Neural Radiance Fields pt 3

made with nerfstudio

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

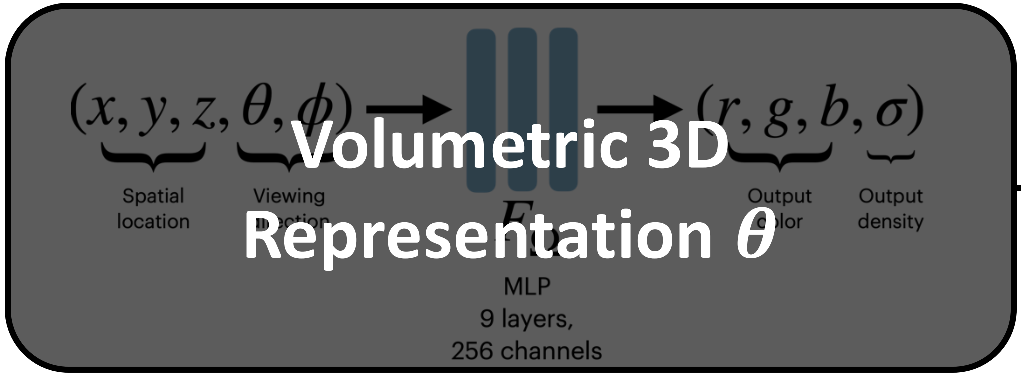
Angjoo Kanazawa and Alexei Efros

UC Berkeley Fall 2025

Lots of content from Noah Snavely and Ben Mildenhall, Pratul Srinivasan, and Matt Tancik from [ECCV 2022 Tutorial on Neural Volumetric Rendering for Computer Vision]

Where we are

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



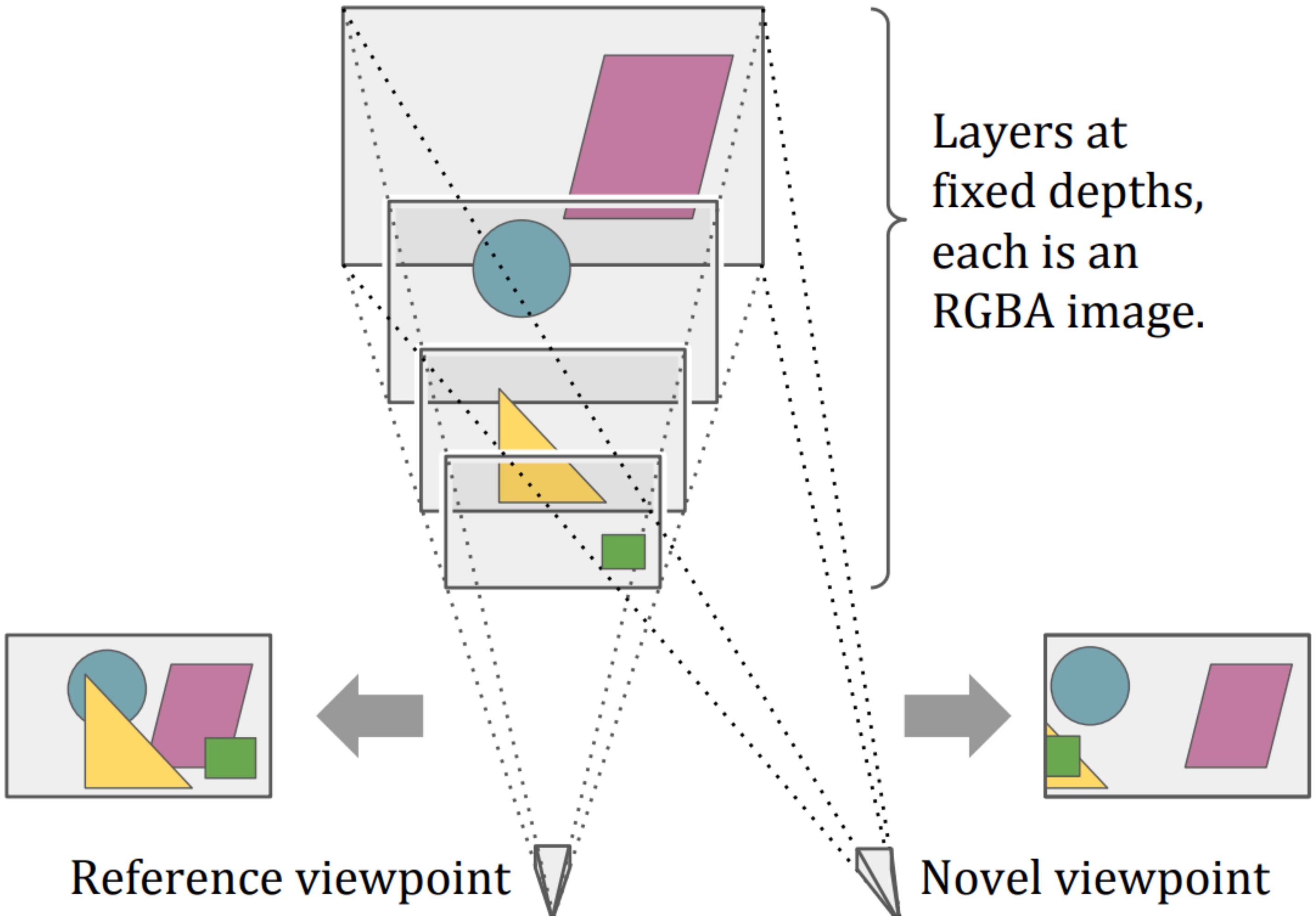
Differentiable
Volumetric Rendering

Rendered Image:
I'

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

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

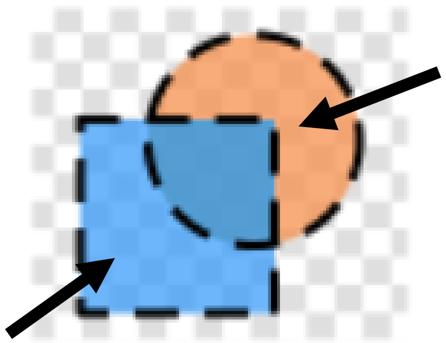
A Precursor: Multi-plane Images



Zou et al. Stereo Magnification, SIGGRAPH 2018

Alpha Blending

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



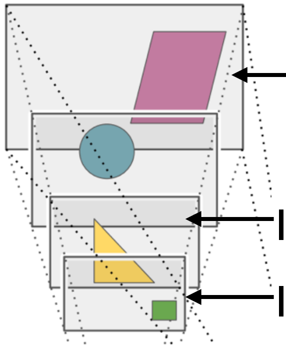
(C_b, α_b)

(C_a, α_a)

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

General D layer case:

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

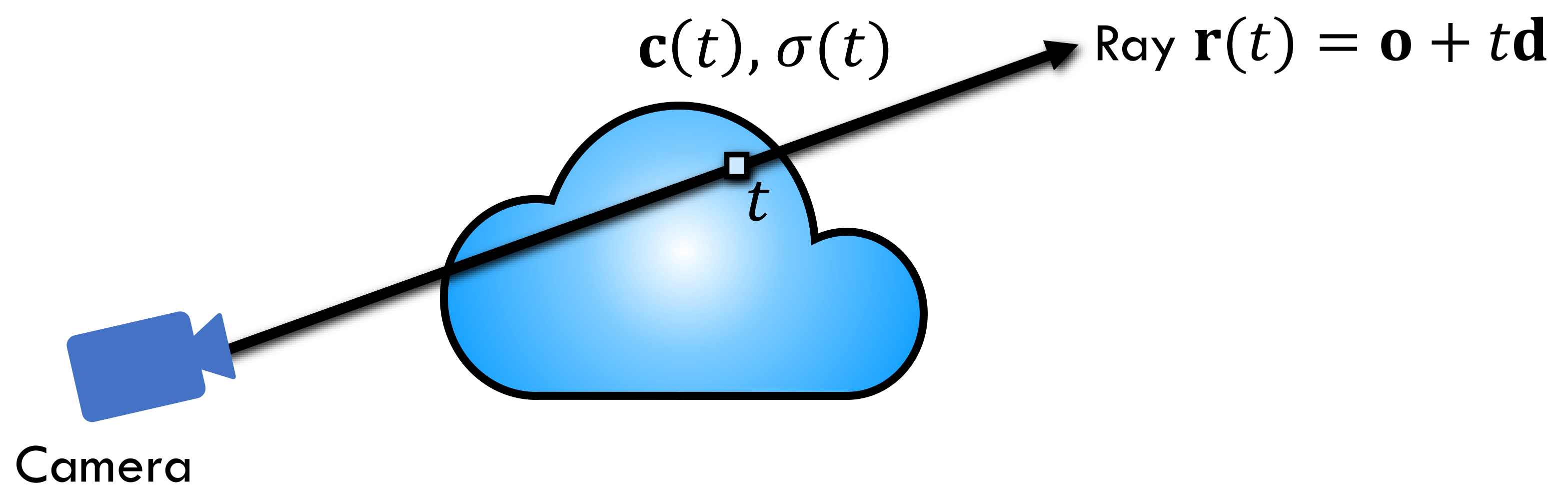


Volumetric formulation for NeRF

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

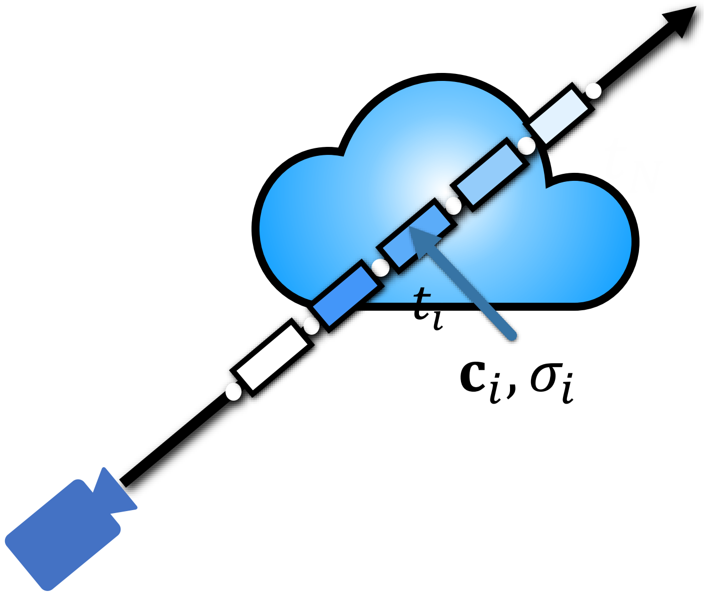
$$\boldsymbol{c}(\boldsymbol{r}) = \int_{t_0}^{t_1} P[first\ hit\ at\ t]\boldsymbol{c}(t)dt$$

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

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

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

where $T_i = \prod_{j=1}^{i-1}(1-\alpha_j) \quad \alpha_i = 1 - \exp(-\sigma_i \delta_i)$



Differentiable Volumetric Rendering Formula

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

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

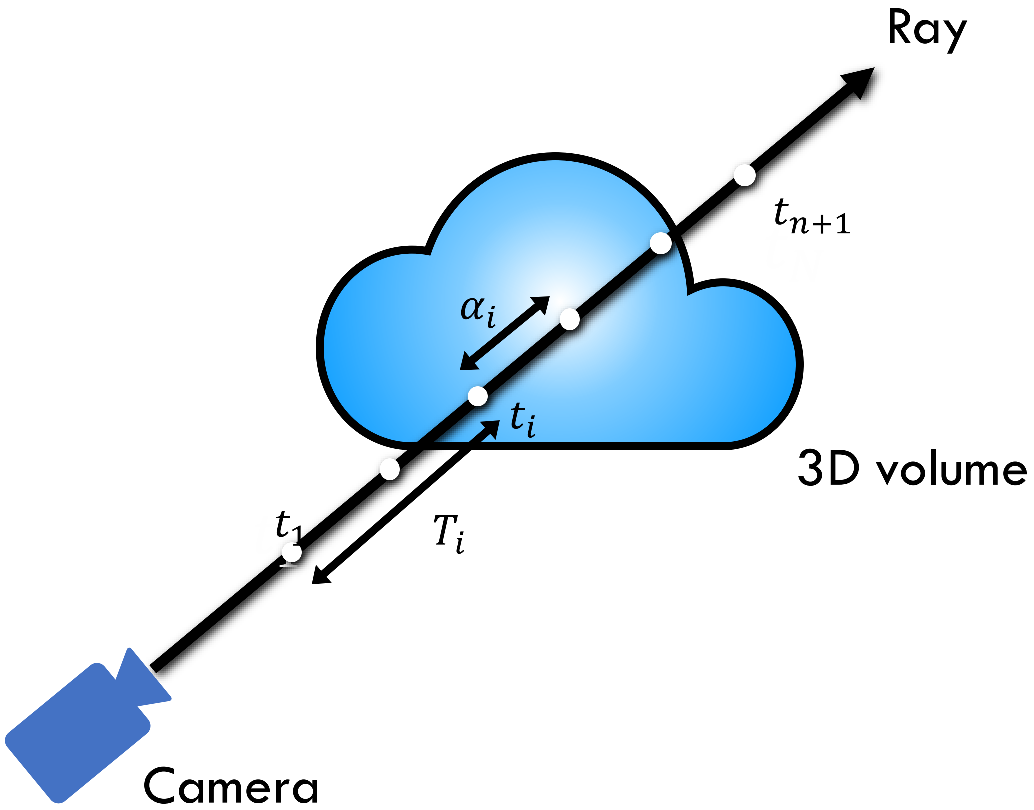
$$\mathbf{c} \approx \boxed{\sum_{i=1}^{n} w_i \mathbf{c}_i}$$

colors

weights

How much light is blocked earlier along ray:

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



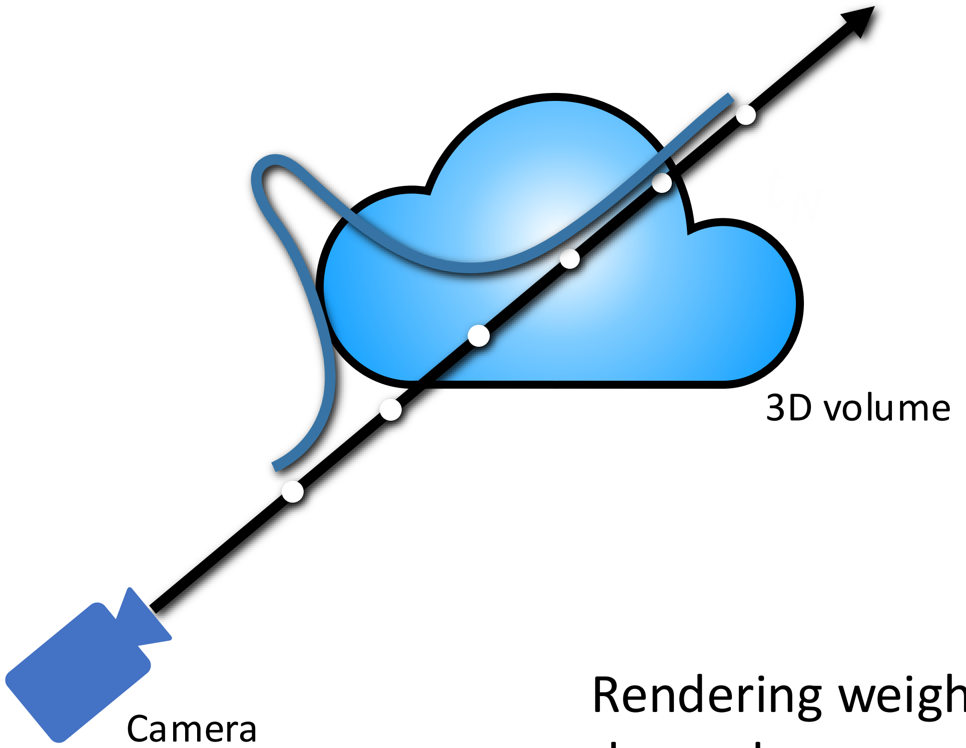
How much light is contributed by ray segment *i*:

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

Delta is the length of each segment: $\delta_i = t_{i+1} - t_i$

Visual intuition: rendering weights is specific to a ray

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

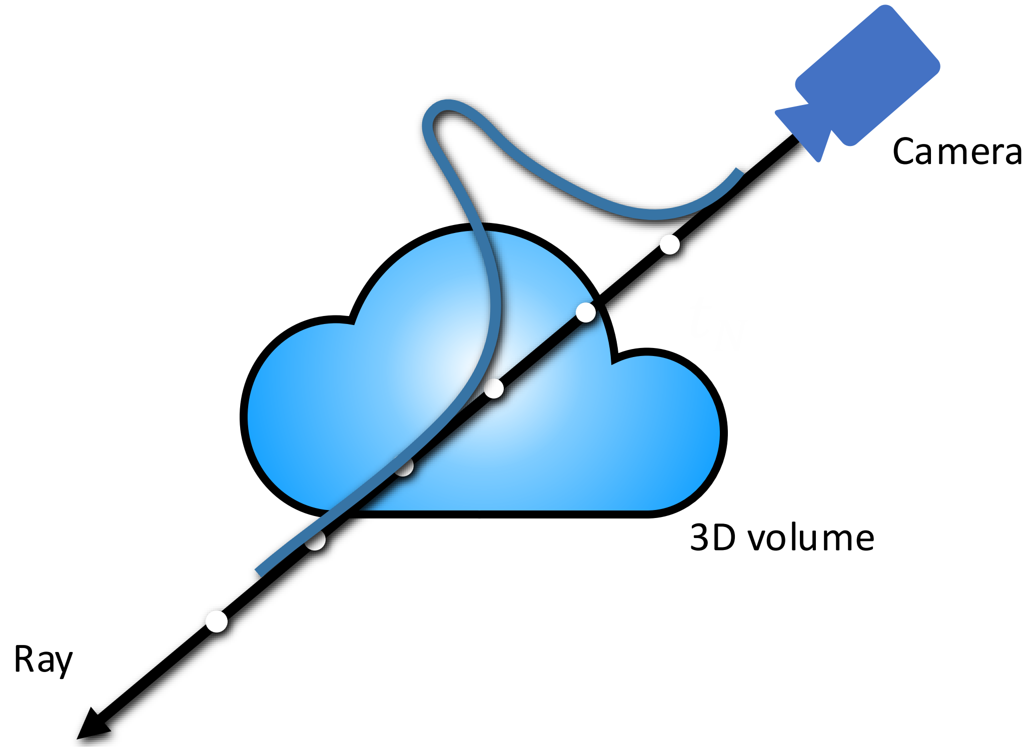


Rendering weights are not a 3D function — depends on ray, because of tranmisttance!

10

Visual intuition: rendering weights is specific to a ray

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

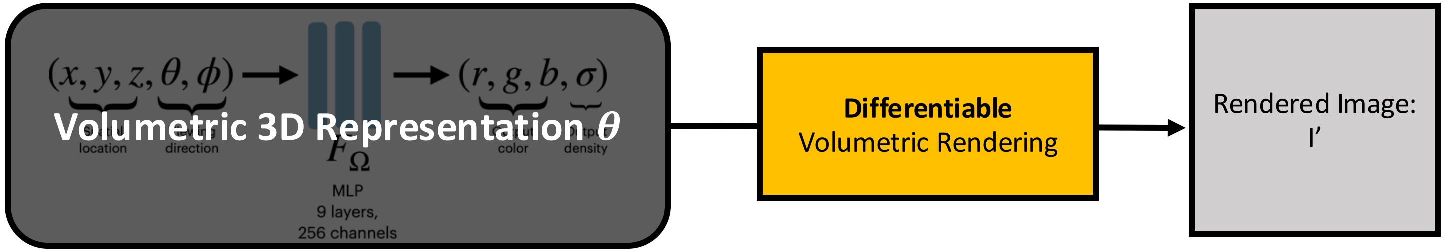


Rendering weights are not a 3D function — depends on ray, because of tranmisttance!

