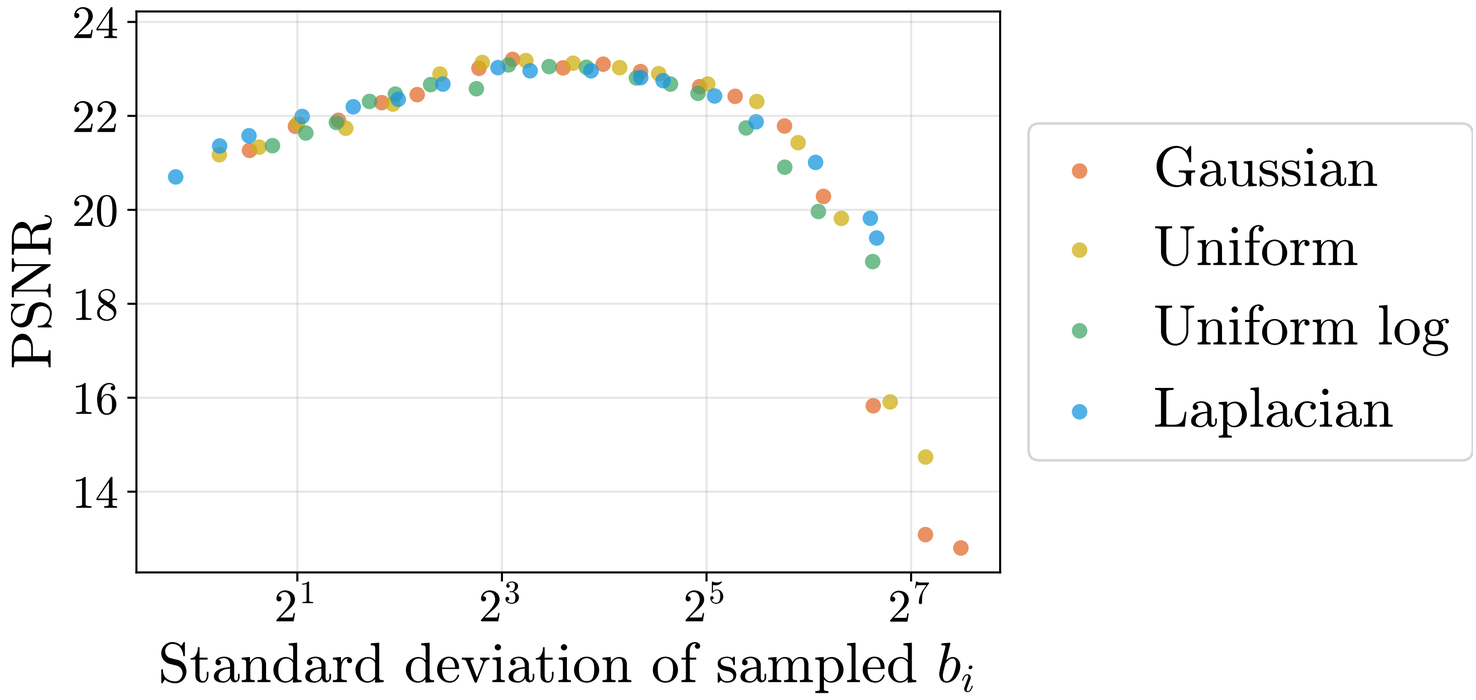
Performance vs. scale value



Network kernel shape

Which random distribution for **B**?

- We empirically observe that the standard deviation of entries in **B** matters much more than higher order moments

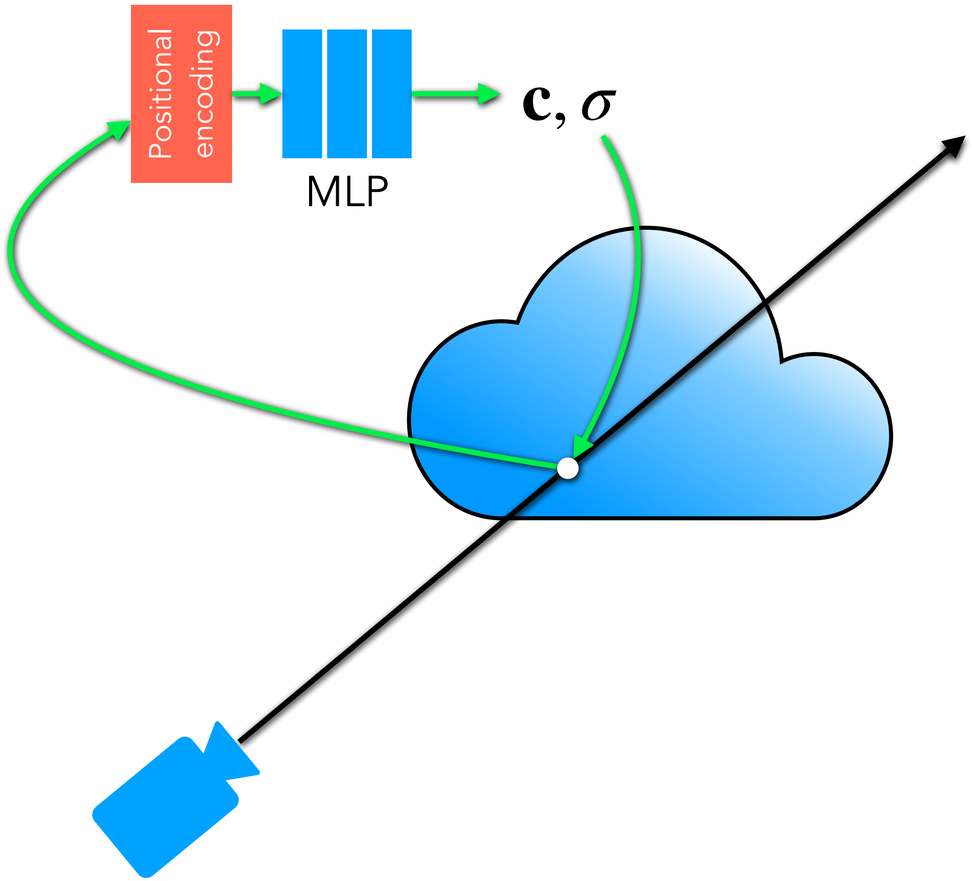


Overview

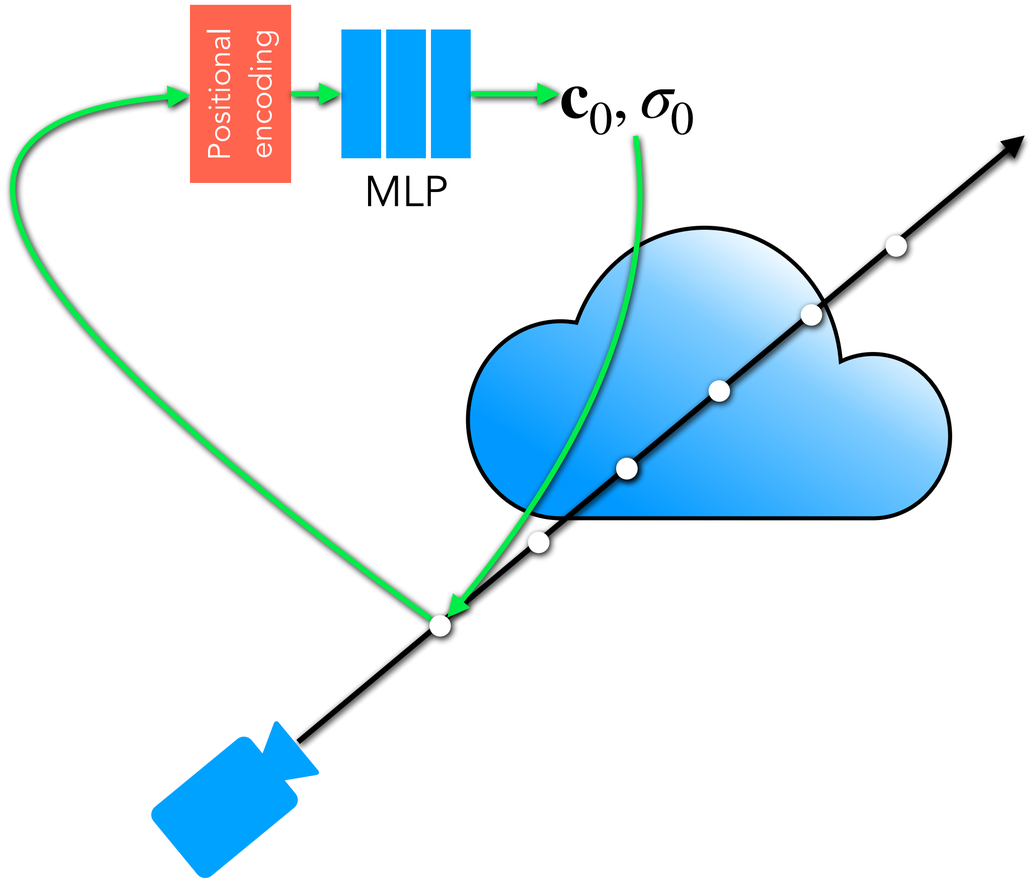
- Volumetric rendering math
- Neural networks as representations for spatial data
- Neural Radiance Fields (NeRF)
- NeRF improvements and extensions

NeRF = volume rendering + coordinate-based network

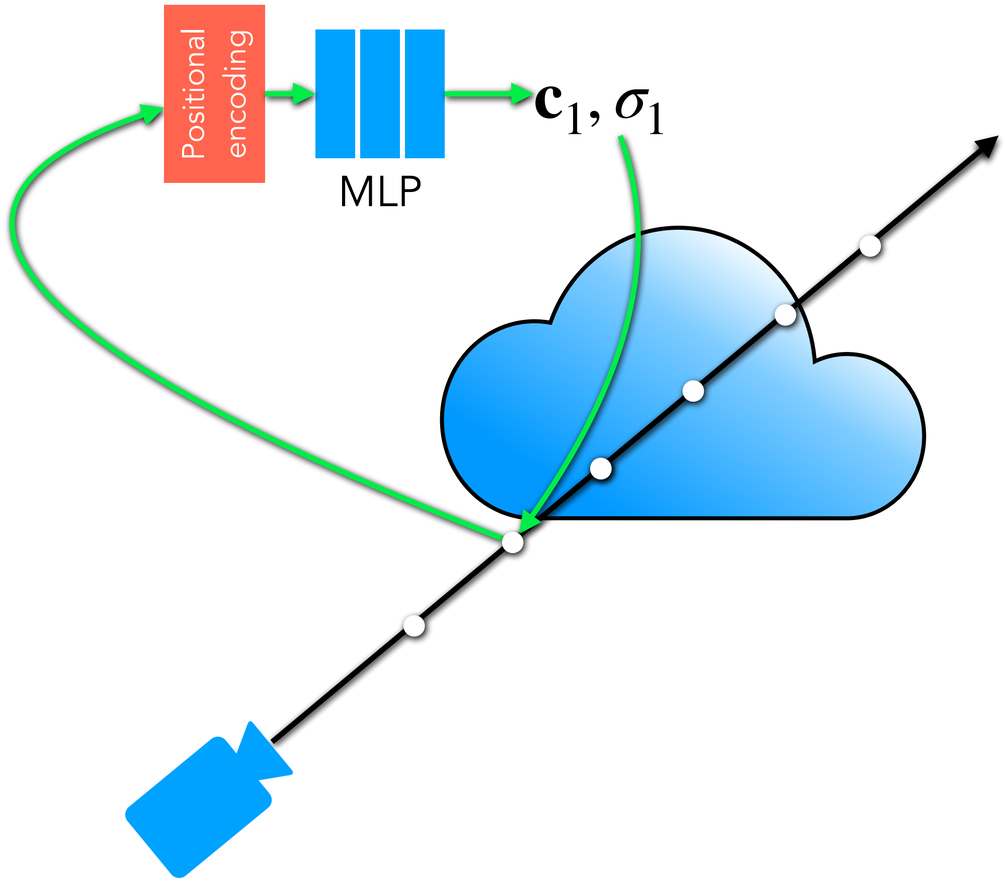
“How do we store the values of $\mathbf{c}, \sigma$ at each point in space?”