11

What’s the point

- Remember, for each pixel or a ray we render a color with this formula based on the Volumetric 3D Representation
- We use this to supervise the 3D Representation (sigma, RGB volume)



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

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

Connection to alpha compositing

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

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

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

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

$$\prod_i \exp(x_i) = \exp(\sum_i x_i)$$
$$\alpha_i = 1 - \exp(\sigma_i \delta_i)$$
$$1 - \alpha_i = -\exp(\sigma_i \delta_i)$$

Summary

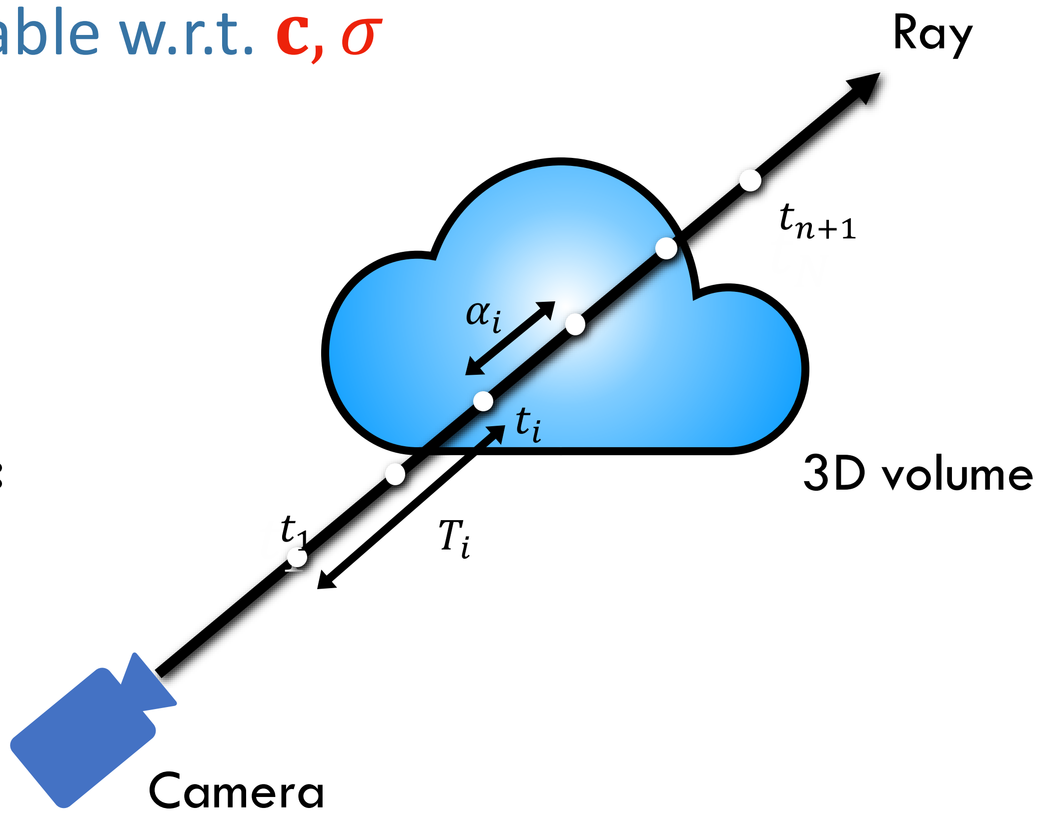
for a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

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



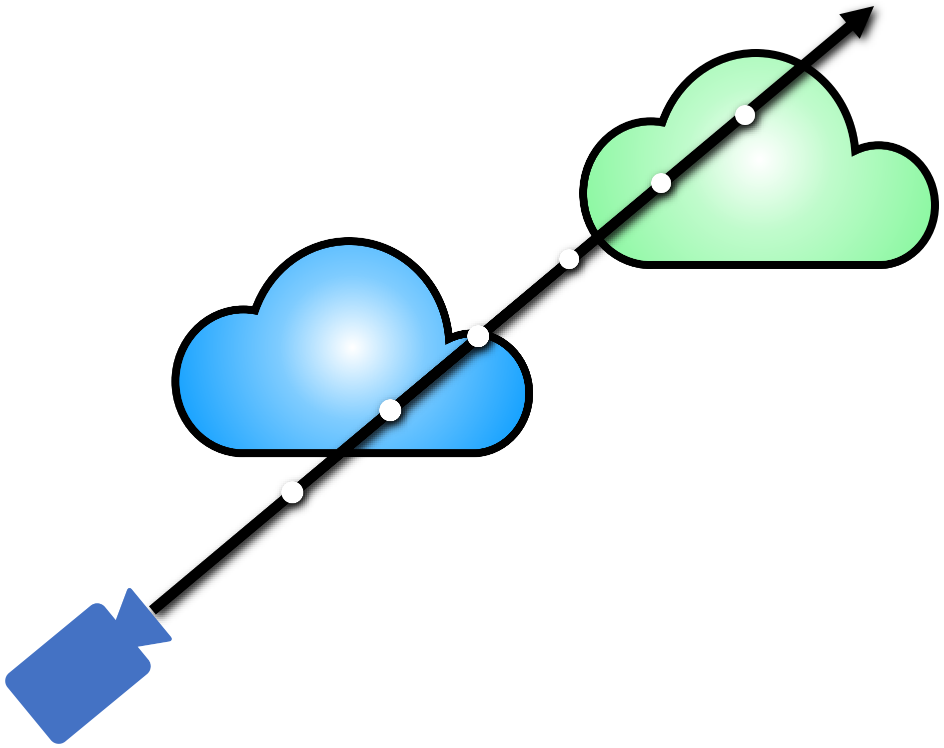
How much light is blocked earlier along ray:

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

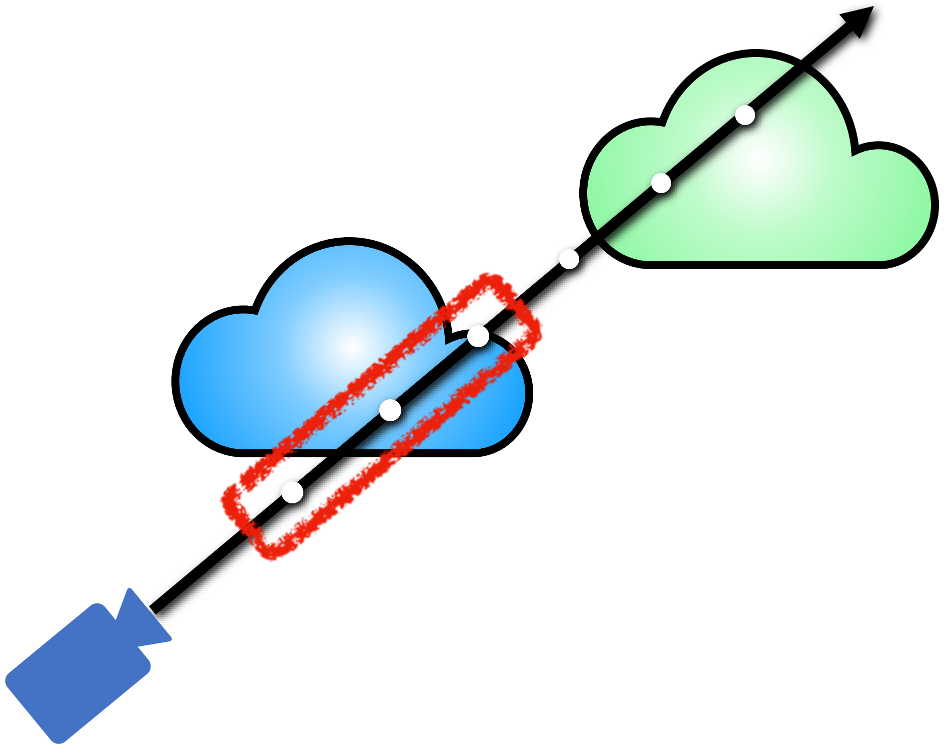
How much light is contributed by ray segment *i*:

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

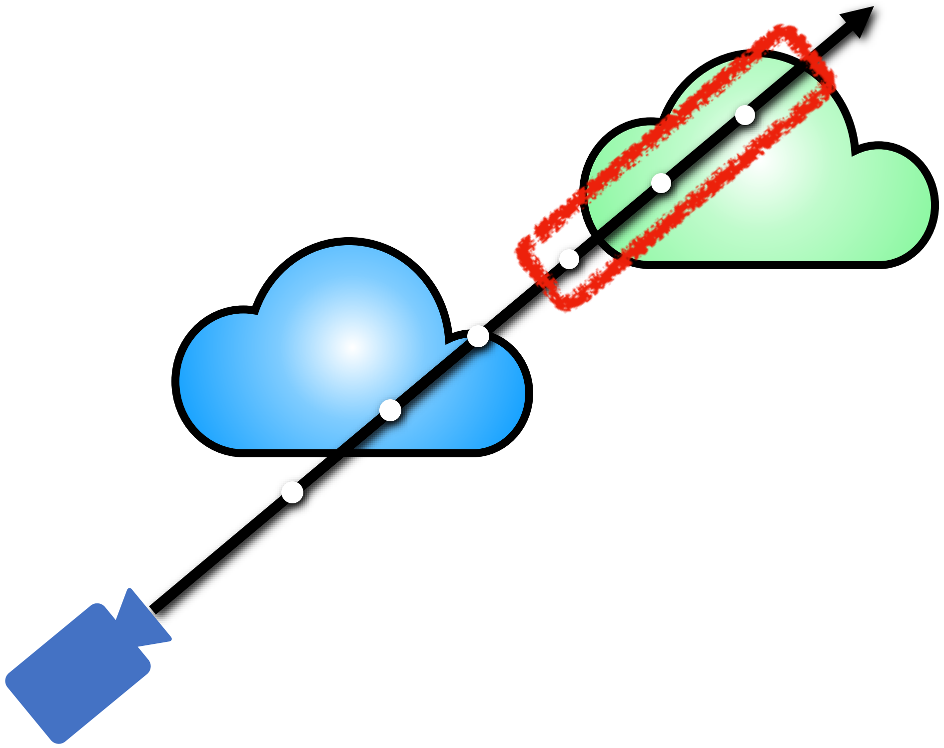
Alpha mattes and compositing



Alpha mattes and compositing



Alpha mattes and compositing



Alpha mattes and compositing

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

Poole et al 2022, DreamFusion

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

Rendering weight PDF is important

Remember, expected color is equal to

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

$T(t)\sigma(t)$ 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

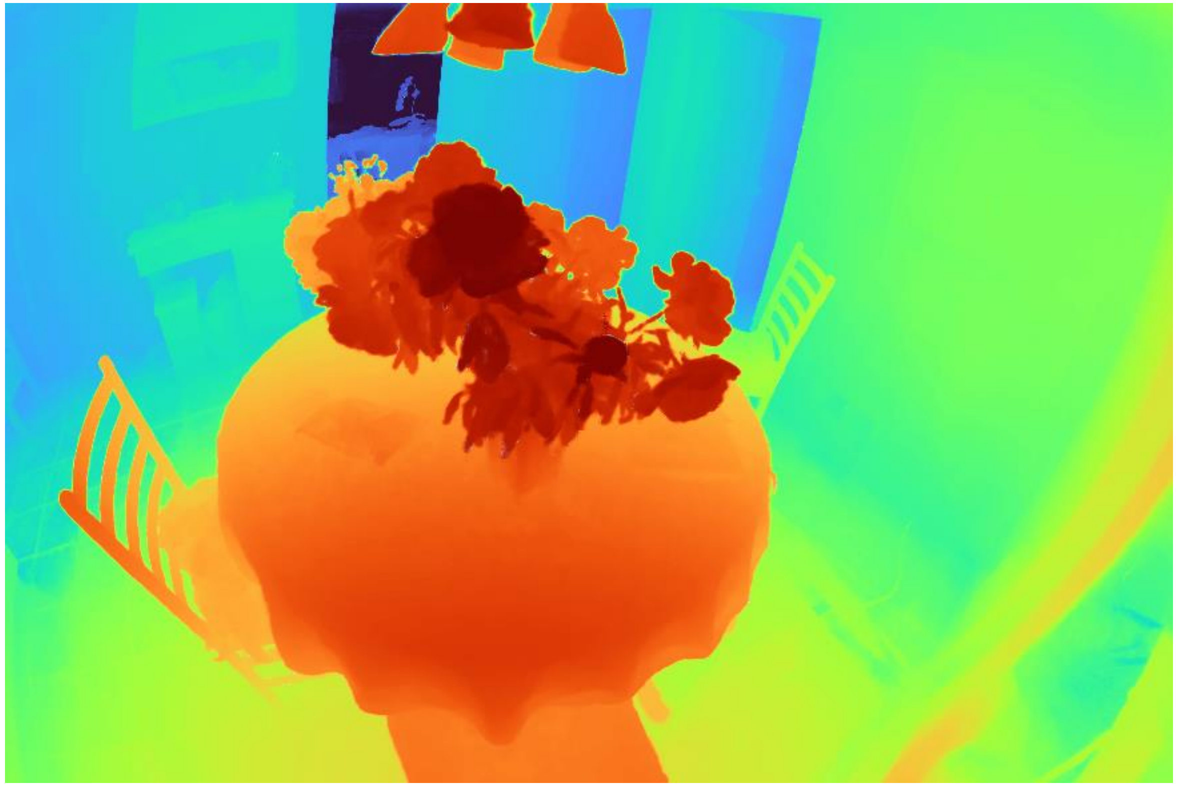
We can use this distribution to compute expectations for other quantities, e.g. "expected depth":

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

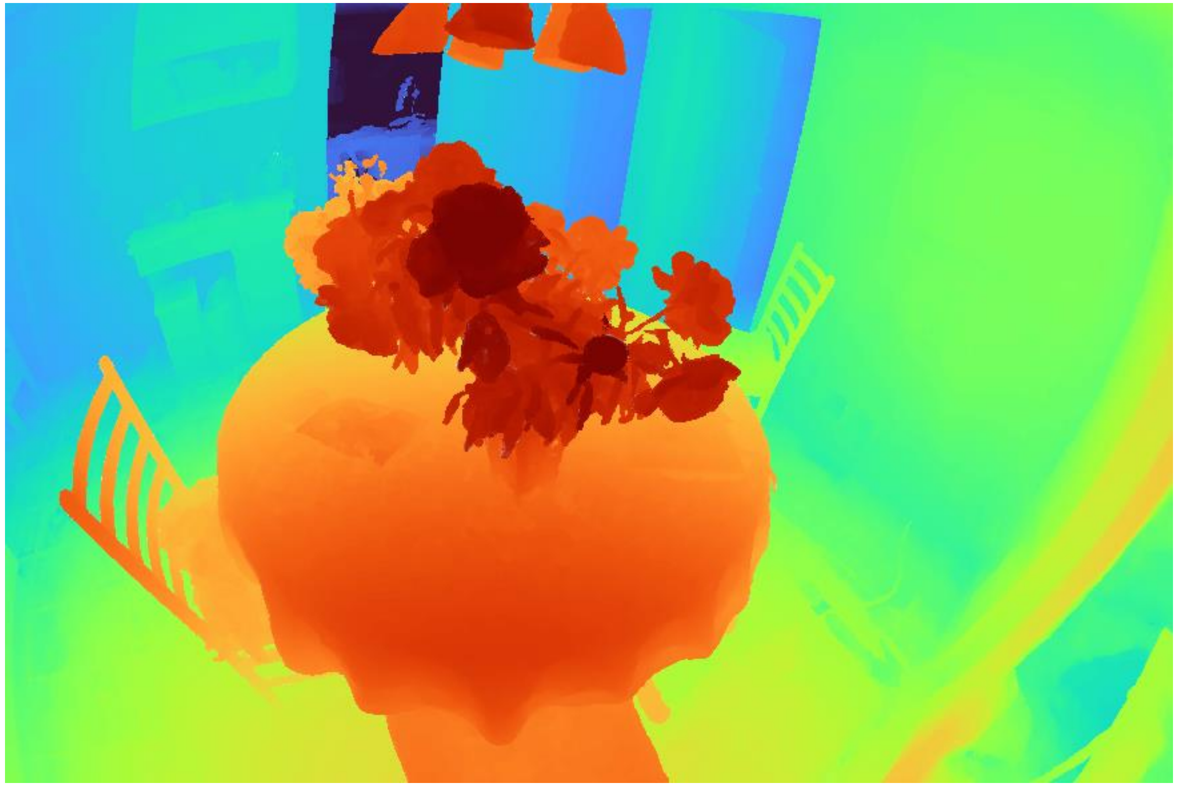
This is often how people visualise NeRF depth maps.

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

Rendering weight PDF is important — depth



Mean depth



Median depth

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.)

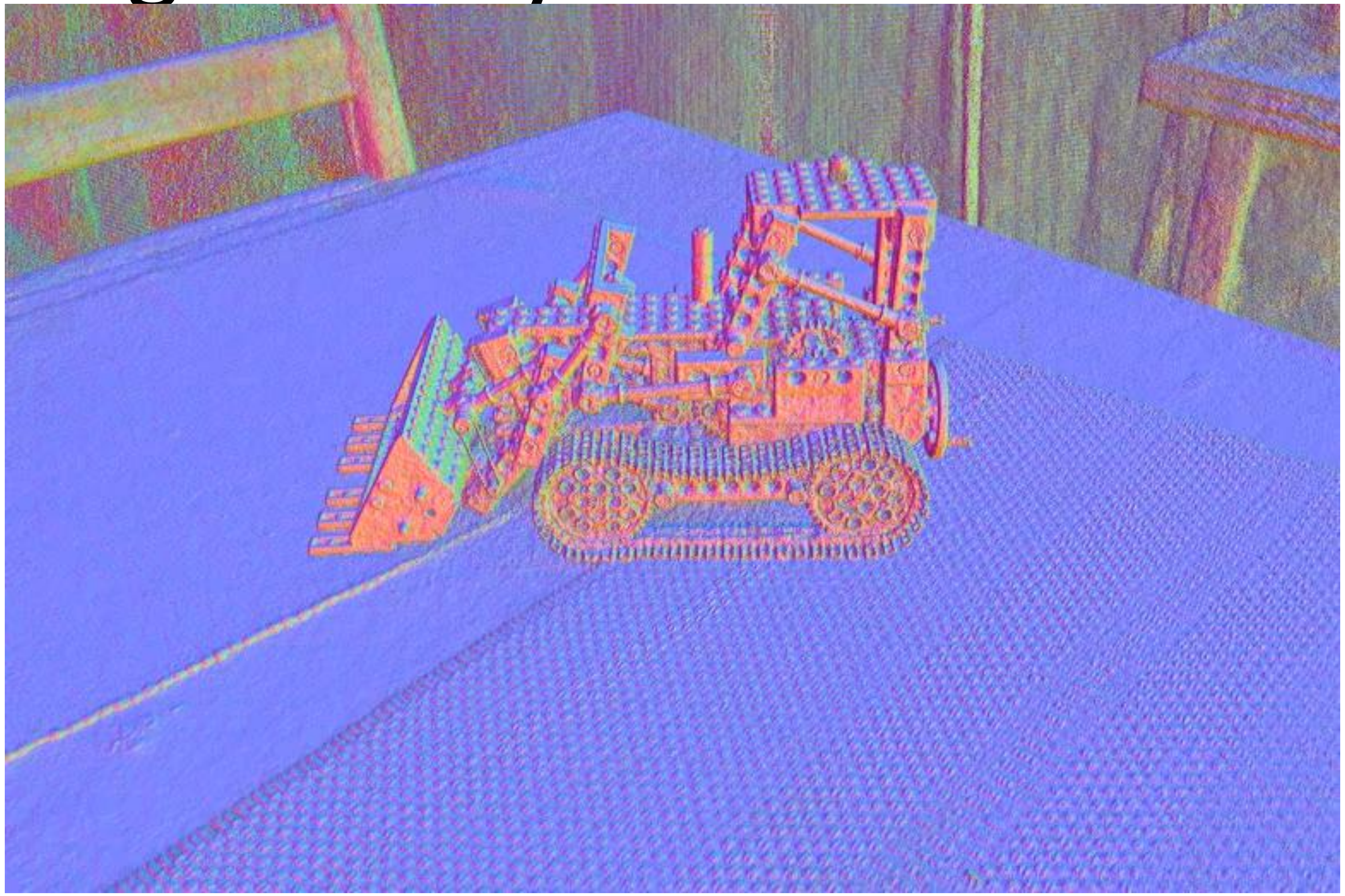
$$\sum_i T_i \alpha_i \mathbf{v}_i$$

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

Volume Rendering CLIP features

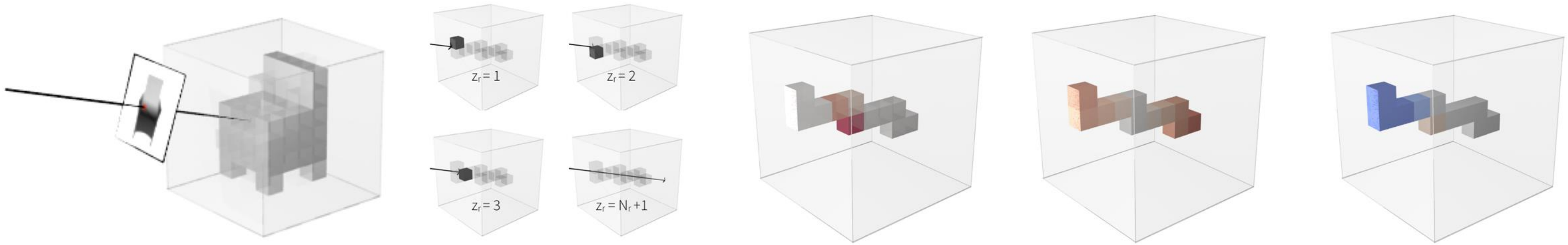
LERF: Language Embedded Radiance Fields, Kerr* and Kim* et al. ICCV 2023

Density as geometry



Normal vectors (from analytic gradient of density)

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!

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

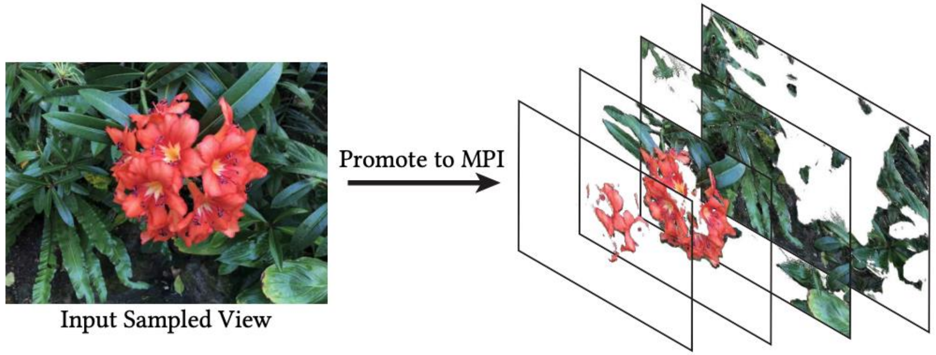
Tulsiani et al 2017, Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency

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)

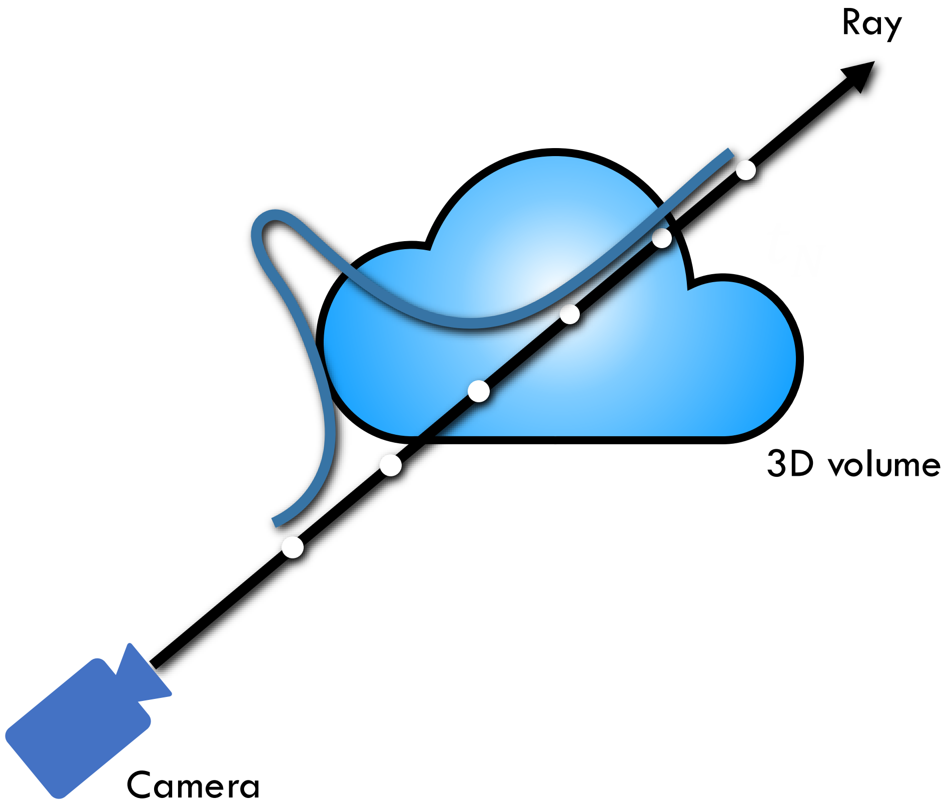
Typical deep learning pipelines - images go into a 3D CNN, big RGBA 3D volume comes out



Neural Volumes
(Lombardi et al. 2019)
Direct gradient descent to optimize an RGBA volume, regularized by a 3D CNN

Signal Processing Consideration in NeRFs

What is this process?