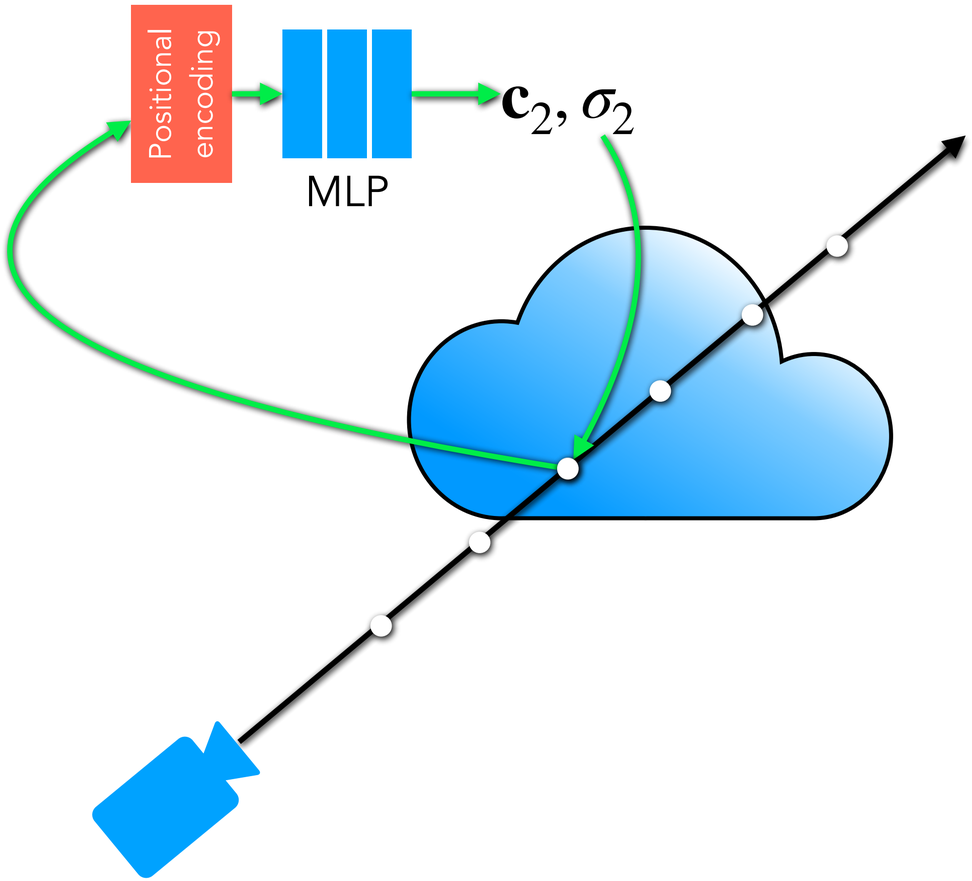
“How do we store the values of $\mathbf{c}, \sigma$ at each point in space?”