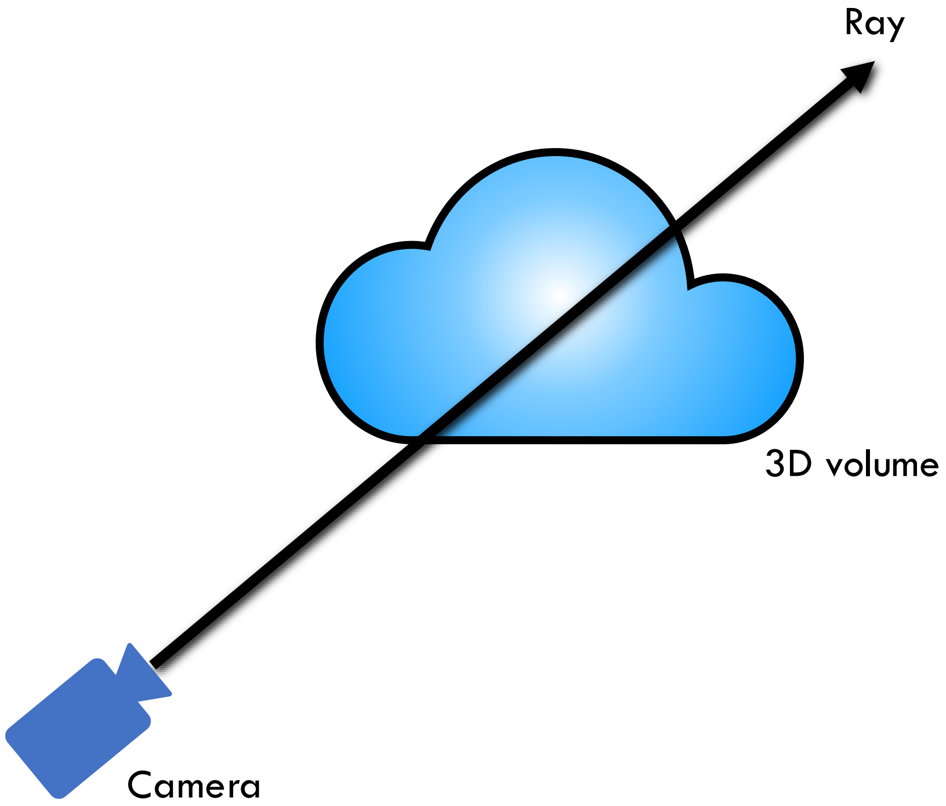
What is happening here?

Aliasing!!

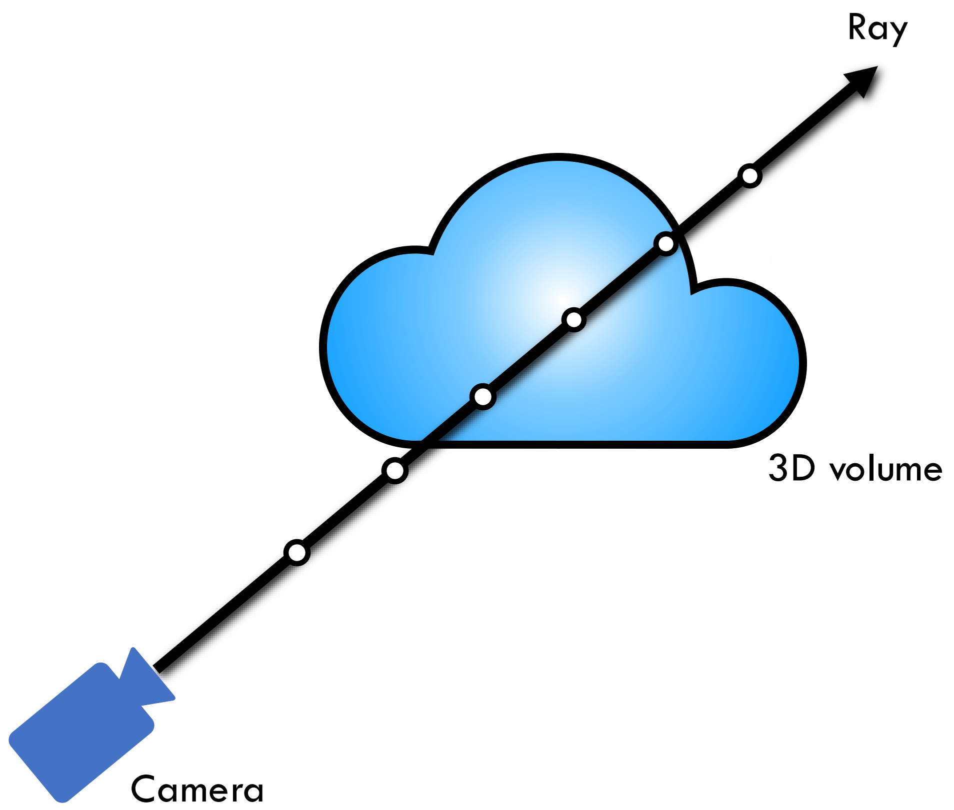
Naïve (original) NeRF

Slide Credit: Pratul Srinivasan

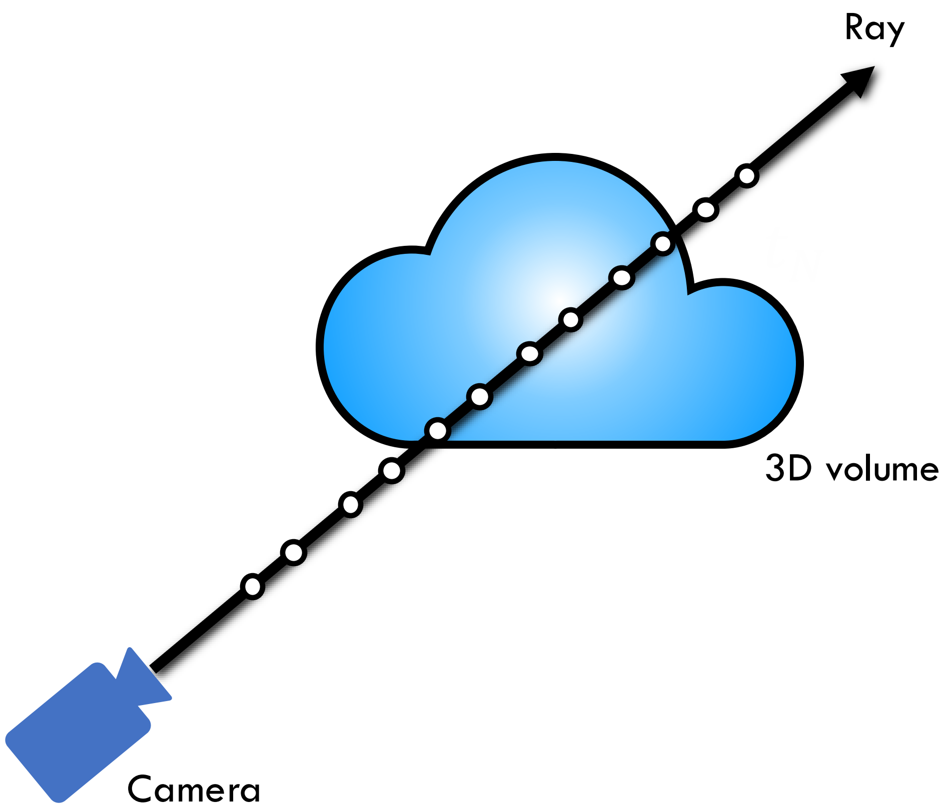
Where to place samples along rays?



How to be more efficient than dense sampling?



How to be more efficient than dense sampling?



Hierarchical Sampling vs. Acceleration Structures

Hierarchical Sampling vs. Acceleration Structures

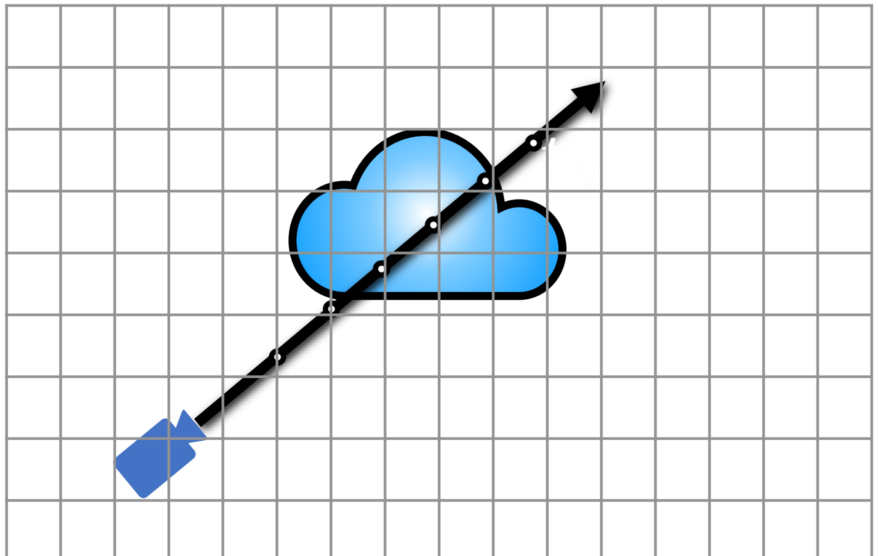
Hierarchical Sampling

Iteratively use samples from NeRF to more efficiently sample visible scene content

Acceleration Structures

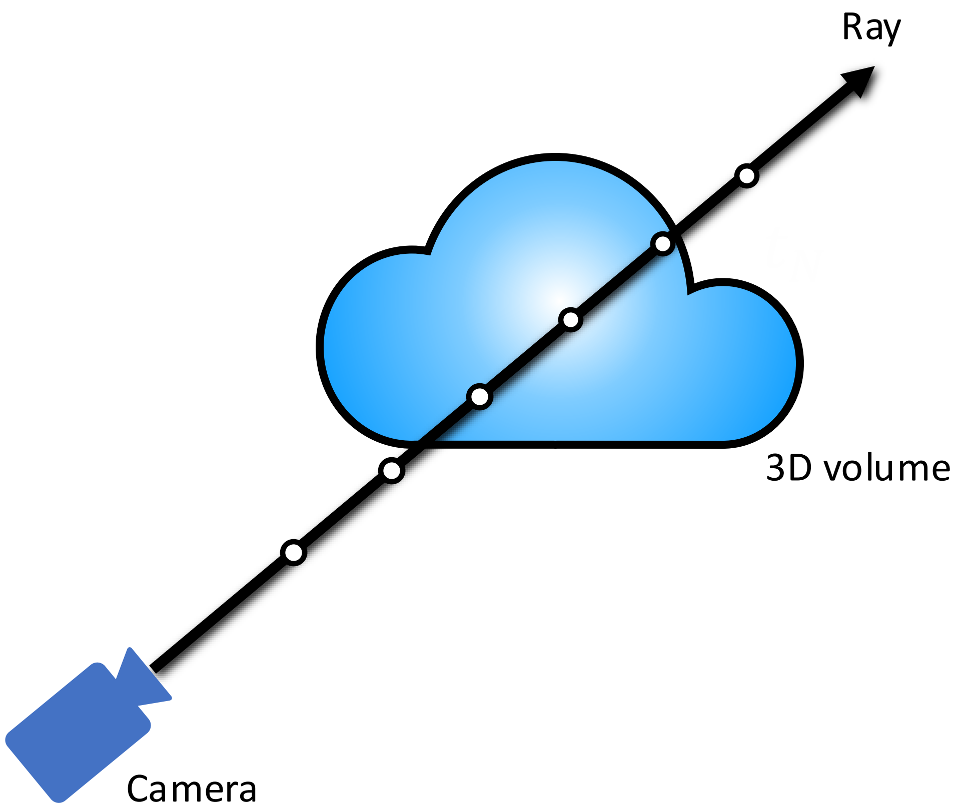
Distill/cache properties of NeRF into a structure that helps generate samples: e.g. Occupancy Grids

Straightforward compute —> storage tradeoff



Hierarchical ray sampling

Key Idea: sample points proportionally to expected effect on final rendering

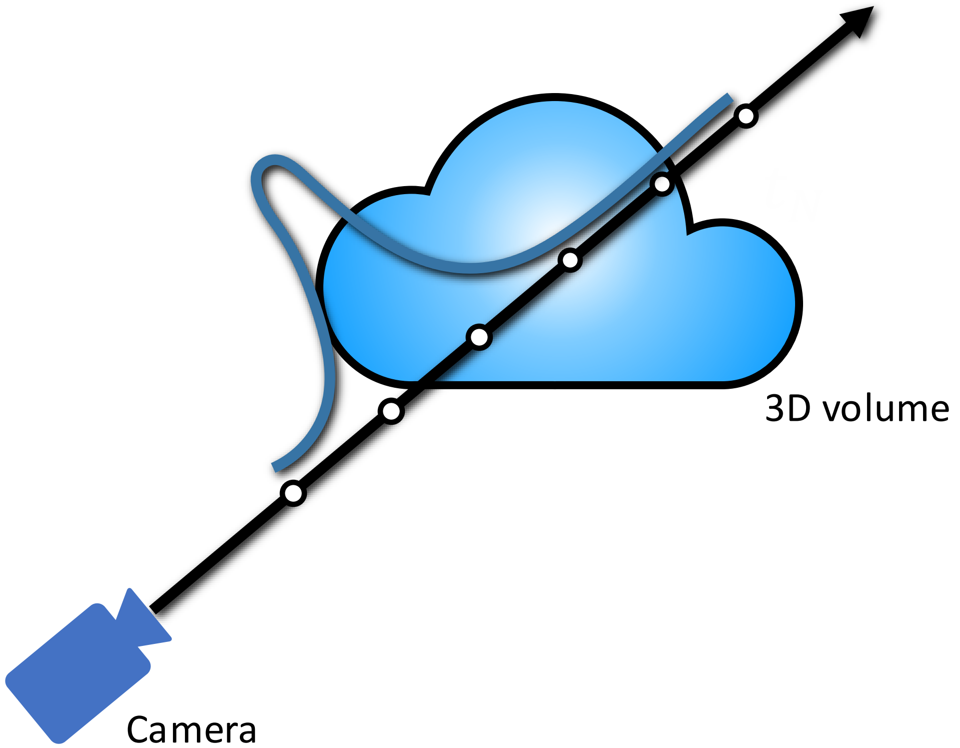


63

Key Idea: sample points proportionally to expected effect on final rendering

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

treat weights as probability distribution for new samples

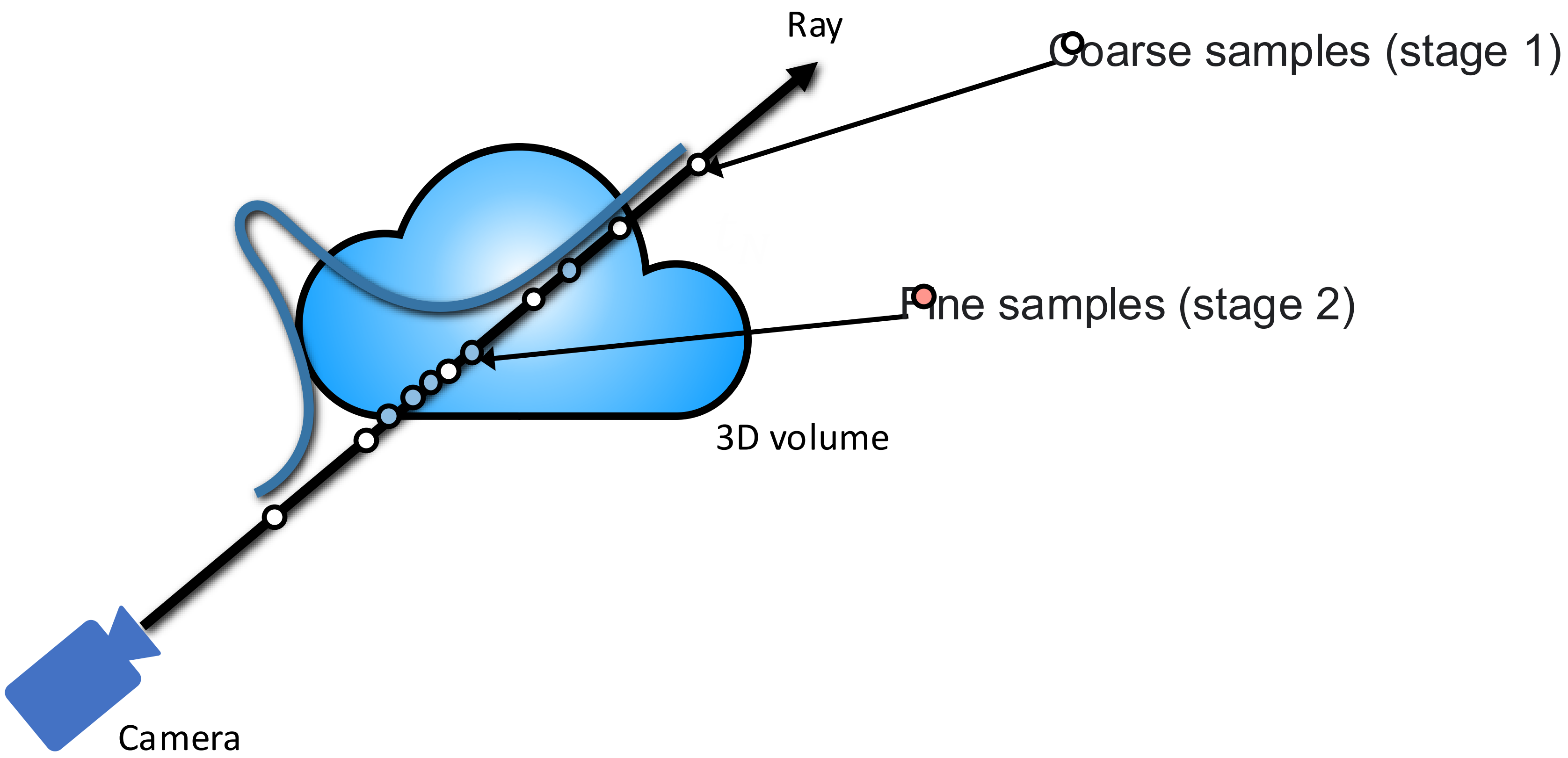


64

Key Idea: sample points proportionally to expected effect on final rendering

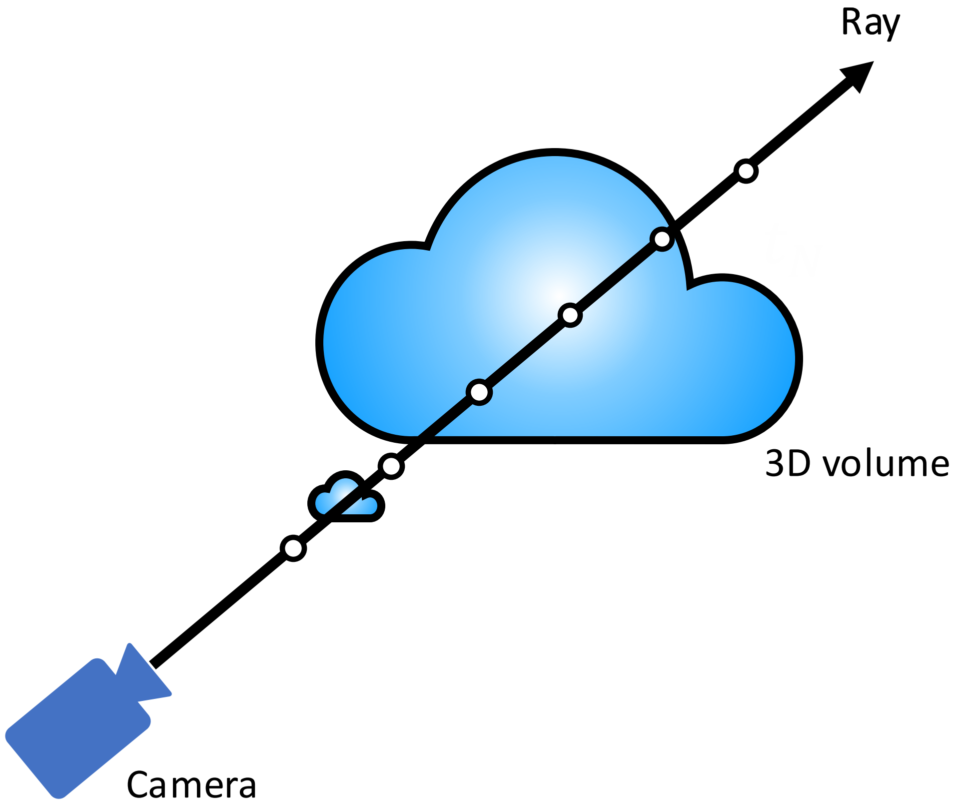
$$C \approx \sum_{i=1}^{N} \boxed{T_i \alpha_i} c_i$$

treat weights as probability distribution for new samples

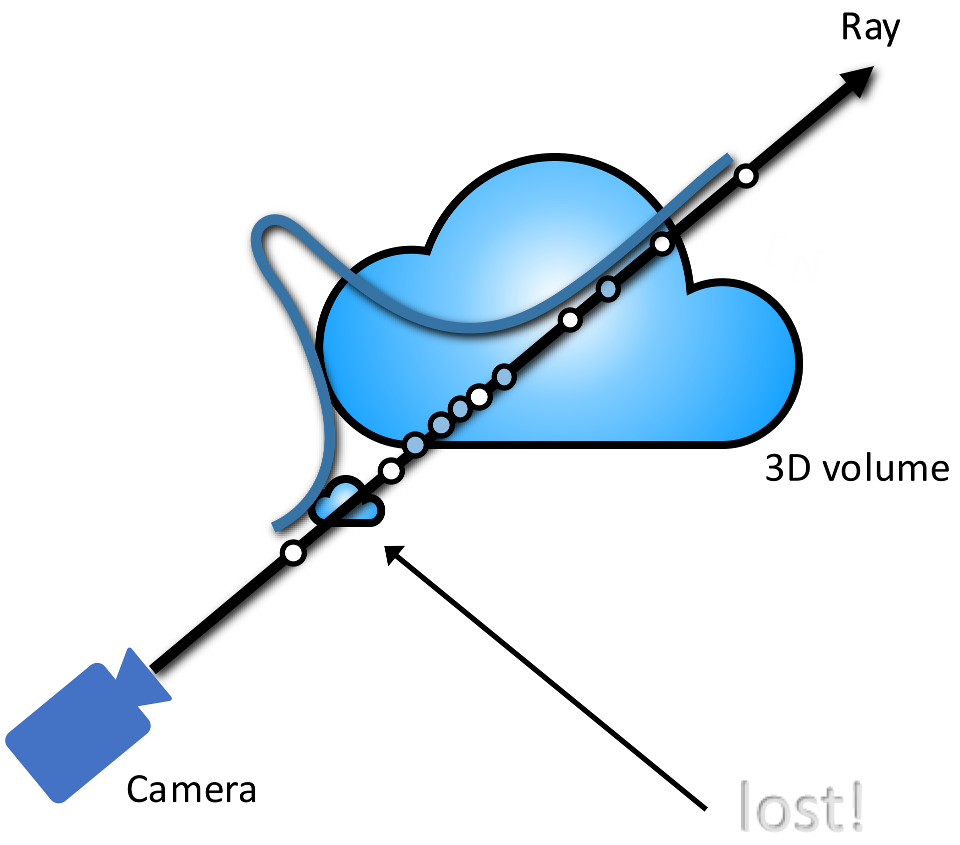


65

What about aliasing during coarse sampling?

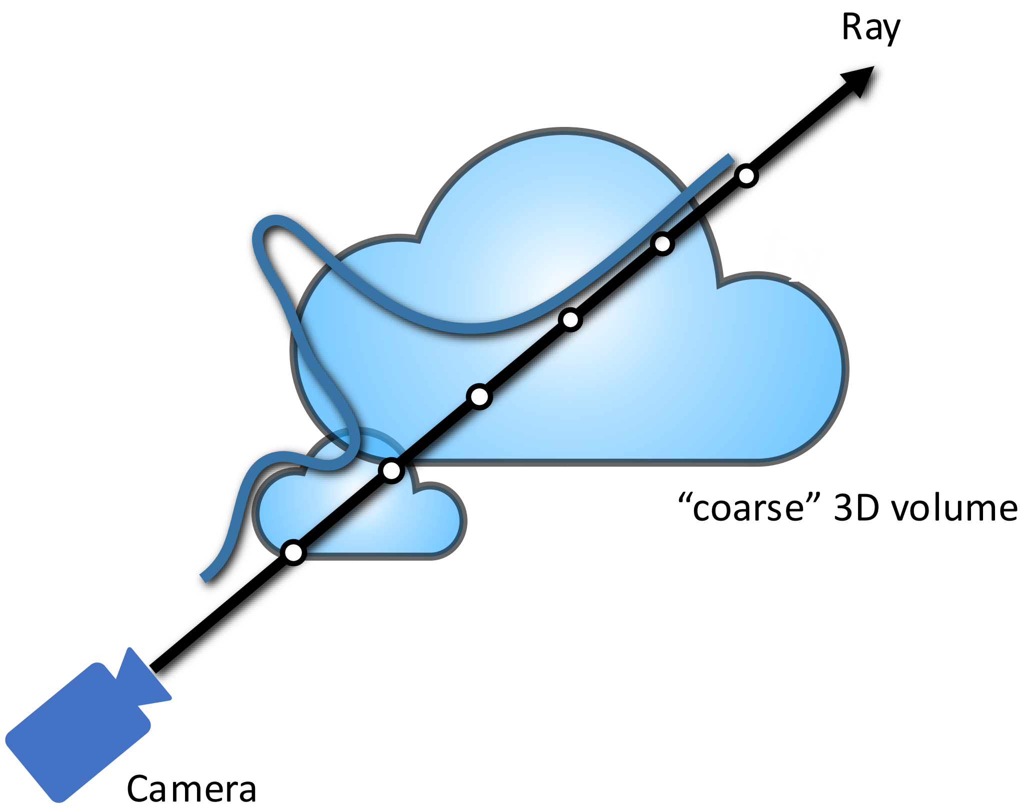


What about aliasing during coarse sampling?



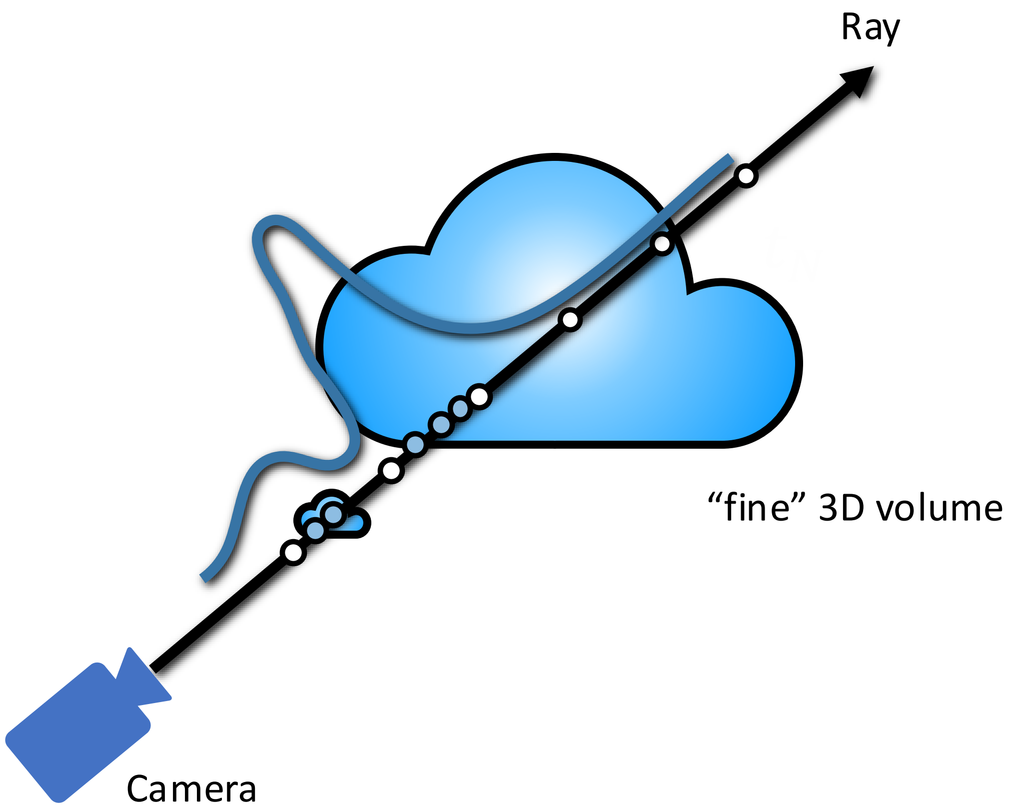
What about aliasing during coarse sampling?

Solution: train two NeRFs! —> lower resolution for first "coarse" level



What about aliasing during coarse sampling?

Solution: train two NeRFs! —> higher resolution for second “fine” level



This strategy is used in the original NeRF paper, you can implement it as an EC.

Further Reading on this Topic

- MipNeRF
  - Low-pass filter the positional encoding
- MipNeRF360
  - For 360 scenes
  - Train a "Proposal Network" instead of Coarse & Fine Networks that tells where to sample

Other Advanced Topics

Compression capability of NeRF

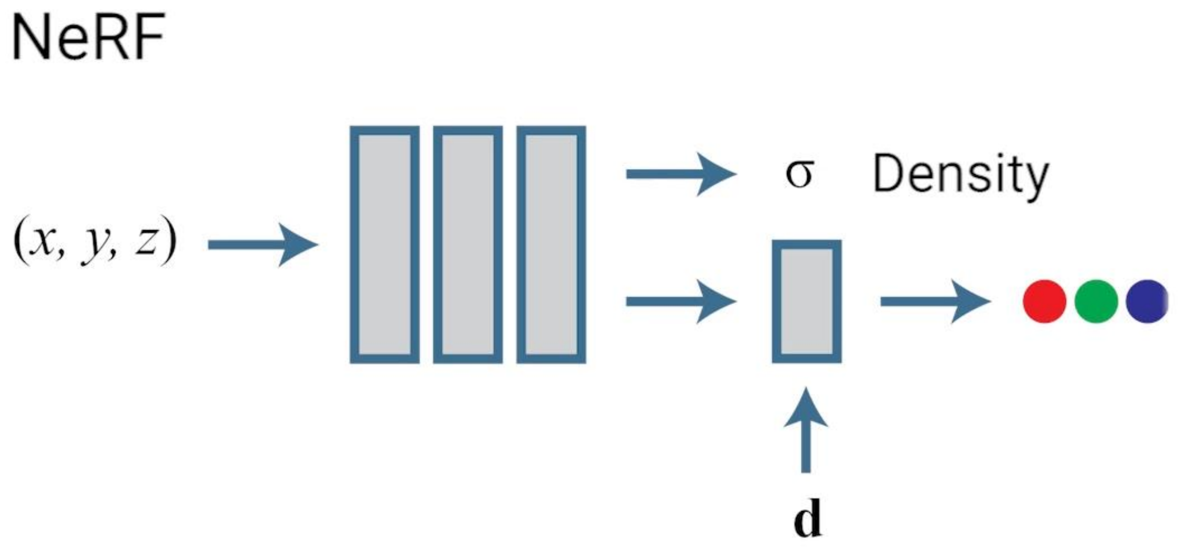
- If Image is 256x256x3, why memorize this with a MLP?
  - 3 layer MLP with 256 neurons ~ 3*256^2 learnable parameters

- In 3D:
  - if there are 100 input images: 100 * 256x256x3 = 19M
  - 8 layer MLP with 256 neurons ~ 524K
  - Just 3% of what it takes to hold 100 images

  - MLP size doesn't change even if we have 1000 input images

- Trade off?
  - SPEED!

Why is NeRF Slow?

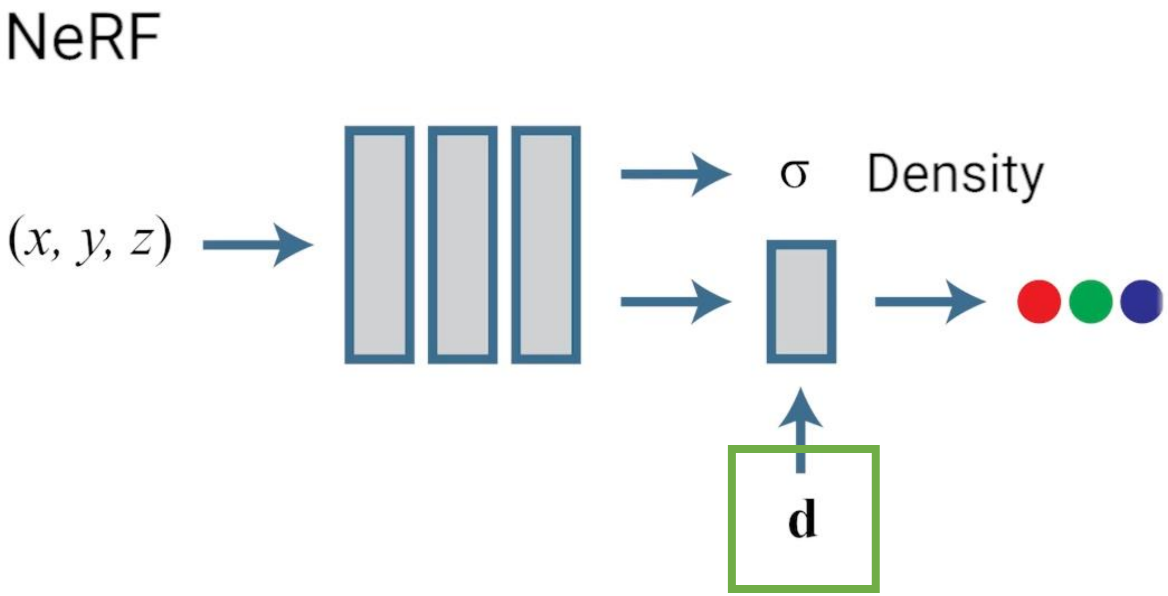



```
For each image
    For each pixel (800, 800)
        For each sample (256)
            Eval NeRF Network
```

You have to sample densely in R^5

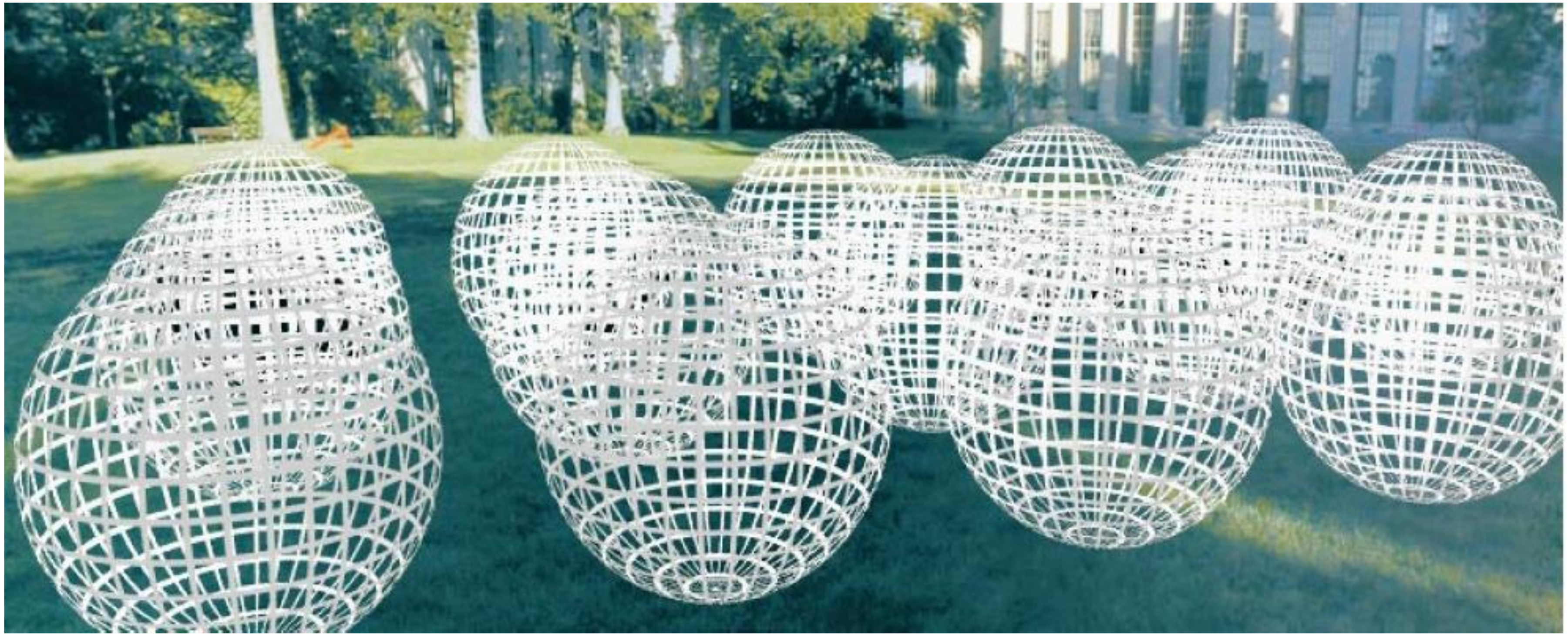
~ 163 million network calls

Cache everything?



R^5 is a lot to cache

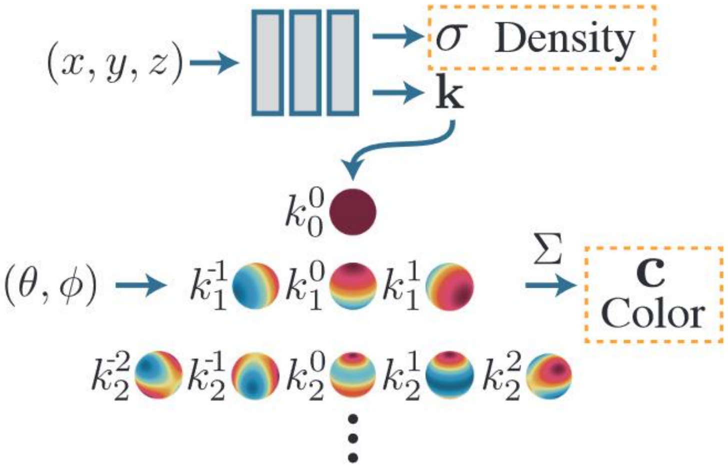
Recall:



We can factorize the location R^3 & the view direction R^2 !

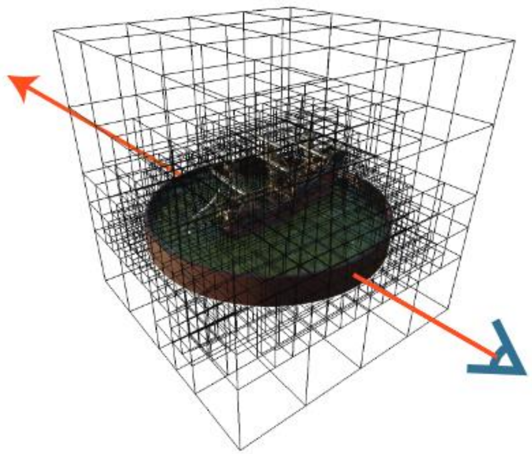
Idea

1. Factor out view-dependent effects



Spherical Harmonics

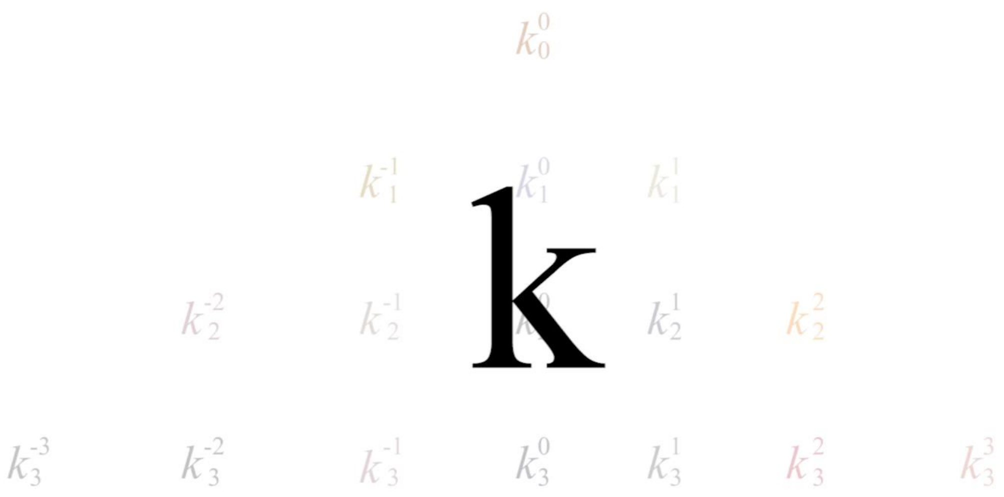
2. Store density + SH in efficient data structure (Octree)



PlenOctree =
View-dependent Octree

Spherical Harmonics to model view dependent Color

- Fourier basis on the sphere



SH models function on a sphere:

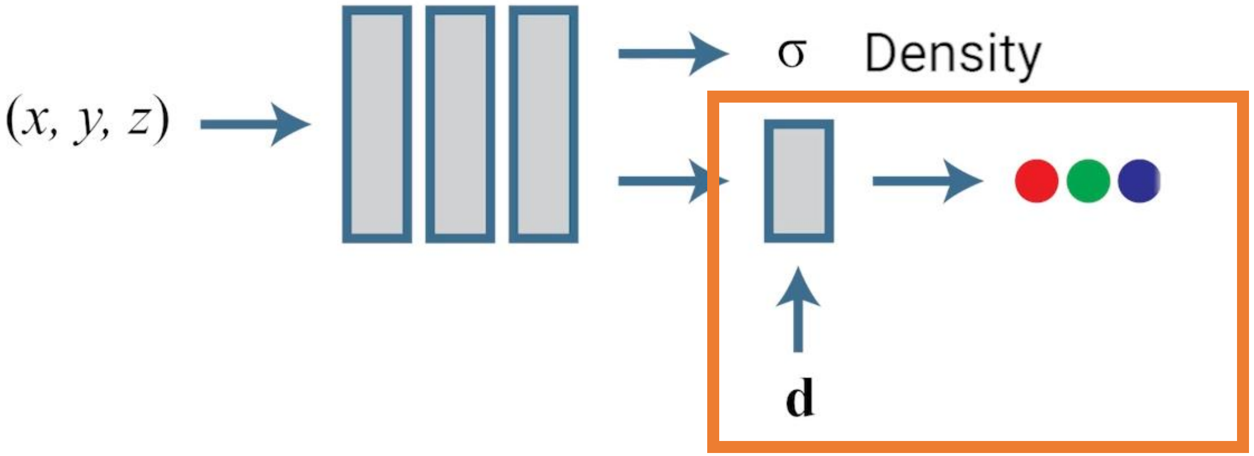
$$f(s) = \sum_{i=0}^{n^2} k_i y_i(s)$$

s – point on a unit sphere, $s = (\varphi, \theta)$
k_i - coefficient of i-th basis
$y_i(s)$ - SH basis, analytically computed, as shown on the video