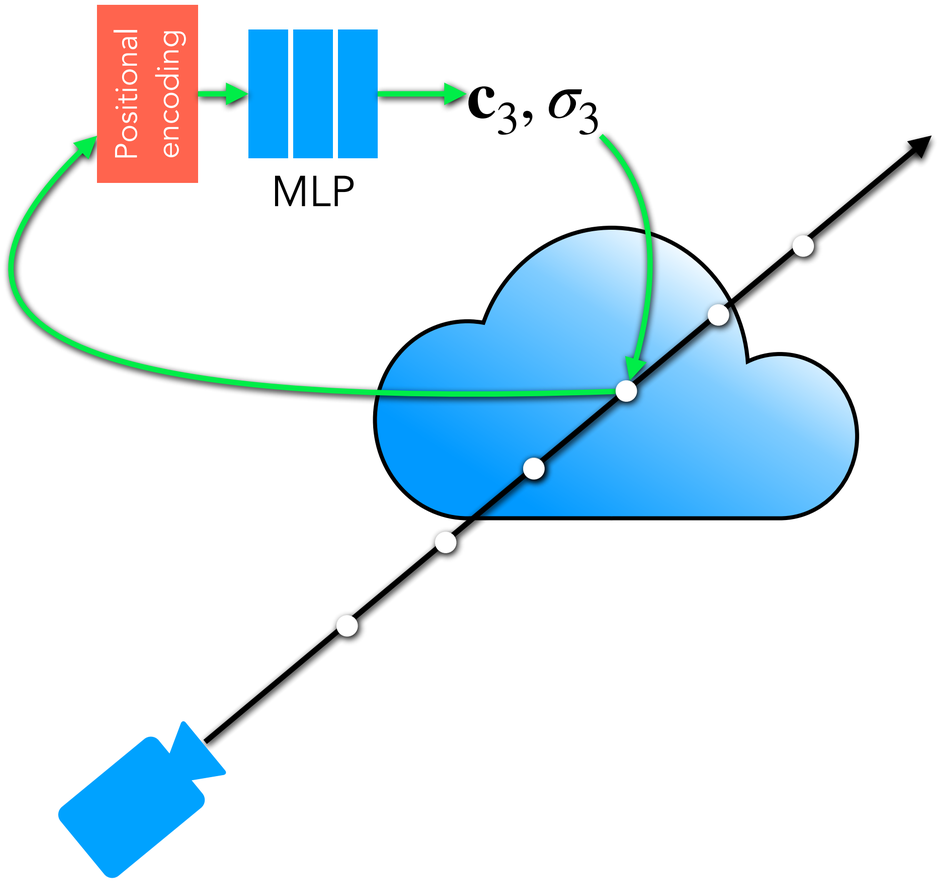
“How do we store the values of $\mathbf{c}, \sigma$ at each point in space?”



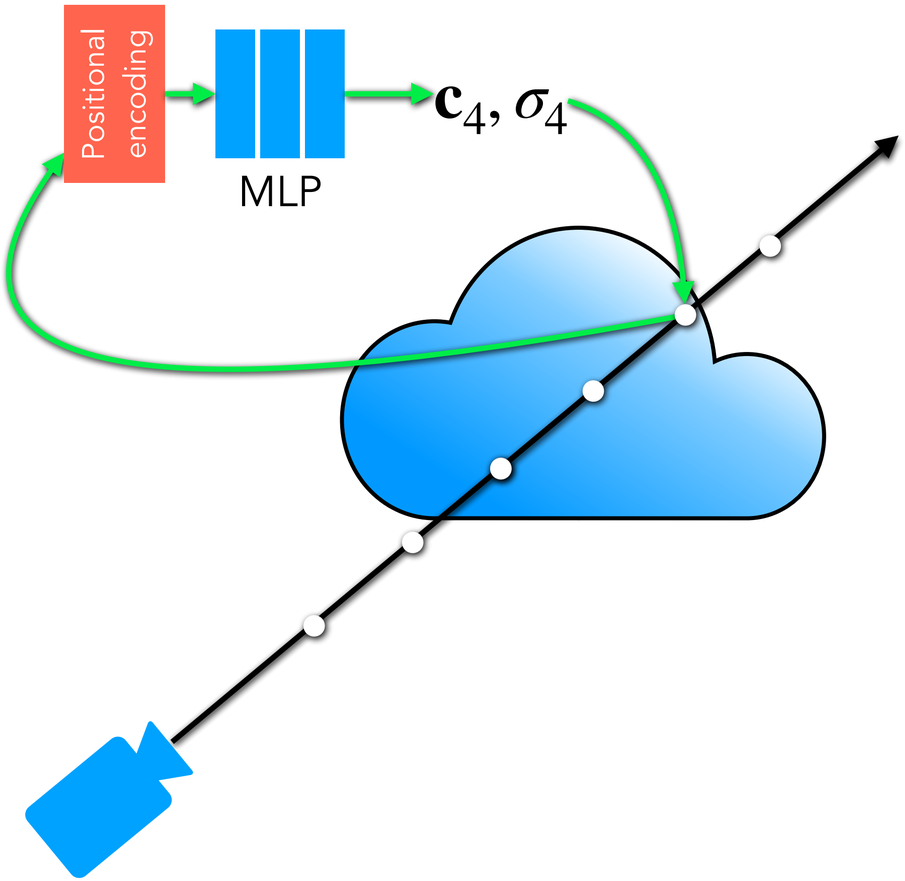
“How do we store the values of $\mathbf{c}, \sigma$ at each point in space?”



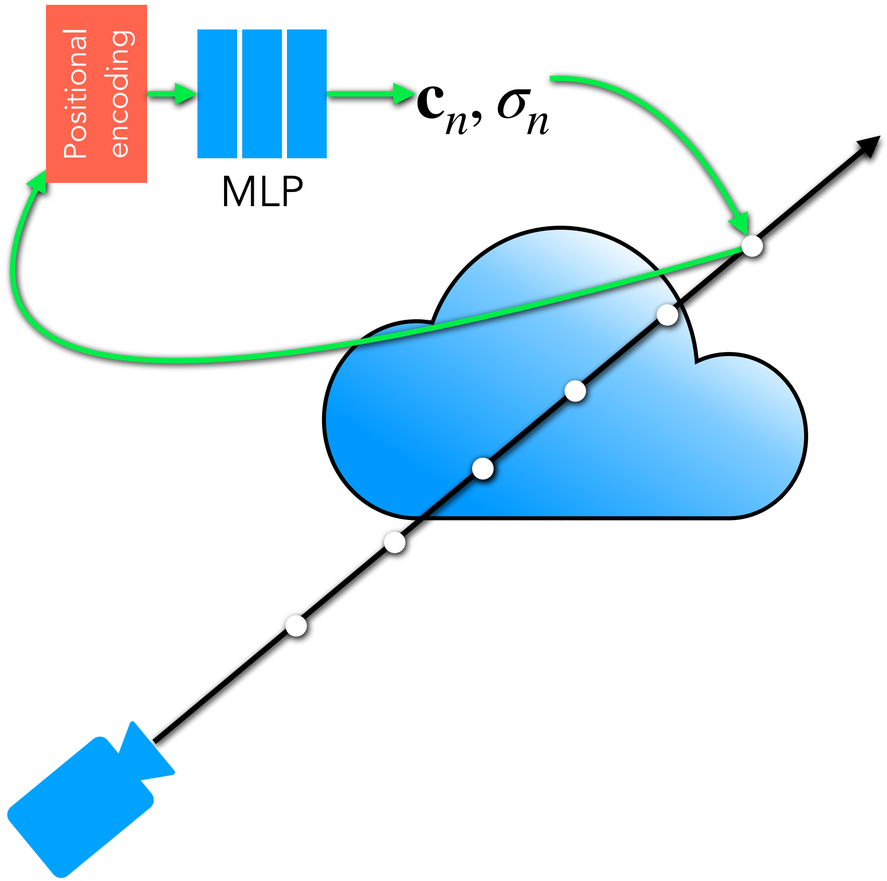
“How do we store the values of $\mathbf{c}, \sigma$ at each point in space?”