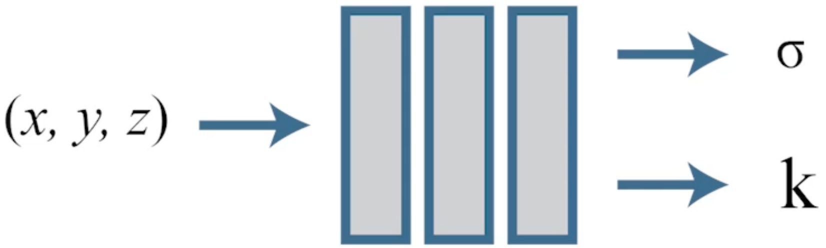
Learn k_i

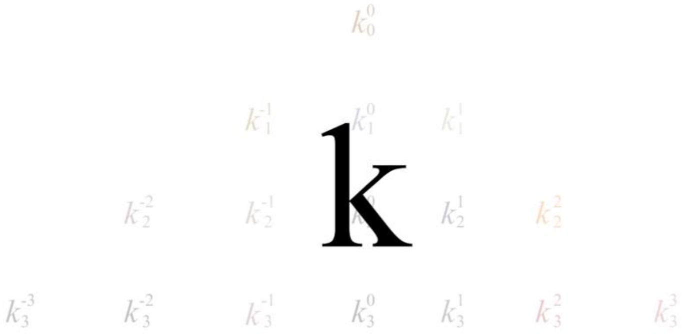
NeRF with Spherical Functions

NeRF

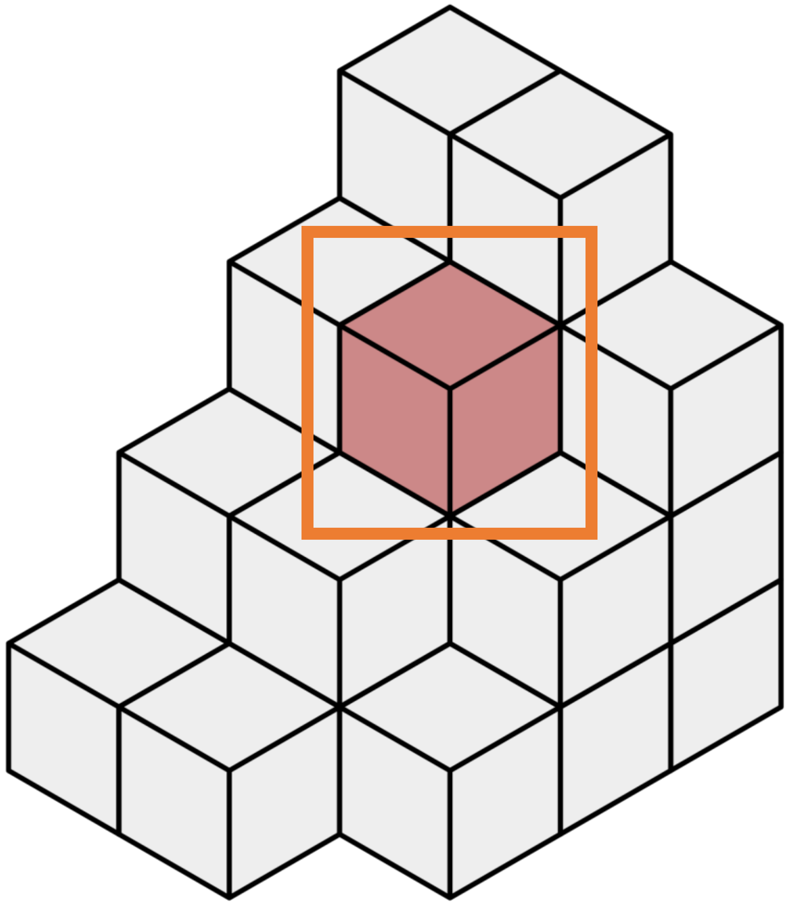


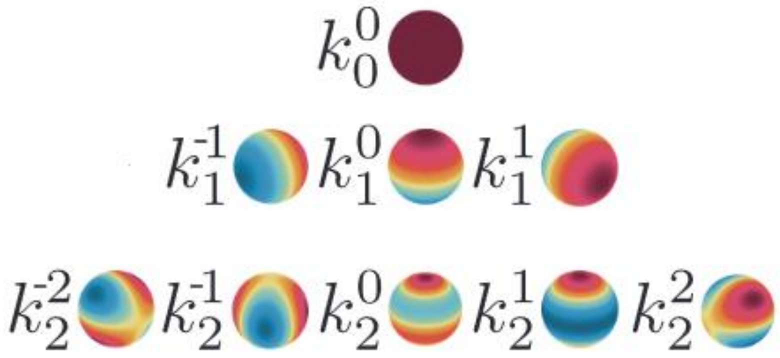
NeRF with Spherical Harmonics (NeRF-SH)





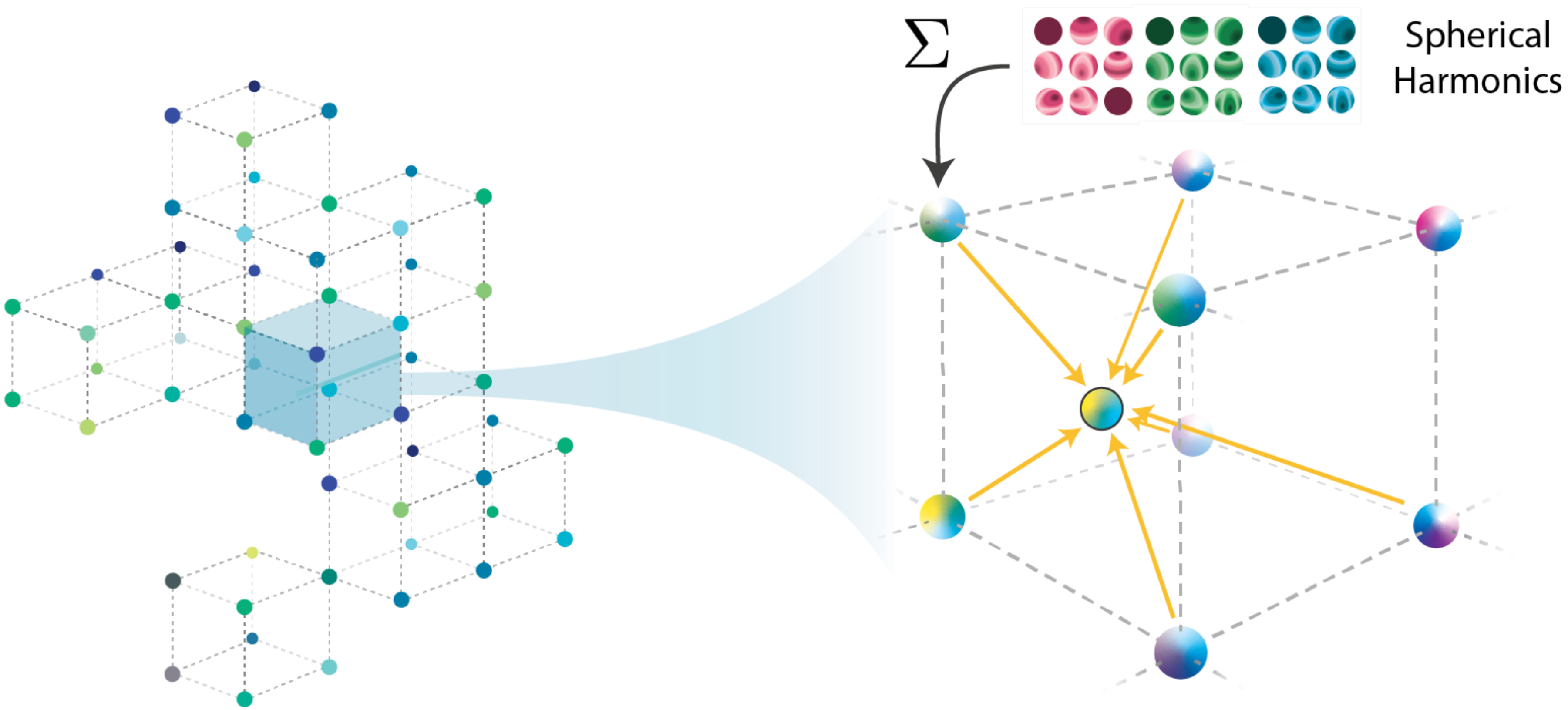
PlenOctree =
Sparse Voxels + SH coefficients





Only stores nonempty voxels

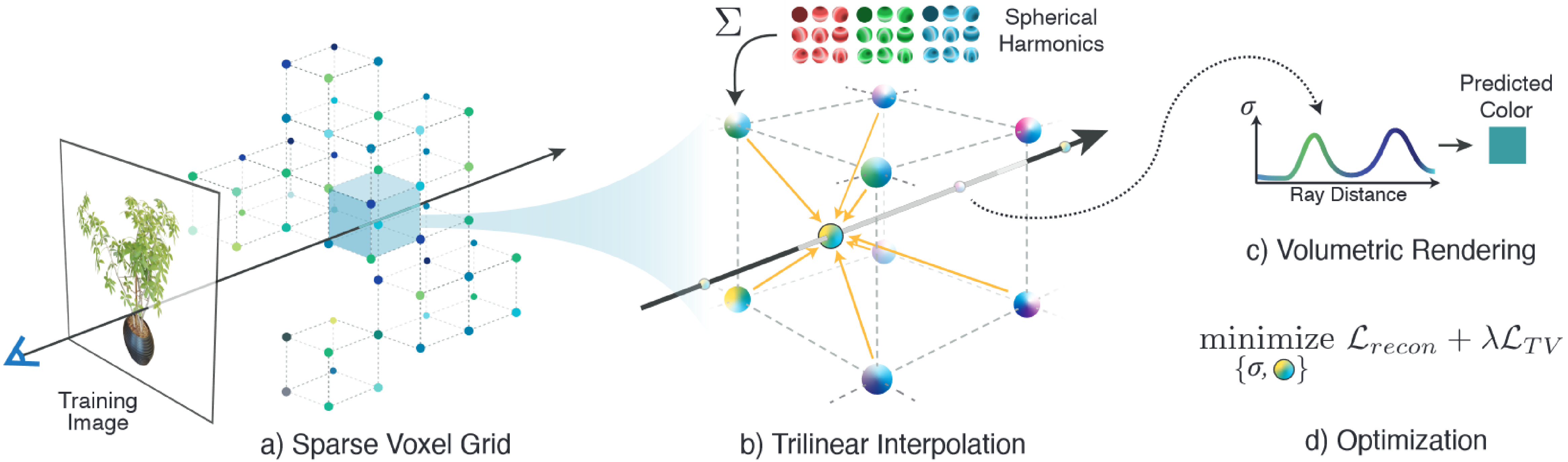
Plenoxel = "Plenoptic Volume Element"



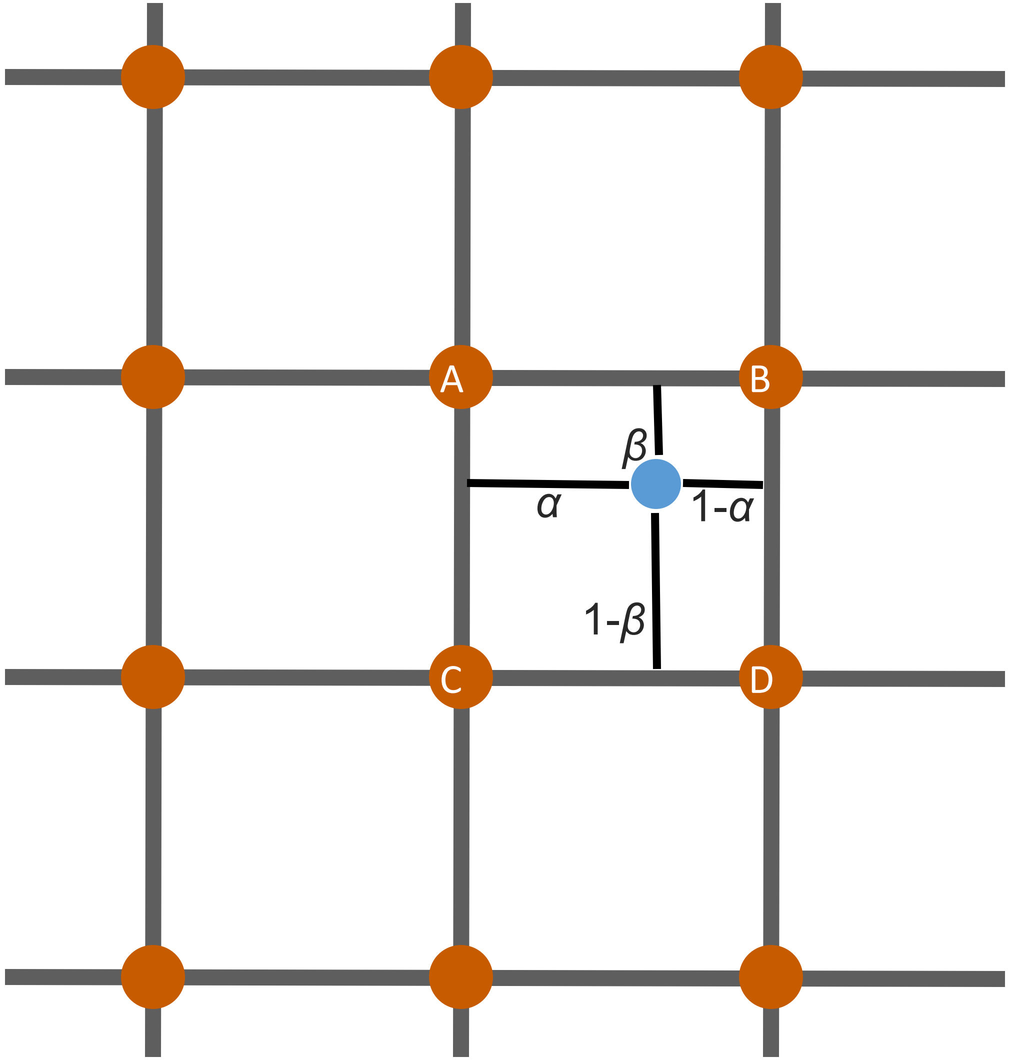
Generalization of PlenOctree – A data structure consisting of Plenoxels

Yu et al. CVPR 2022

No MLP required..



Make continuous via tri-linear interpolation

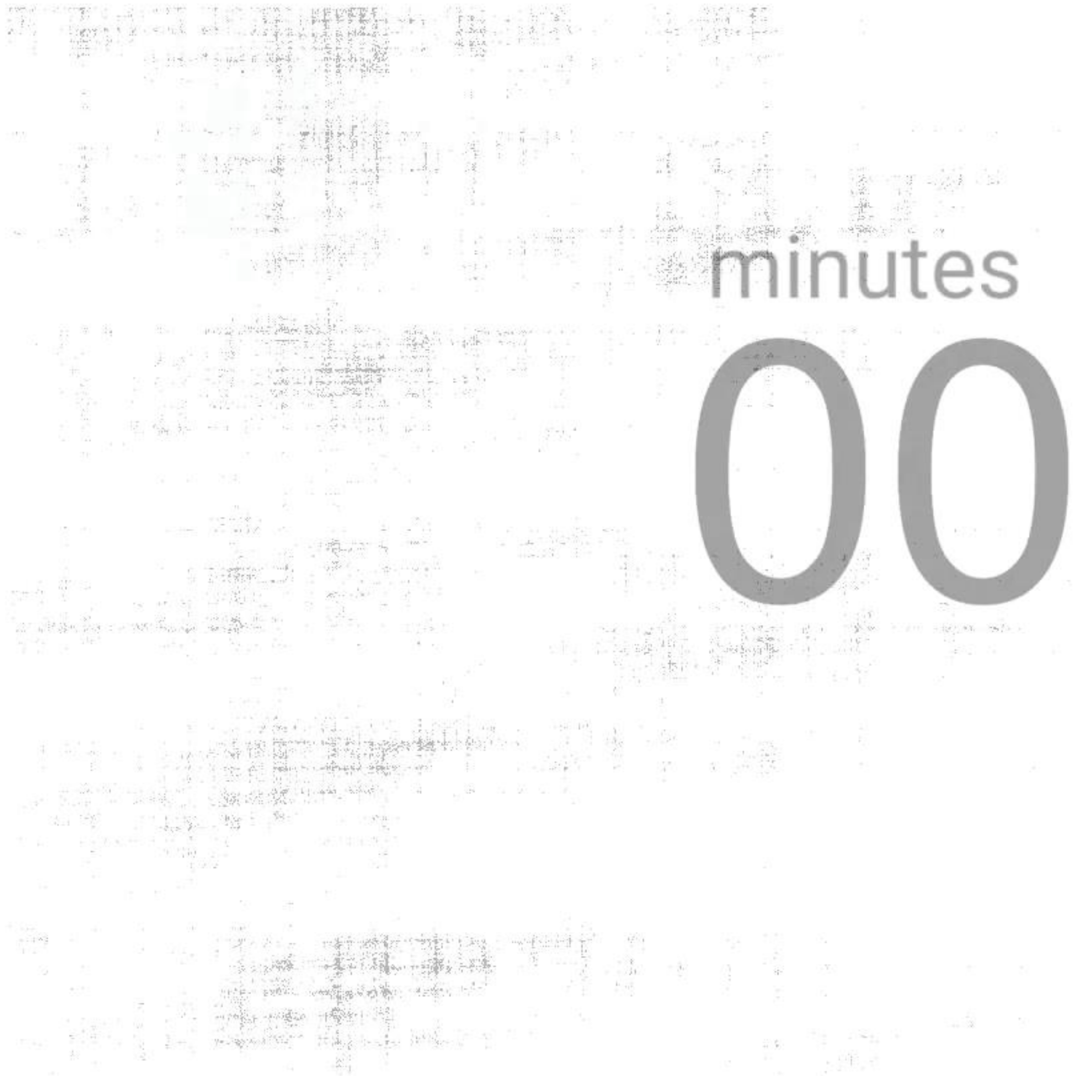


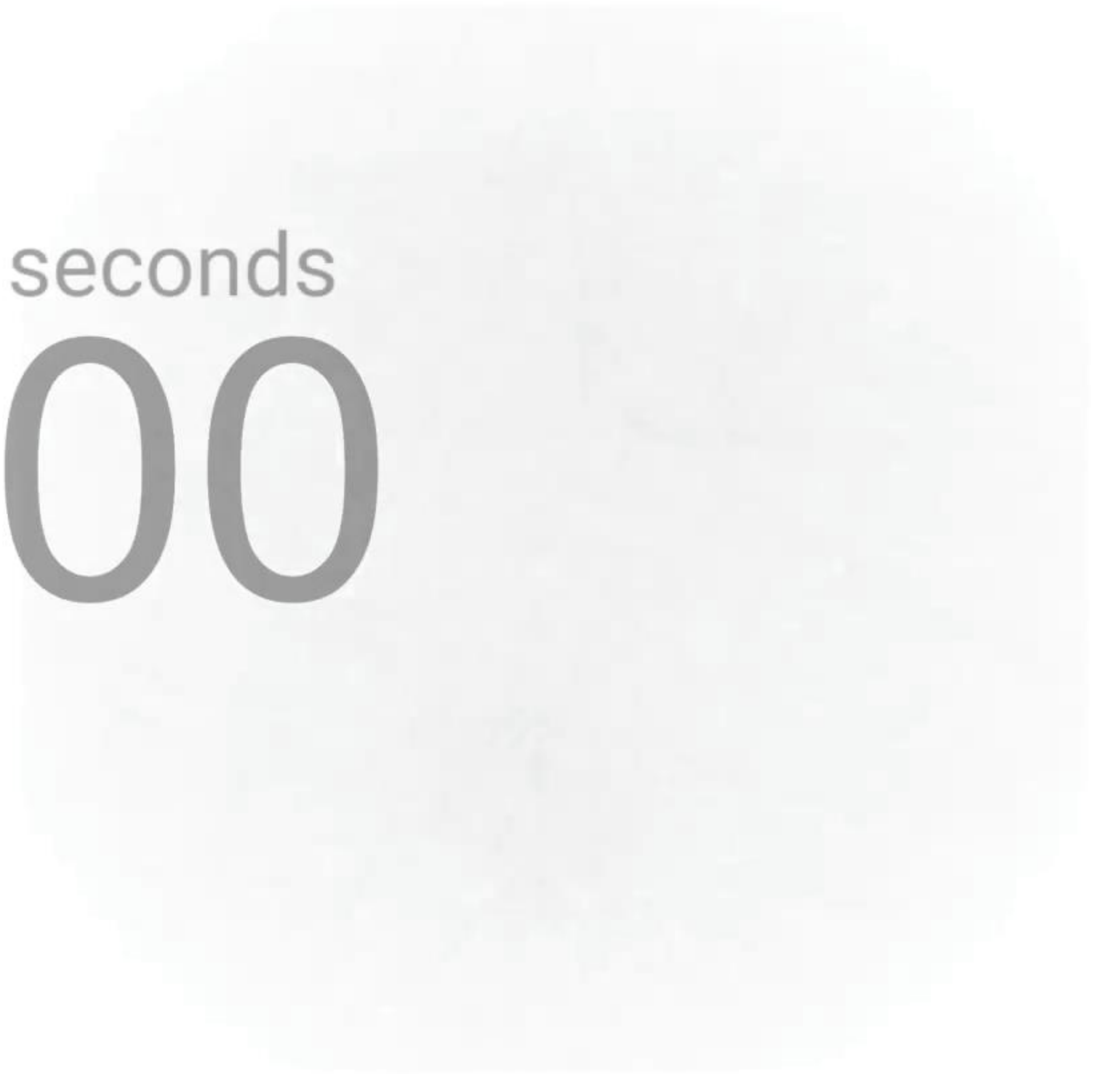
$$= \beta(\alpha A + (1 - \alpha B))$$
$$+(1 - \beta)(\alpha C + (1 - \alpha D))$$