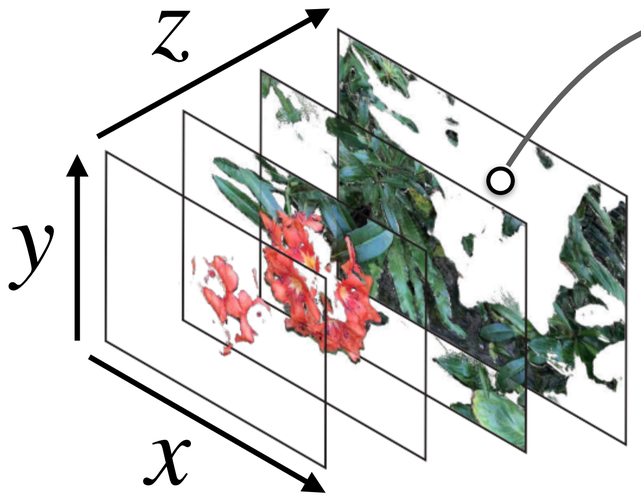
“How do we store the values of $\mathbf{c}, \sigma$ at each point in space?”



“How do we store the values of $\mathbf{c}, \sigma$ at each point in space?”



Use neural network to replace large N-d array



$$(r, g, b, \sigma)$$

versus

$$(x, y, z, \theta, \phi) \rightarrow \boxed{\quad} \rightarrow (r, g, b, \sigma)$$

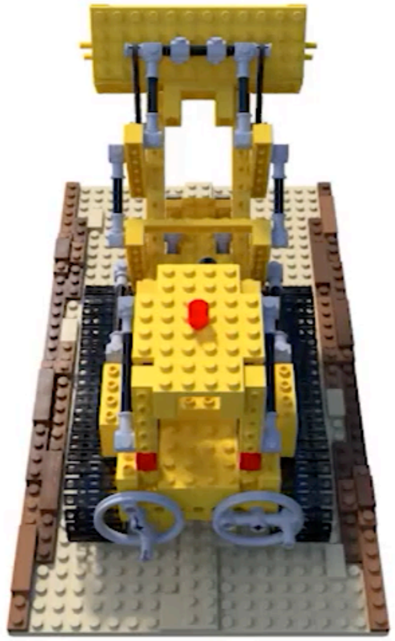
$$F_\Omega$$

Train network using gradient descent to reproduce all input views of scene

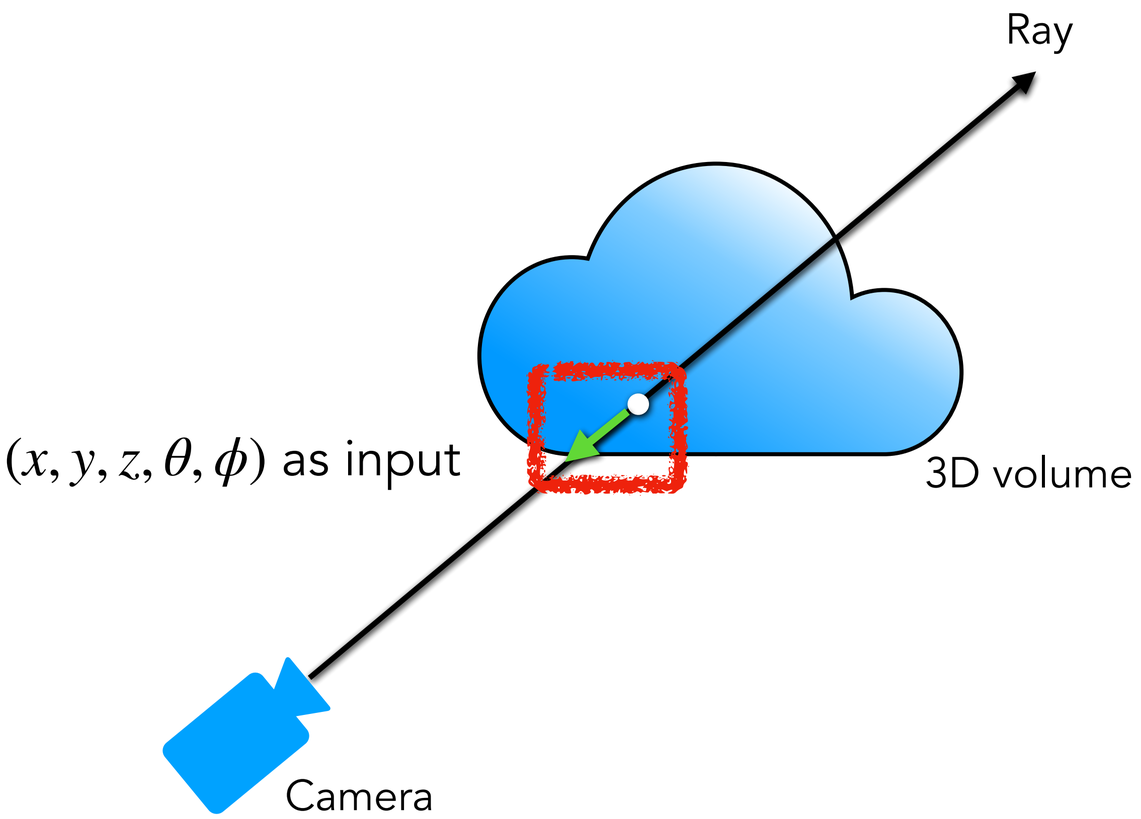
Volume rendering of MLP colors/densities