NeRF

[Mildenhall et al. ECCV 2020]

Plenoxels

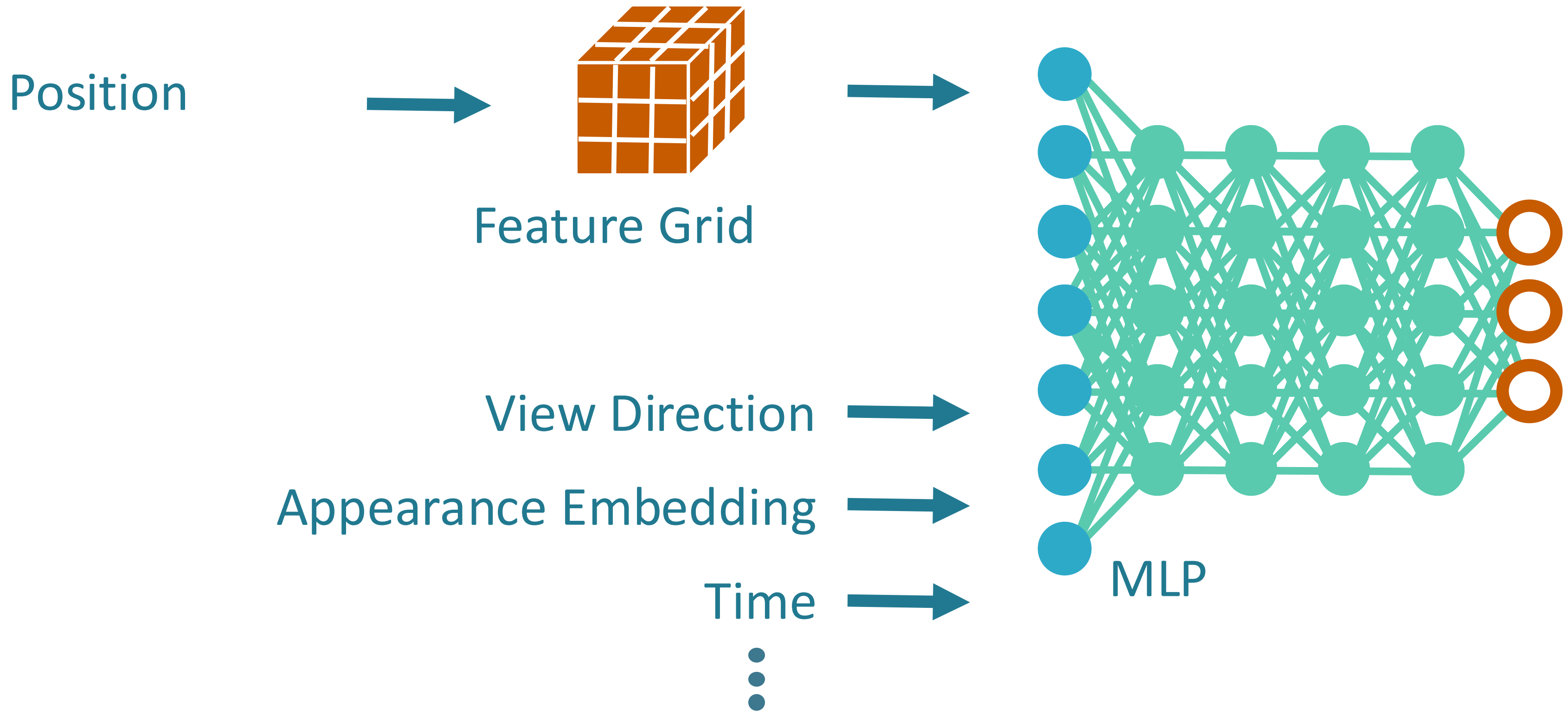




minutes seconds

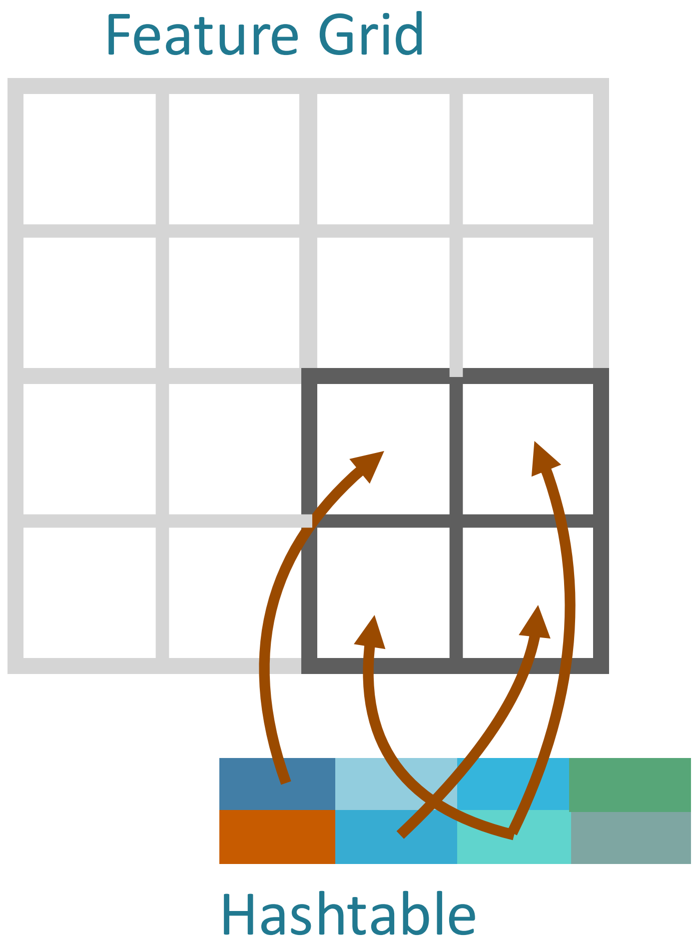
00:00

But MLPs are convenient



InstantNGP

- Hybrid approach
- Grid → hashmap → Feature retrieval
- Pass to a small MLP
- Fast and convenient!



Feature Grid > Hashmap

Mapping with collisions

Müller et al. SIGGRAPH 2022

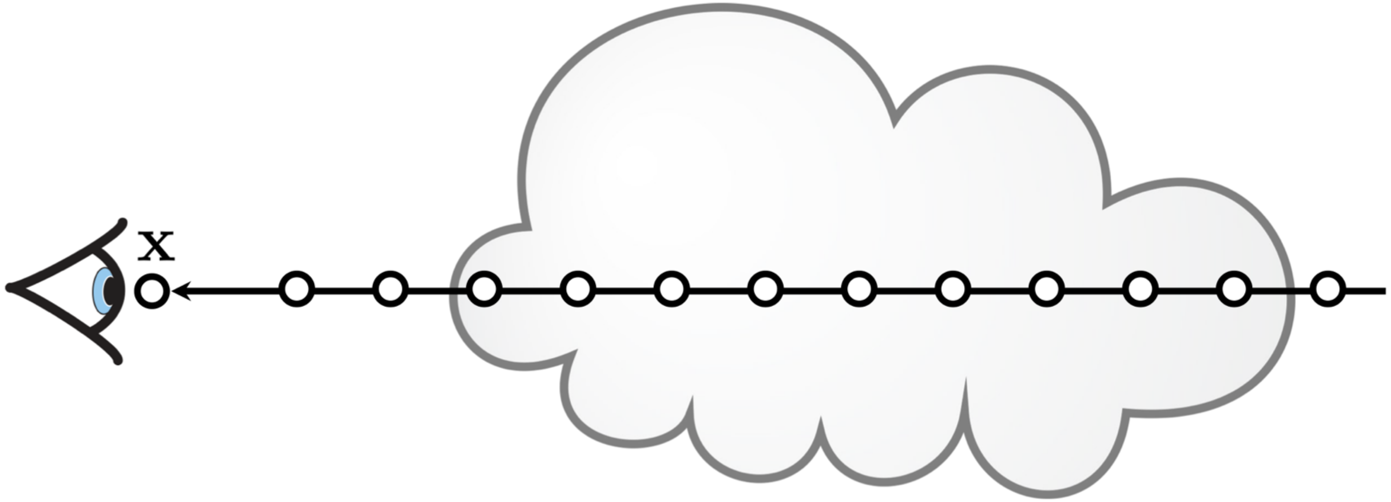
Latest: Gaussian Splatting

- Approximate with 3D Gaussian points
- Rasterize instead of volrend
- **Still alpha-blend**

- no MLP + SH like plenoxels

Kerbl and Kopanas et al. SIGGRAPH 2023

Recap: Rendering Volumes

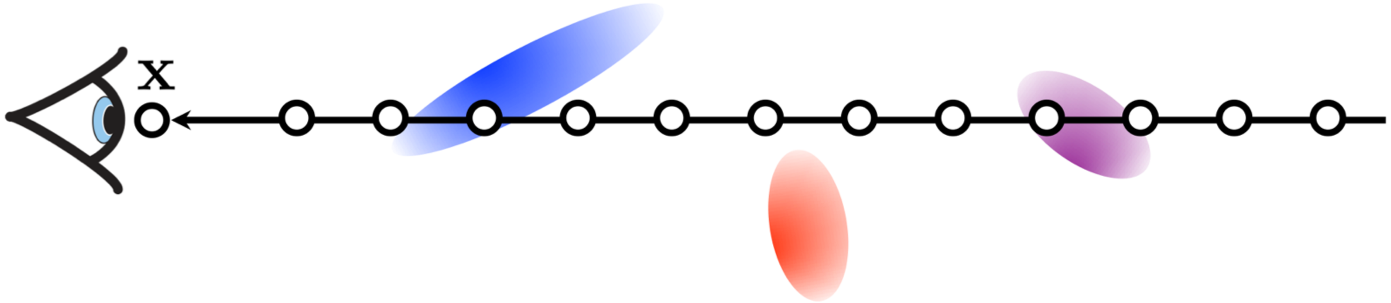


1. Draw samples along the ray

2. Aggregate their contributions to render

Slides thanks to Ioannis Gkioulekas & Shubham Tulsiani

Rendering Primitives (e.g. Gaussians)

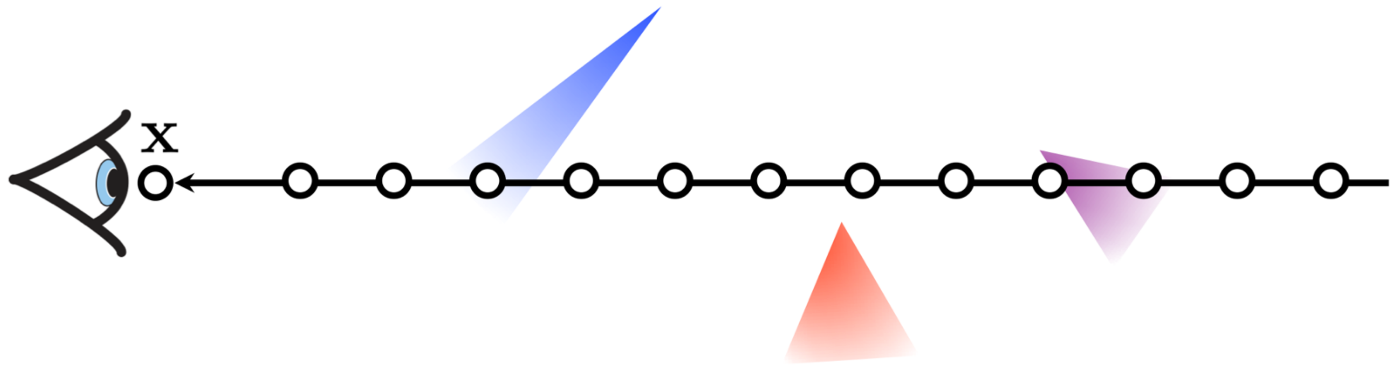


1. Draw samples along the ray

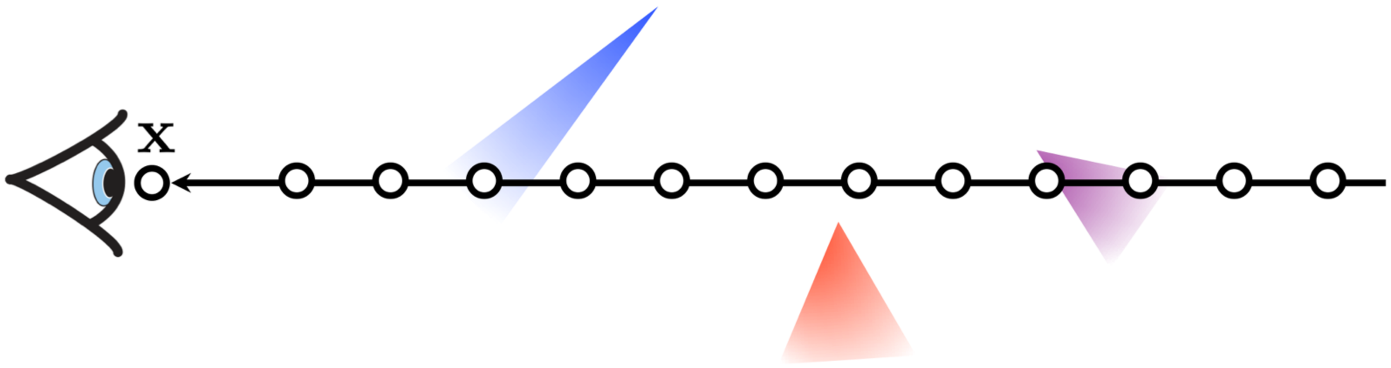
2. Aggregate their contributions to render

Slides thanks to Ioannis Gkioulekas & Shubham Tulsiani

Rendering Primitives (e.g. Triangles/Meshes)



Rendering Primitives (e.g. Triangles/Meshes)



1. ~~Draw samples along the ray~~

(wasteful — we know where the primitives are!)

1. Find primitives that affect the pixel

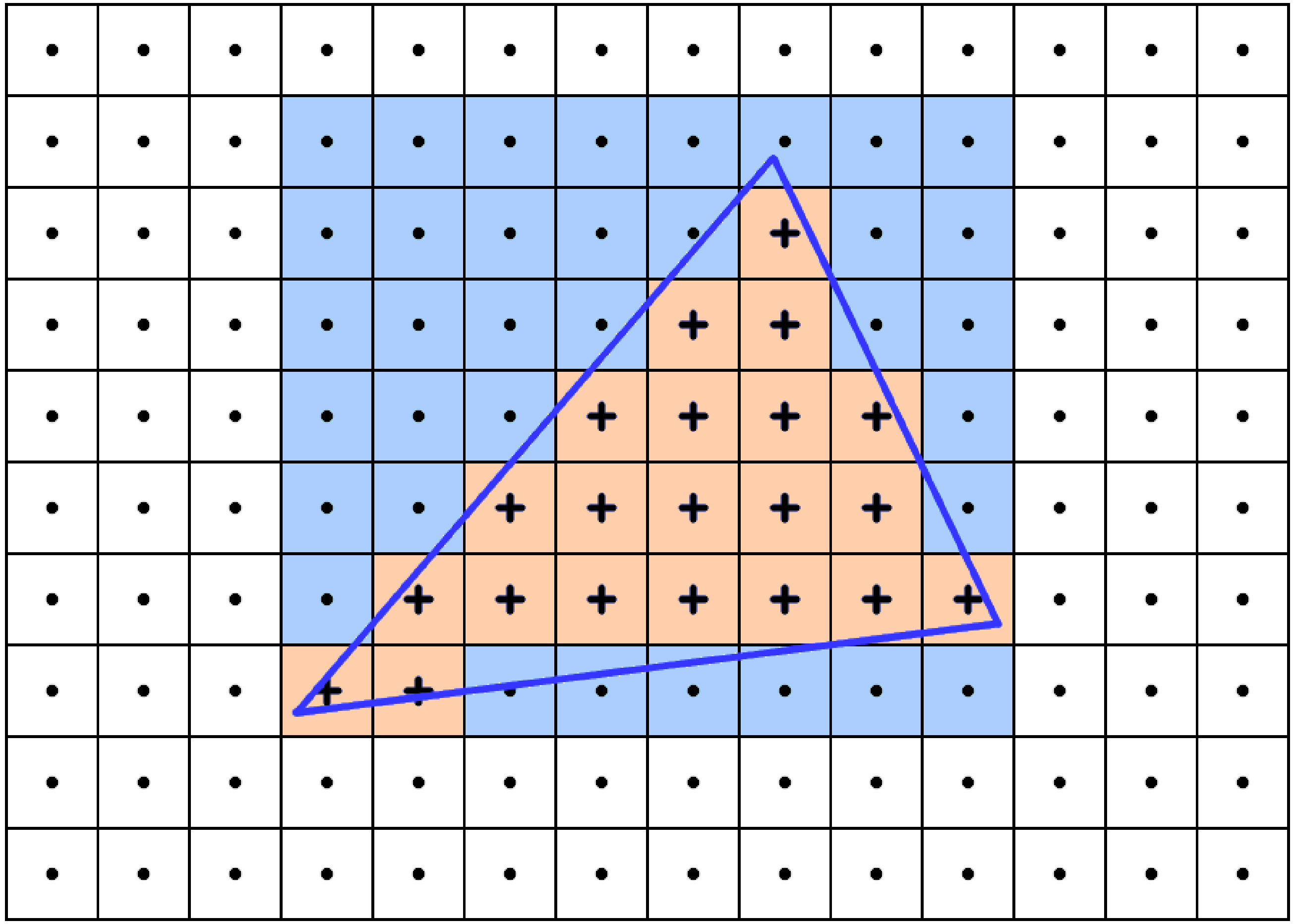
2. Aggregate their contributions to render

Rendering a mesh via Rasterization

Rasterization = conversion of primitives to pixels (details in CS184)

Rasterizat

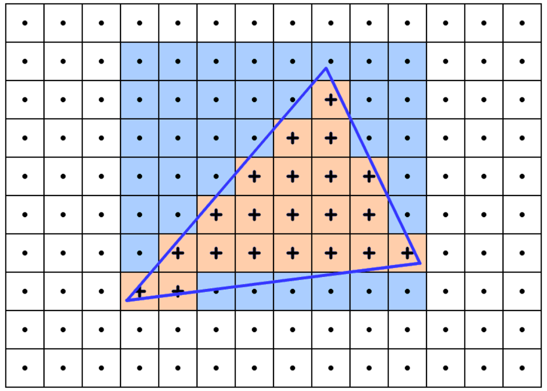
Blending (+ Sl



Slides thanks to Ioannis Gkioulekas & Shubham Tulsiani

Rendering a mesh via Rasterization

Rasterization = conversion of primitives to pixels (details in CS184)



Rasterization

Blending (+ Shading)

```
def render(mesh, camera):
    ### some structure to store K triangles
    ### for each pixel
    sorted_k_closest_img = ...

    ### Iterate over all triangles
    for triangle in mesh:
        tri_2d = project(triangle, image)
        ########
        # Update closest_K_img for pixels within tri_2d
        ########


    ### Iterate over all pixels
    for pixel in camera.grid:
        ### Iterate over triangles influencing this pixel
        for triangle in sorted_k_closest_img[pixel]:
            ########
            # Aggregate appearance
            ########
```

Differentiable Gaussian Rendering

What is the representation of a 3D Gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?

Rasterization

Blending

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

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

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

Slides thanks to Ioannis Gkioulekas & Shubham Tulsiani

Differentiable Gaussian Rendering

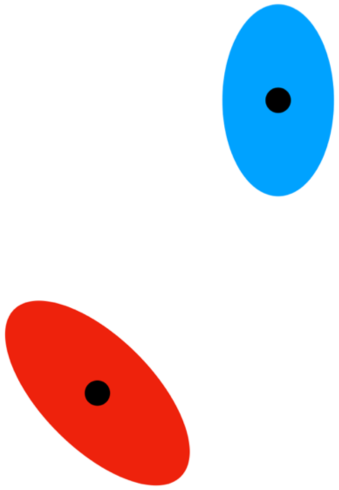
What is the representation of a 3D Gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?

Position $\mathbf{p}$

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



Slide adapted from Vincent Sitzmann.

Slides thanks to Ioannis Gkioulekas & Shubham Tulsiani

Differentiable Gaussian Rendering

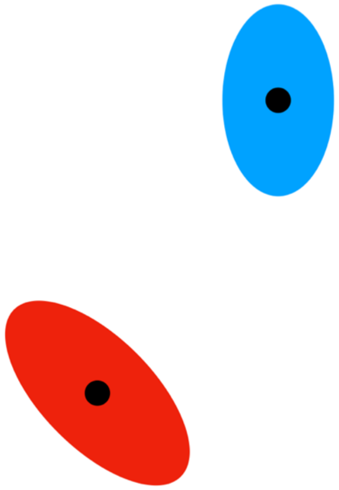
What is the representation of a 3D Gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?

Position $\mathbf{p}$

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



$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

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

$$R \in SO(3)$$

Slide adapted from Vincent Sitzmann.

Differentiable Gaussian Rendering

What is the representation of a 3D Gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?

Position $\mathbf{p}$

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

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

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

$$R \in SO(3)$$

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

Slide adapted from Vincent Sitzmann.