Ground truth image

$$\nabla \left\| \quad - \quad \right\|^2$$

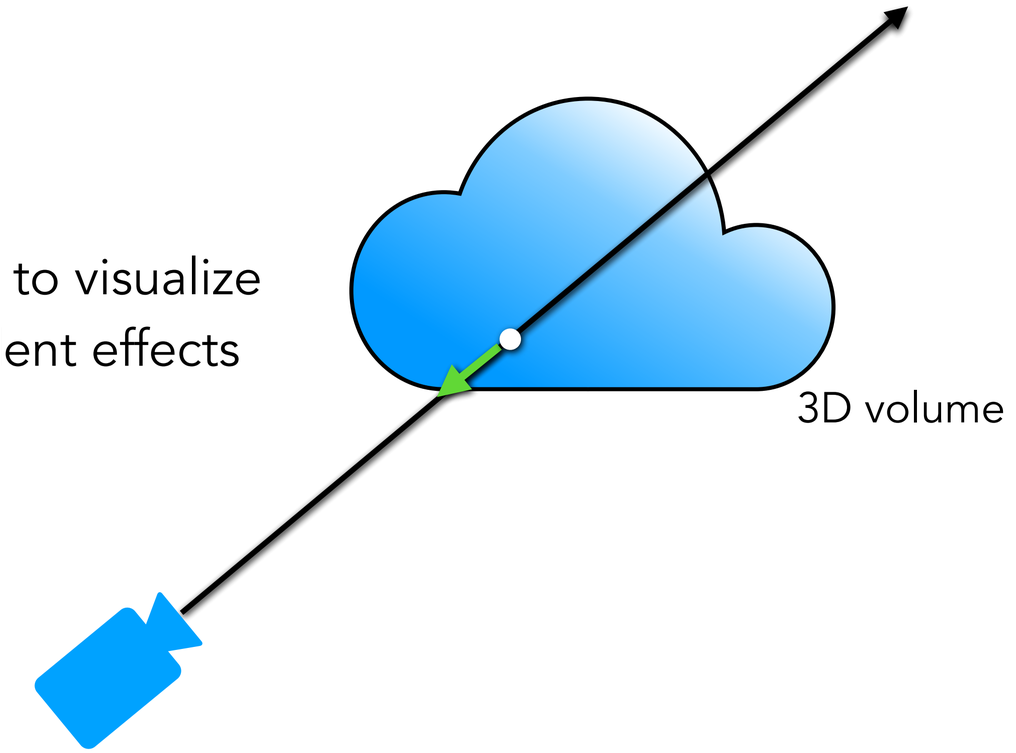


Viewing directions as input



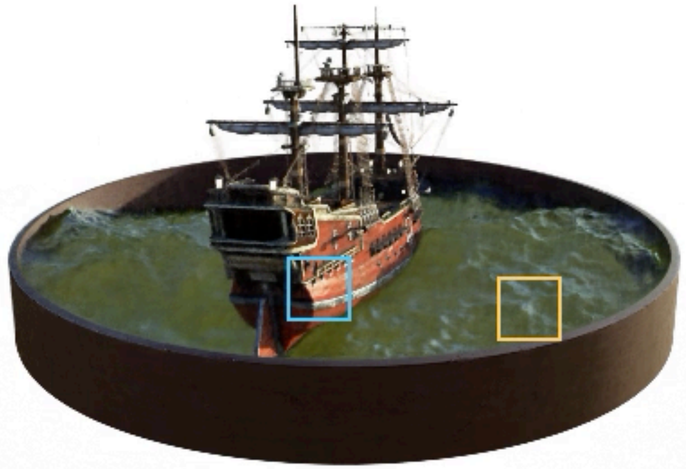
Viewing directions as input

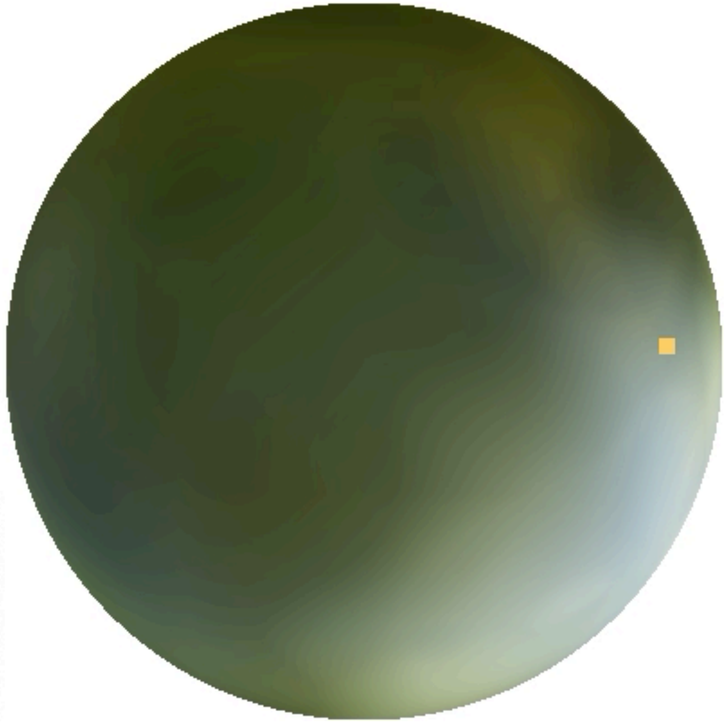
Change (θ, ϕ) to visualize view-dependent effects



Visualizing view-dependent effects

Radiance distribution for point on side of ship





Radiance distribution for point on water's surface

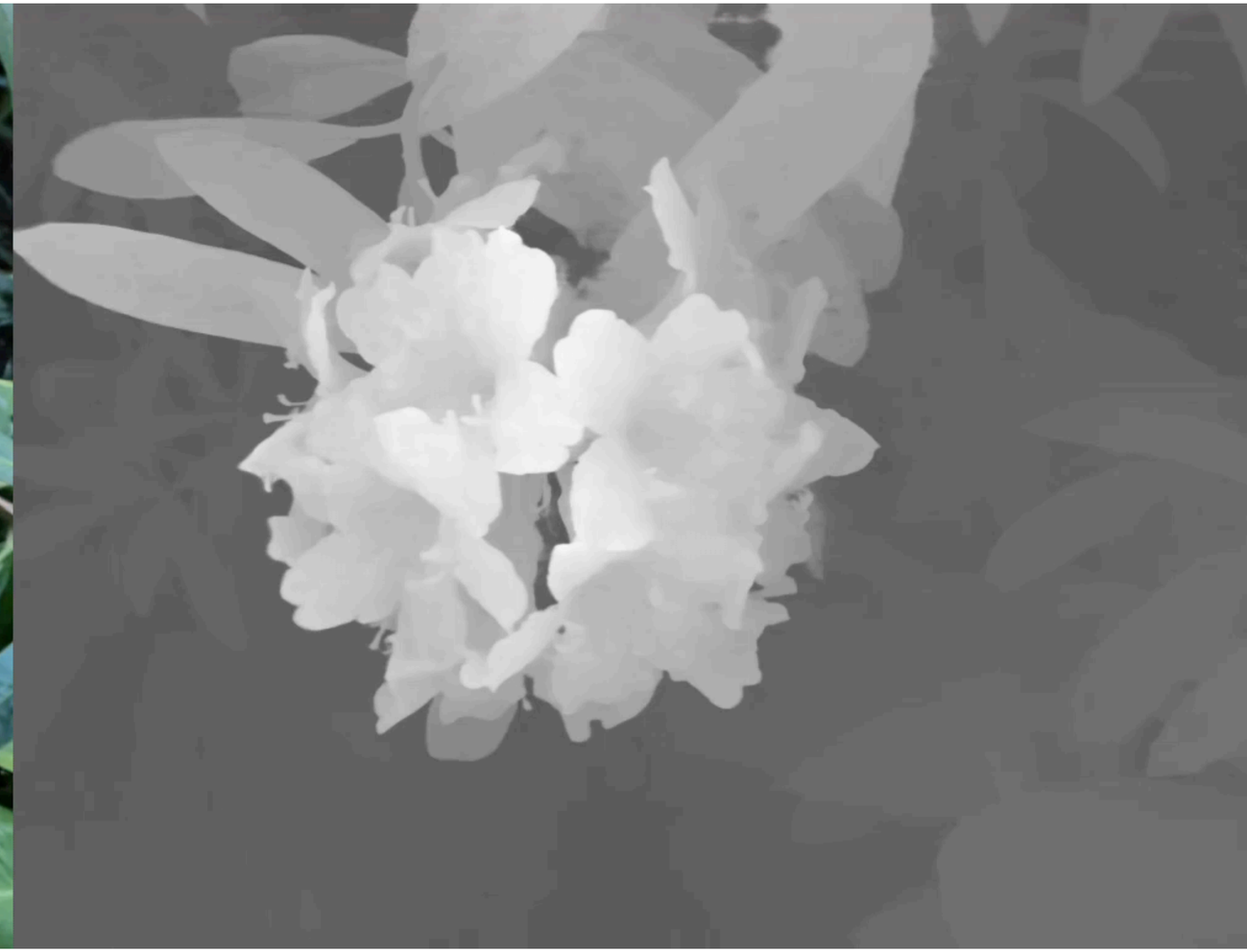
Visualizing view-dependent effects

Regular NeRF rendering

Manipulating input viewing directions

Visualizing learned density field as geometry

Regular NeRF rendering



Expected ray termination depth

Visualizing learned density field as geometry

Regular NeRF rendering

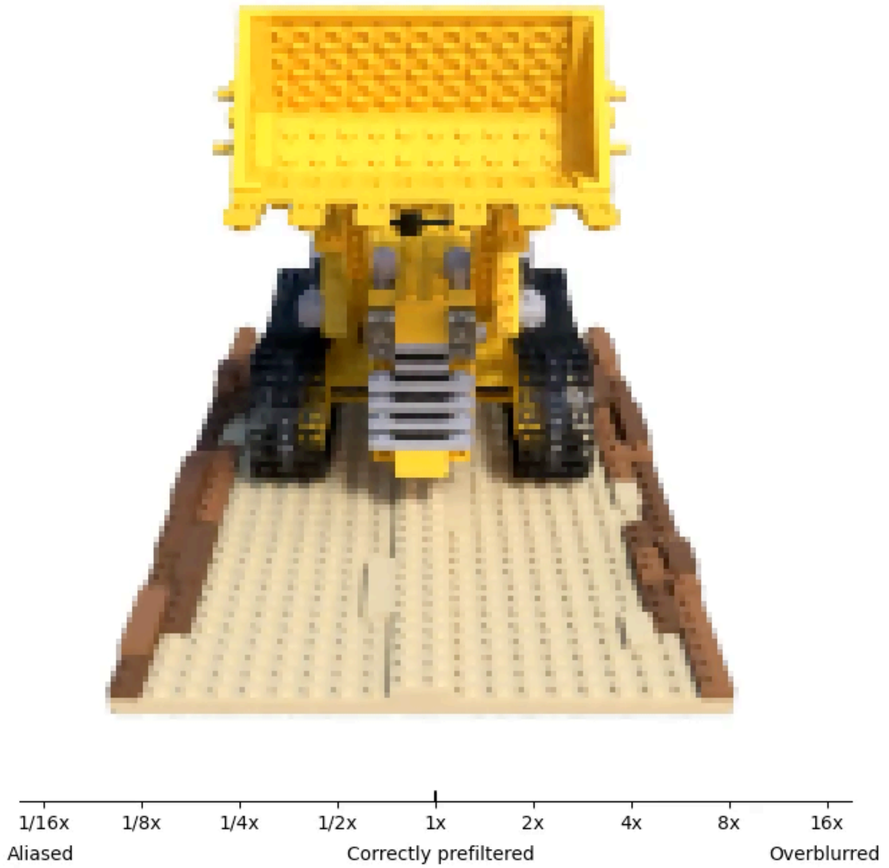
Expected ray termination depth

Overview

- Volumetric rendering math
- Neural networks as representations for spatial data
- Neural Radiance Fields (NeRF)
- NeRF improvements and extensions

NeRF problems

- Scene representation is not anti-aliased



[Barron] prefiltered positional encoding

Barron et al 2021, *Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields*

NeRF problems

- Rendering is very slow

[Hedman] realtime online viewer

Reiser et al 2021, *KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs*
Garbin et al 2021, *FastNeRF: High-Fidelity Neural Rendering at 200FPS*
Yu et al 2021, *PlenOctrees For Real-time Rendering of Neural Radiance Fields*
Hedman et al 2021, *Baking Neural Radiance Fields for Real-Time View Synthesis*

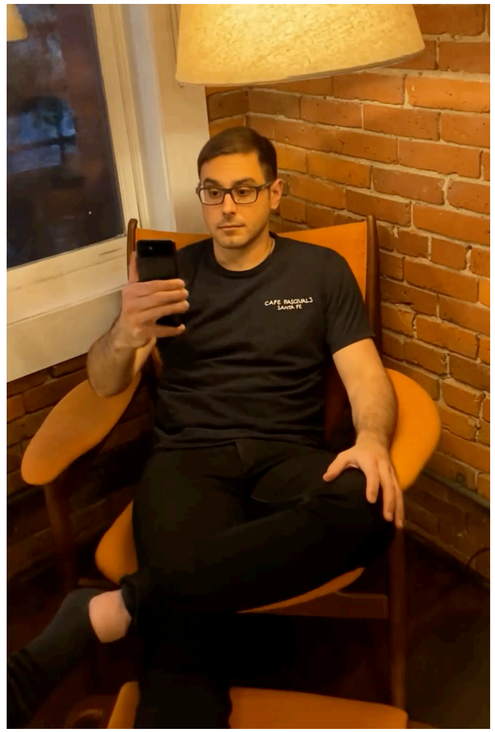