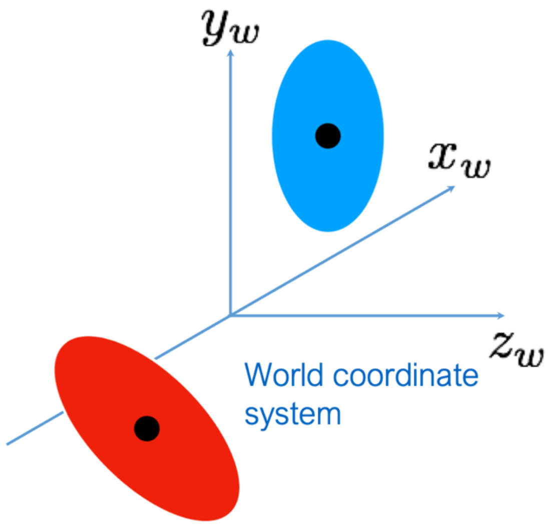
Slides thanks to Ioannis Gkioulekas & Shubham Tulsiani

Differentiable Gaussian Rendering

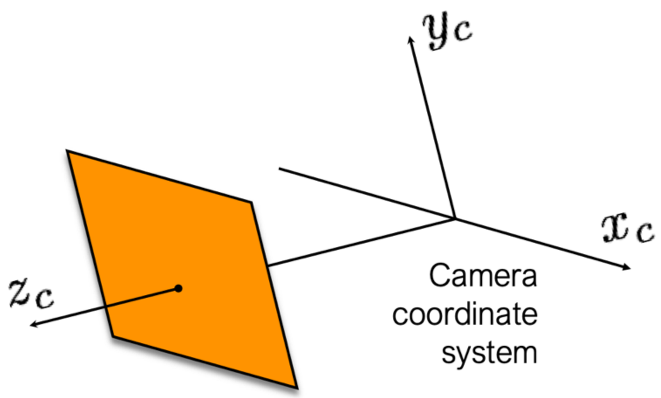
What is the representation of a 3D Gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?



$\mathbf{p}, R, S$



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

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

Slides thanks to Ioannis Gkioulekas & Shubham Tulsiani

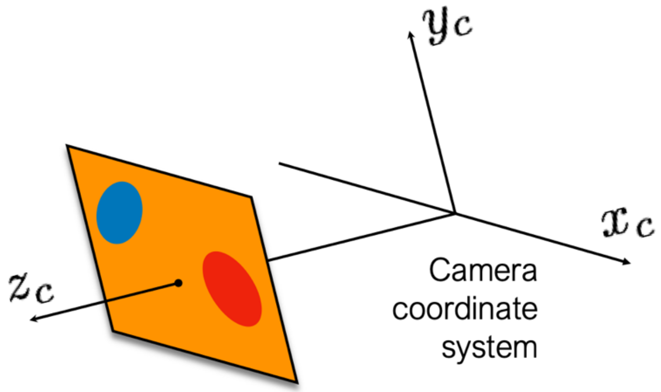
Differentiable Gaussian Rendering

$$J = \begin{bmatrix} \frac{f_x}{z} & 0 & -f_x \frac{x}{z^2} \\ 0 & \frac{f_y}{z} & -f_y \frac{y}{z^2} \end{bmatrix}$$

What is the representation of a 3D Gaussian?

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

How to project to 2D and rasterize?



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

A: Can approximate as a 2D Gaussian!

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

How to model/aggregate appearance?

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

π: Projection function for mapping 3D points to pixels

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

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

2D covariance:

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

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

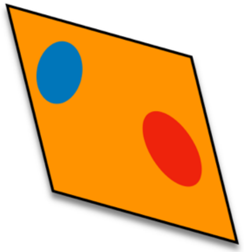
Slides thanks to Ioannis Gkioulekas & Shubham Tulsiani

Differentiable Gaussian Rendering

What is the representation of a 3D Gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?



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

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

1. Sort Gaussians from closest to furthest from the camera

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

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

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

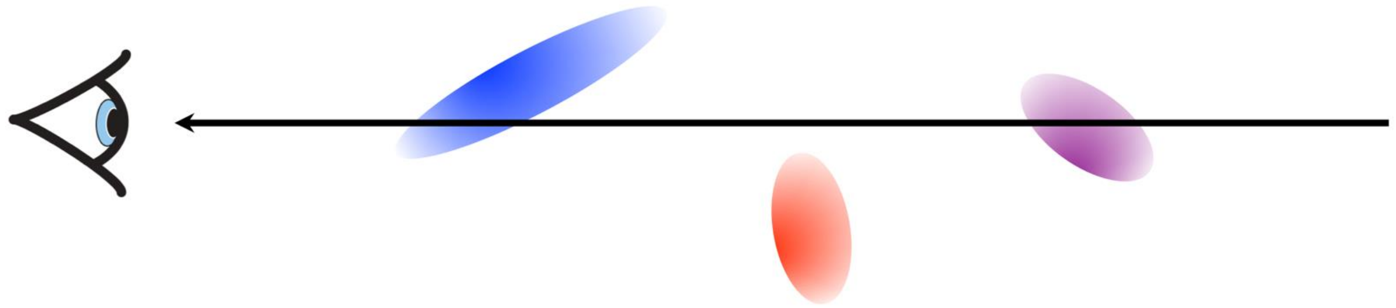
Slides thanks to Ioannis Gkioulekas & Shubham Tulsiani

Differentiable Gaussian Rendering

What is the representation of a 3D Gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?



Compute per-Gaussian weights based on opacities of current and previous Gaussians:

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

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

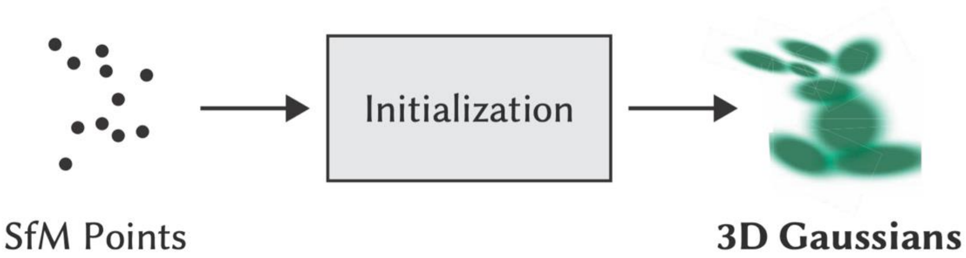
Aggregate to obtain pixel color:

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

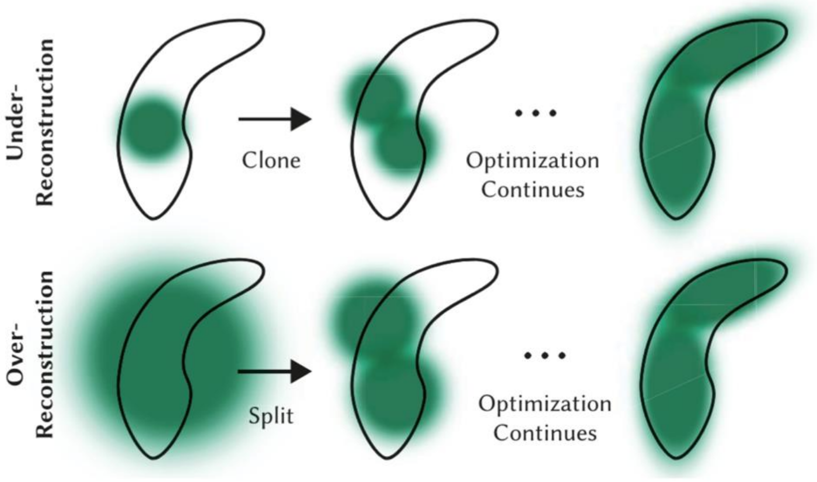
Slides thanks to Ioannis Gkioulekas & Shubham Tulsiani

Gaussian Splatting: Bells and Whistles

+ Lots of efficient GPU optimization strategies



Initialize with sparse point cloud from SfM



Split/clone Gaussians based on heuristics

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

Slides thanks to Ioannis Gkioulekas & Shubham Tulsiani

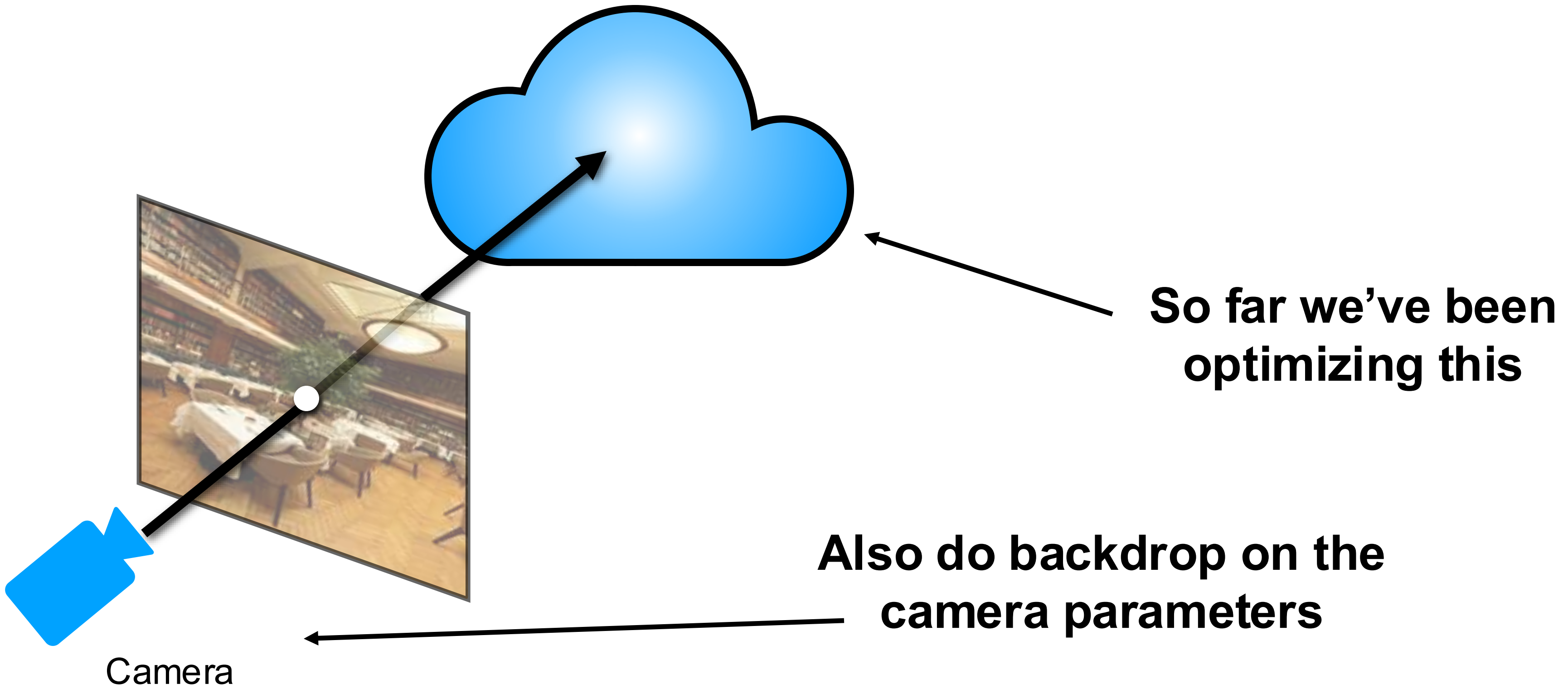
Bottom line

- Gaussian Splats still estimate volumetric rendering (same alpha-compositing, just with gaussians)
- MUCH faster because no sampling, no neural nets, rasterization
- From 30 sec / frame for 800x800 image original NeRF
  - Plenoxels/InstantNGP: 10-30FPS 1920x1080 image
  - GS: ~100-200+ FPS at 1920×1080
- Best balance of quality and speed – current status quo


Ours
MipNeRF360

Camera Quality

Small noise in the camera can be made robust by also optimizing the camera




No Pose Optimization
Block-NeRF

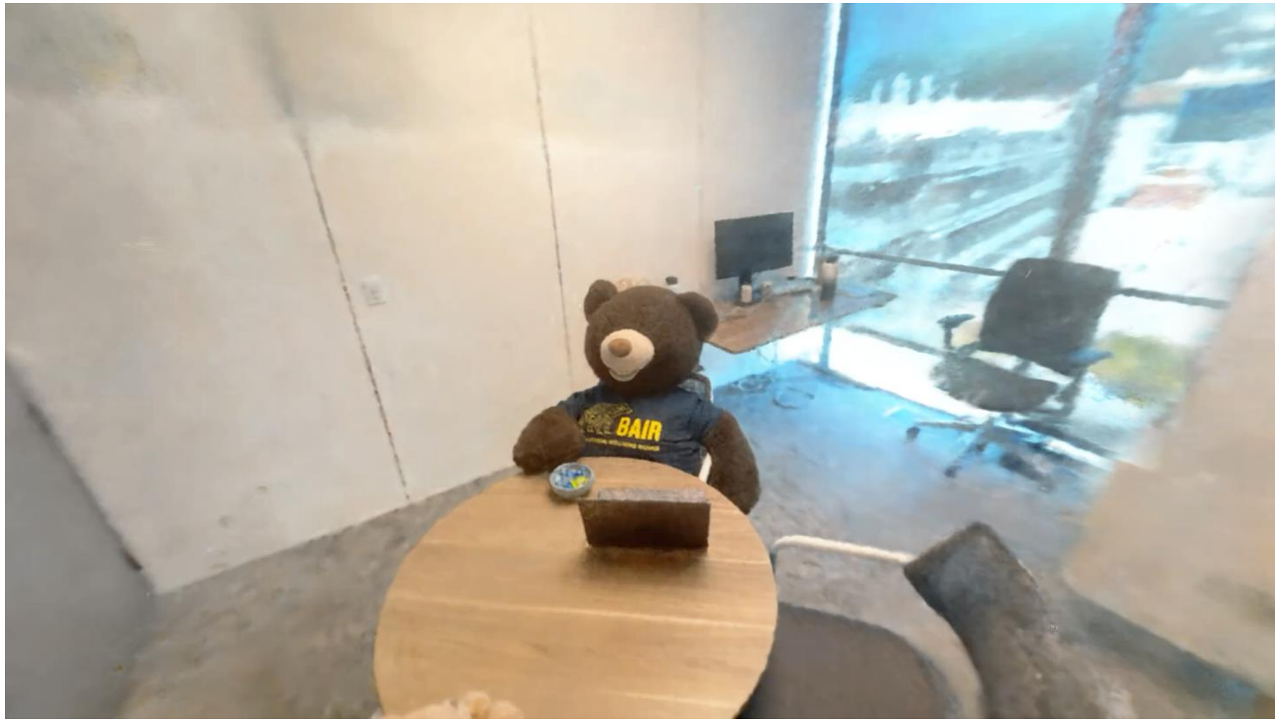
Camera Optimization

Small noise in the results can be improved

Starting from scratch is still an active area of research [Barf Lin et al. 2021, NeRF— …]

Noisy Camera from IMU/Lidar

Result with Camera Optimization



The Dynamic World

Memories of Australia –Andrew S. Hamilton

Holy grail

- Dynamic Novel View Synthesis from Monocular Camera
- Very difficult! Extremely under constrained problem

Simple baseline for adding time

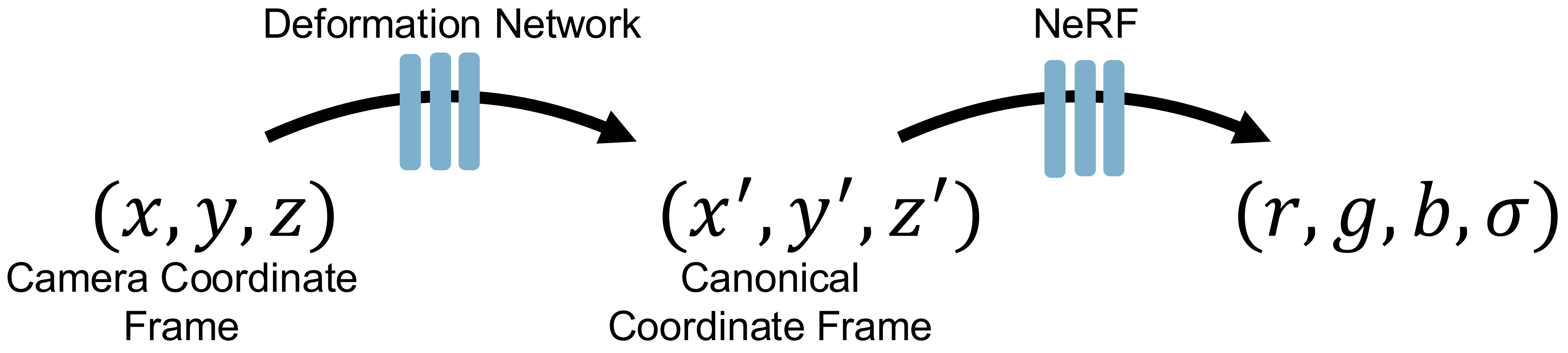
$$(x, y, z, \theta, \phi, t) \rightarrow$$

$$F_{\Omega}$$

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

Hard without simultaneous multiple view!

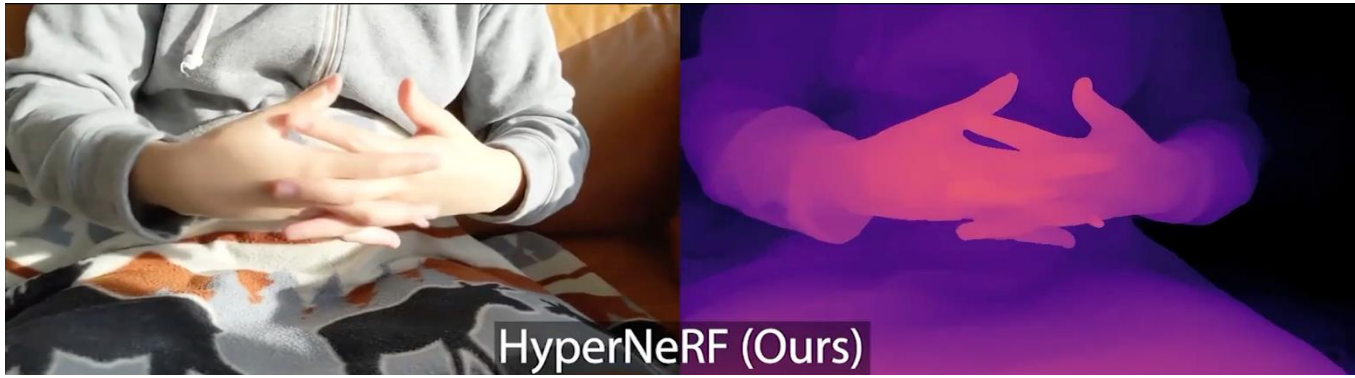
Through a deformation network



Still very under constrained

D-NeRF [Pumarola et al. CVPR 2021], Nerfies [Park et al. ICCV 2021], HyperNeRF, NR-NeRF, etc..

Dynamic View Synthesis: Monocular is hard



D-NeRF [Pumarola et al. CVPR 2021], NSFF [Li et al., CVPR 2021],HyperNeRF[Park et al. SIGASia 2021]….

- But performance on in-the-wild monocular capture still far [Gao et al. NeurIPS 2022]

What if we knew how they deform?

HMMR, Kanazawa et al.
CVPR 2019

HumanNeRF Weng et al. CVPR 2022

Other kinds of dynamic changes

Appearance Changes

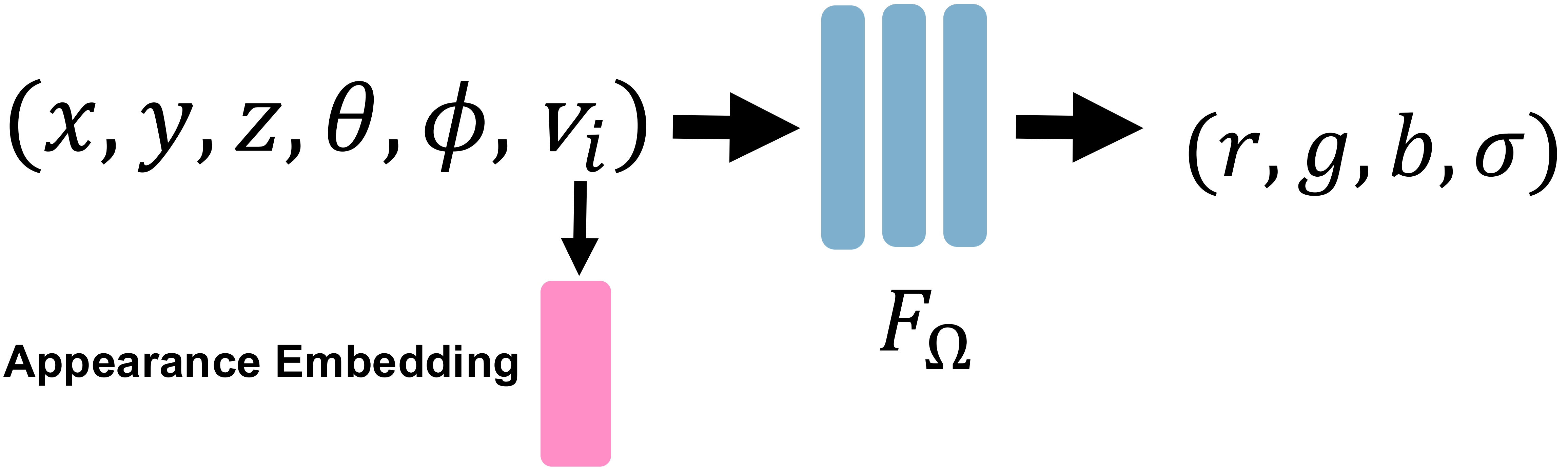
Exposure differences

Lighting changes (day, night)..

Clouds passing by..