NeRF problems

- Network must be retrained for every scene
- Requires many input images

[Wang] network never trained on this scene!

Trevithick et al 2020, *GRF: Learning a General Radiance Field for 3D Scene Representation and Rendering*
Wang et al 2021, *IBRNet: Learning Multi-View Image-Based Rendering*
Yu et al 2021, *pixelNeRF: Neural Radiance Fields from One or Few Images*

NeRF problems



[Park] trained on selfie video

[Srinivasan] trained on multiple lighting conditions

[Tancik] trained on tourist photos

- Needs scene to be static and have fixed lighting

Bi et al 2020, *Neural Reflectance Fields for Appearance Acquisition*
Park et al 2020, *Nerfies: Deformable Neural Radiance Fields*
Li et al 2021, *Neural Scene Flow Fields for Space-Time View Synthesis of Dynamic Scenes*
Srinivasan et al 2021, *NeRV: Neural Reflectance and Visibility Fields for Relighting and View Synthesis*
Tancik et al 2021, *Learned Initializations for Optimizing Coordinate-Based Neural Representations*
Martin-Brualla et al 2021, *NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections*

• • •

https://github.com/yenchenlin/awesome-NeRF

Neural Volumetric Rendering: NeRF, etc.

SIGGRAPH 2021

Neural Rendering Course

Ben Mildenhall
Google Research
bmild.github.io