Nerf-W [Martin-Brualla et al. CVPR 2021]

Appearance Embedding: Pretty Robust Solution

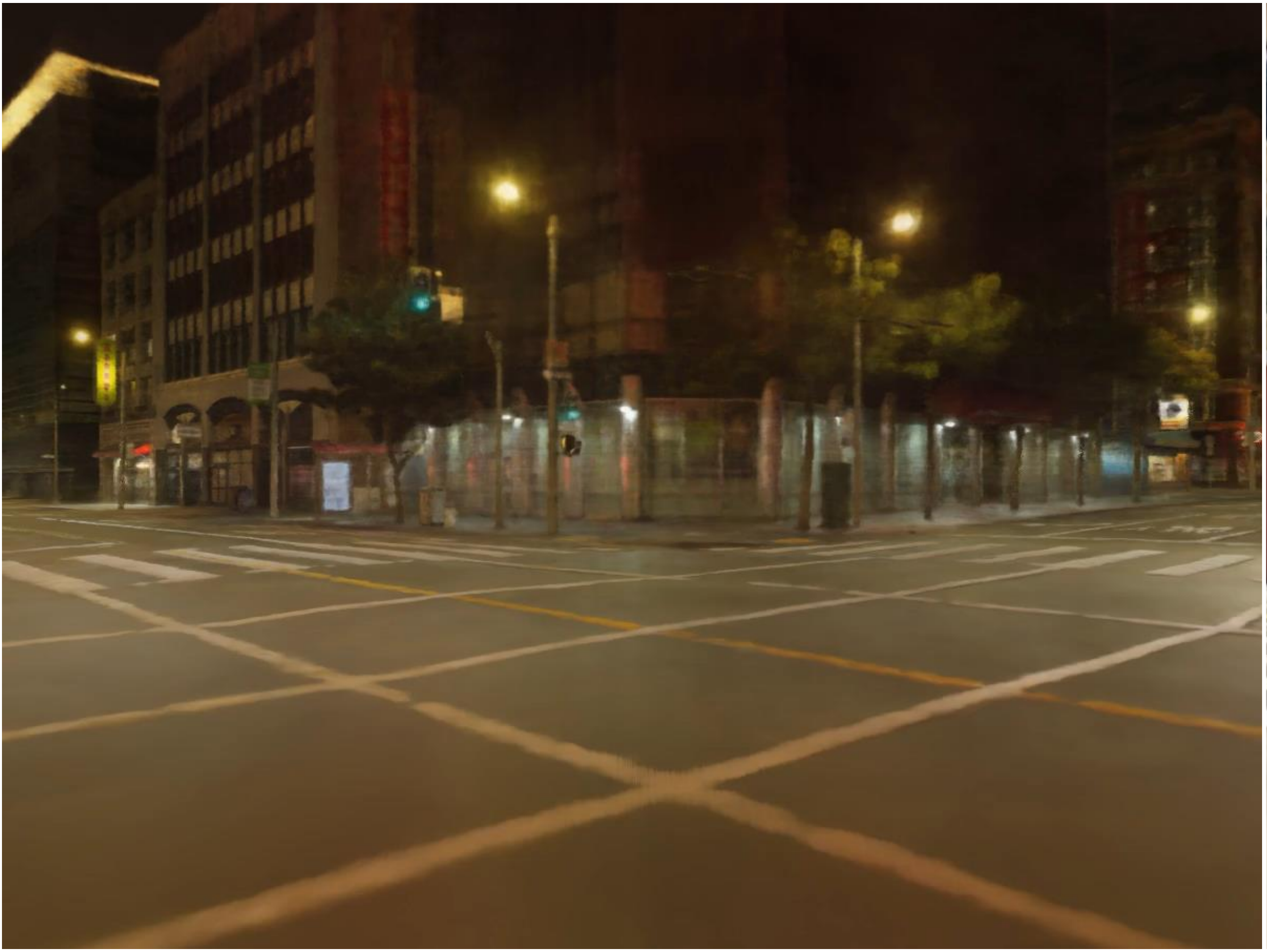


N-dim vector
Optimized *per* image: “Auto-Decoding”
ie GLO: Generative Latent Optimization [Bojanowski et al. ICML 2018]

Nerf-W [Martin-Brualla et al. CVPR 2021]

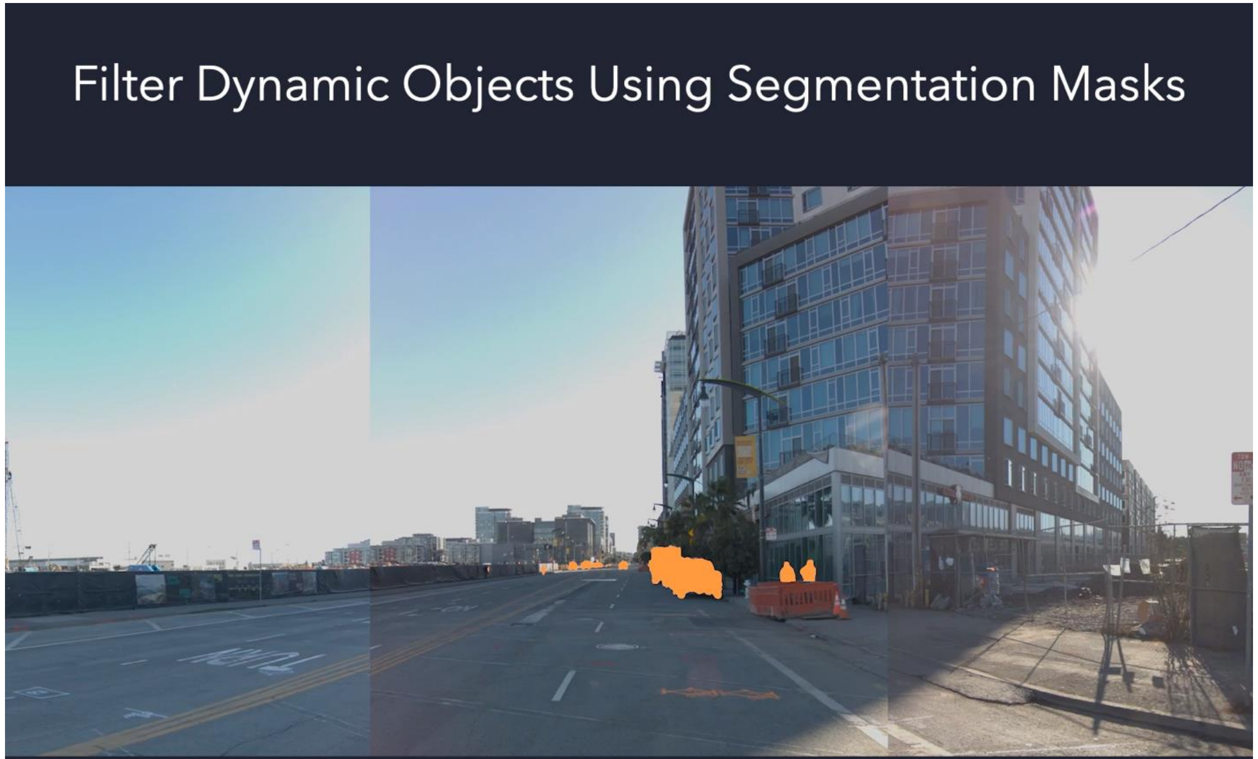
Appearance Changes

Appearance Encoding is Effective



Transient objects

- Happens all the time! People moving around, interacting with the world
- Difficult! Problem of Grouping
  - how do you know which part is connected or
  - Can use two NeRFs, one global, one per-image, but this often leads to degenerate solutions



- Current solution: Ignore (mask out)

Why is dynamic scenes hard?

- Unless you have a light dome
- Essentially you only have a single-view

Building & Reusing Prior Knowledge

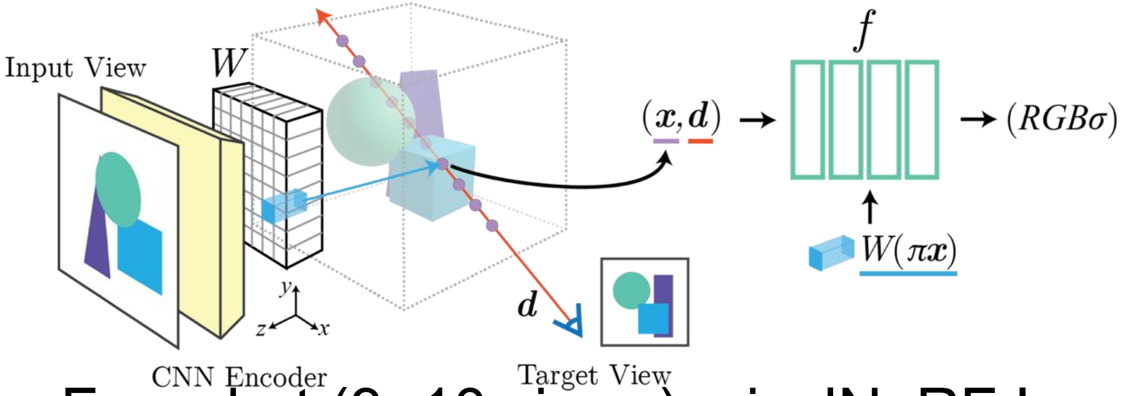
Machine Learning

NeRF is per-scene optimization

- We need lots of images to get good view synthesis!!
- Also there’s no knowledge reused from prior scene reconstructions
- How to bring learning in the picture?

Few-shot NeRF

- One-shot (single-view): pixelNeRF [Yu et al. CVPR'19]



- Few-shot (3~10 views): pixelNeRF,IBRNet [Wang et al. CVPR'21], MVSNet [Chen et al. ICCV'21], etc…

- Challenging for predicting completely unseen real scenes

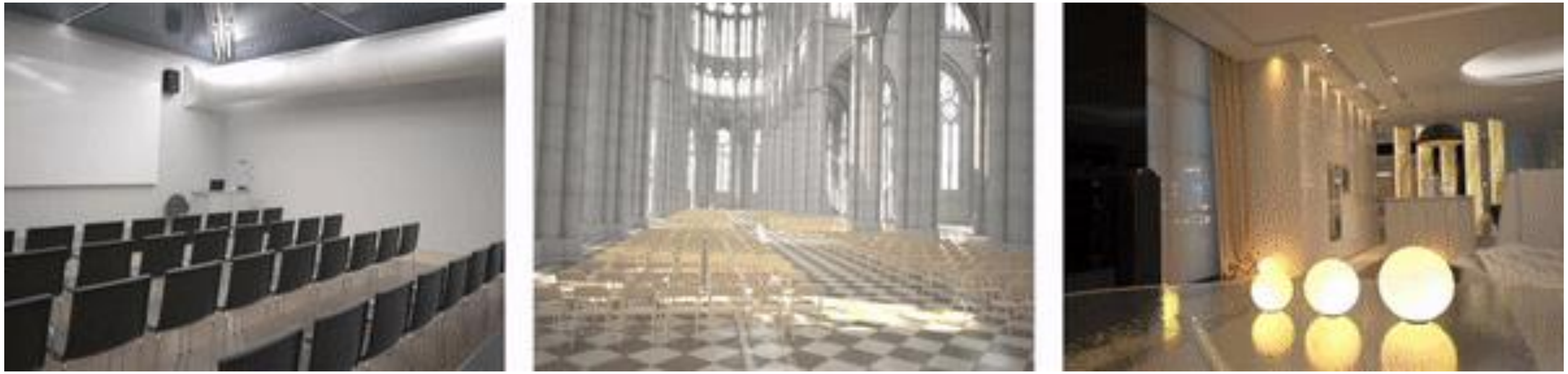
- How to deal with the multi-modal nature of the problem??

IBRNet

Data is the bottleneck

- Large-scale Real-World Multi-view Data is hard to collect:
  CO3D [Reizenstein ICCV 2021]

- A lot to learn from other single-view 3D prediction models:



Gkioxari et al. CVPR 2022

Time Line

NeRF
paper release (arXiv)
March 2020

Real-Time Rendering
PlenOctrees, FastNeRF, SNeRG, etc..
March 2021

Fast Training
Plenoxels, Instant-NGP etc..
March 2022

City-scale
BlockNeRF..

3D Generation
DreamFusion
March 2023

Quality & Speed
Gaussian Splatting
March 2024

Mildenhall*, Srinivasan*, and Tancik* et al. ECCV 2020.

Yu et al. ICCV 2021

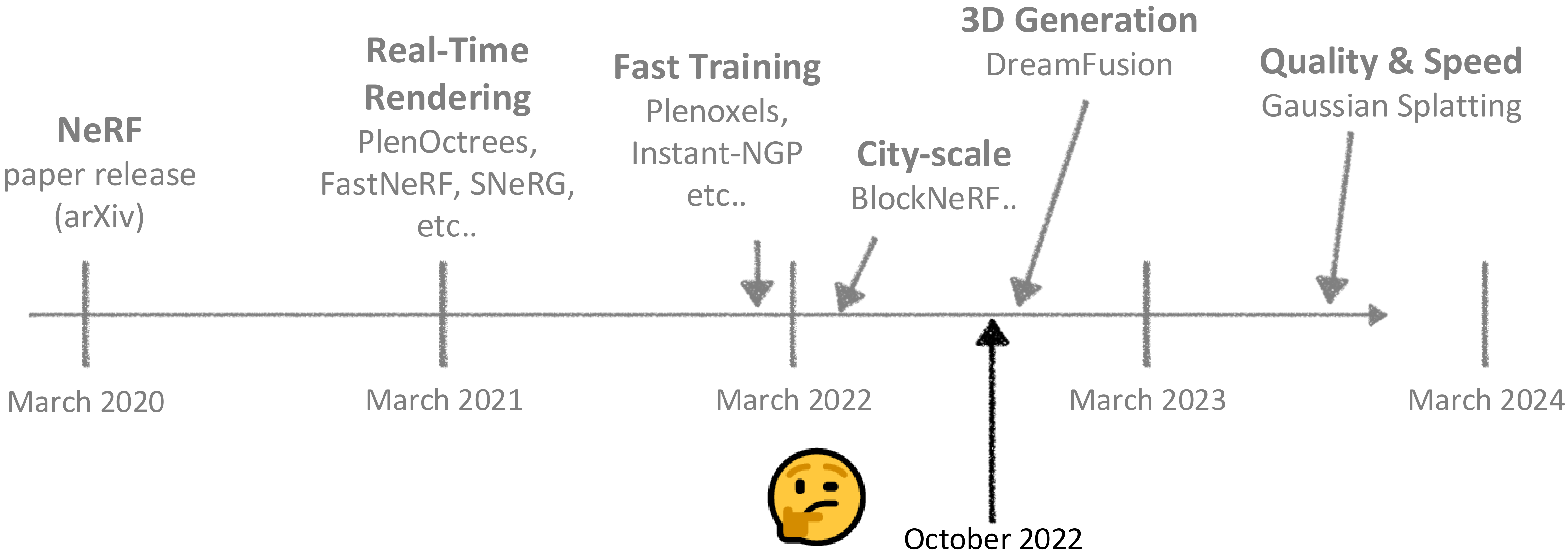
Tancik et al. CVPR 2022

Yu* and Fridovich* et al. CVPR 2022

Poole et al. ICLR 2023

Kerbl* and Kopanas* et al. SIGGRAPH 2023

Time Line



nerfstudio

A Modular Framework for NeRF Development

Matthew Tancik*, Ethan Weber*, Evonne Ng*, Ruilong Li, Brent Yi, Justin Kerr,
Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja,
David McAllister, Angjoo Kanazawa

+143 additional Github collaborators

SIGGRAPH 2023

Design Goals

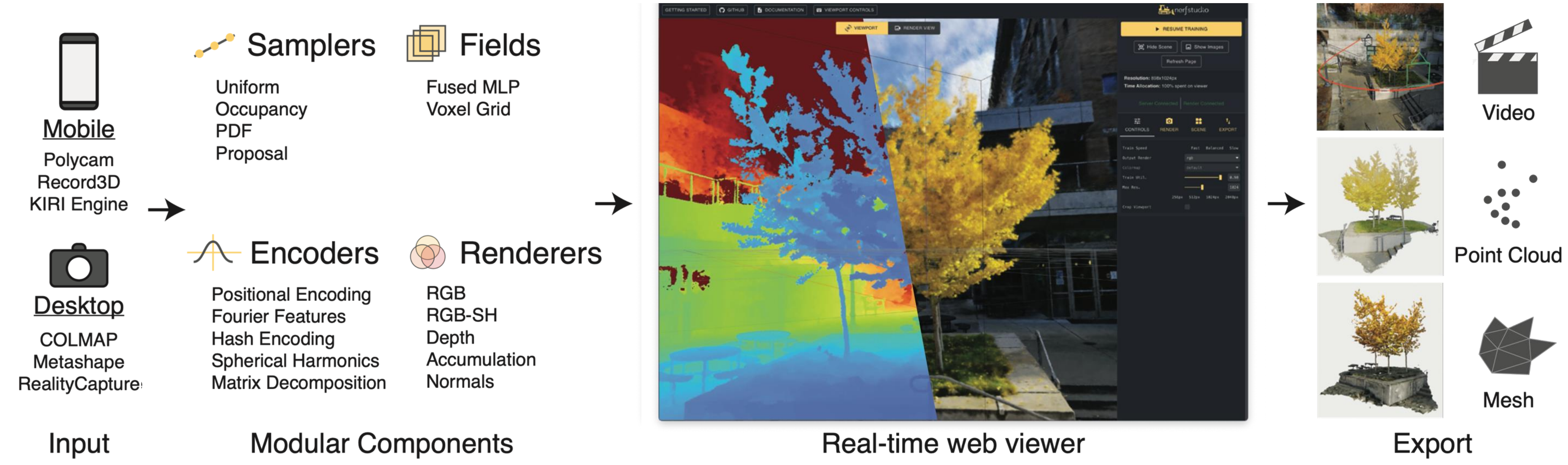
Easy to:

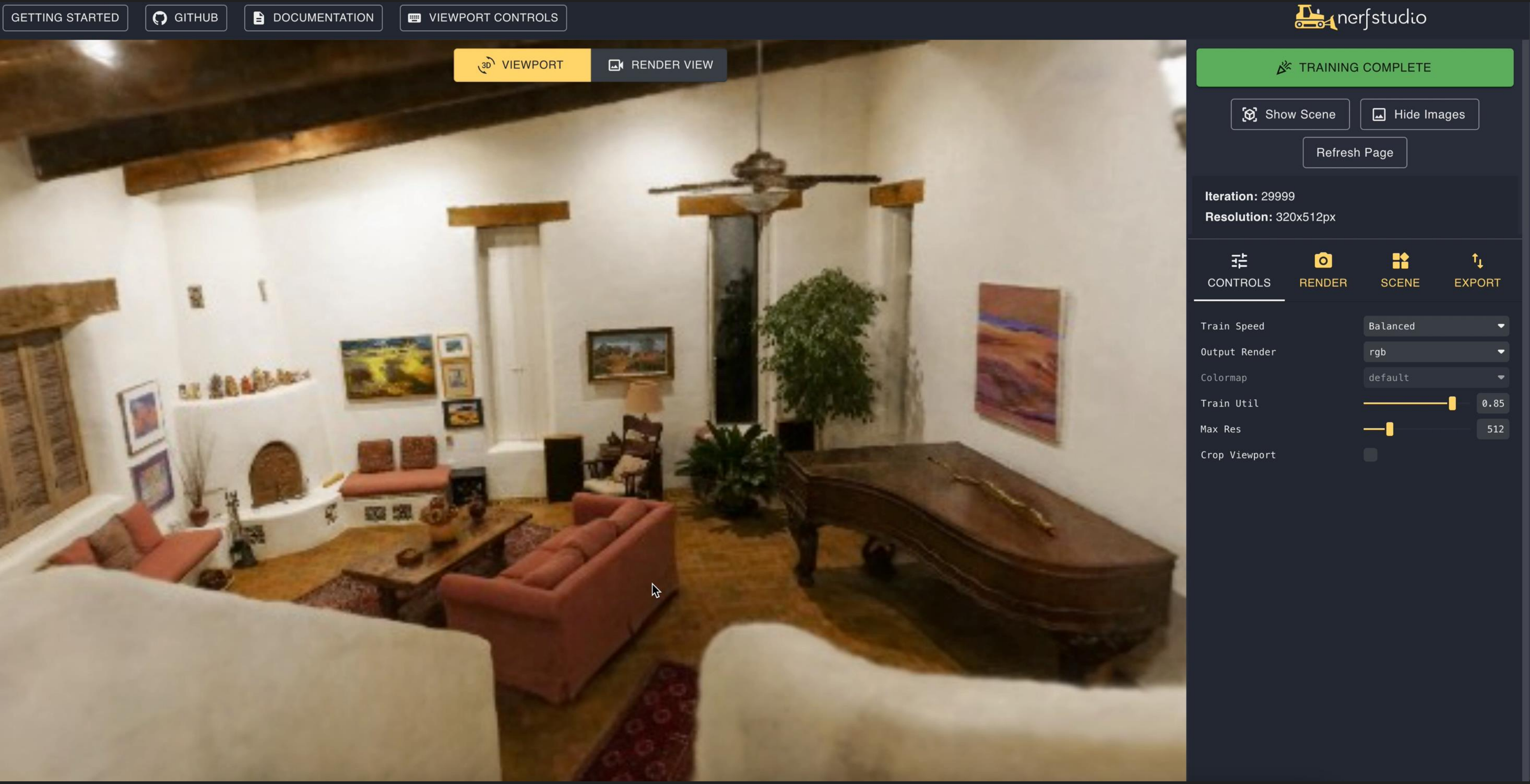
Use

Develop

Learn

An End-to-End Framework




GETTING STARTED
GITHUB
DOCUMENTATION
VIEWPORT CONTROLS
nerfstudio
VIEWPORT
RENDER VIEW
TRAINING COMPLETE
Show Scene
Hide Images
Refresh Page
Iteration: 29999
Resolution: 320x512px
CONTROLS
RENDER
SCENE
EXPORT
Train Speed
Balanced
Output Render
rgb
Colormap
default
Train Util
0.85
Max Res
512
Crop Viewport

Data Pipelines

Onboarding Pipelines

- COLMAP
- Polycam
- Record3D
- MetaShape
- RealityCapture
- Kiri Engine

Easy to Develop

Sampling

Fields & Encoders

Volumetric Rendering

Pythonic and Modular

Easy to Develop

Sampling

- Uniform
- Occupancy
- PDF
- Proposal
- Spacing Fn

Fields & Encoders

- Positional Encoding
- Fourier Features
- Hash Encoding
- Spherical Harmonics
- Matrix Decomposition

- Fused MLP
- Voxel Grid

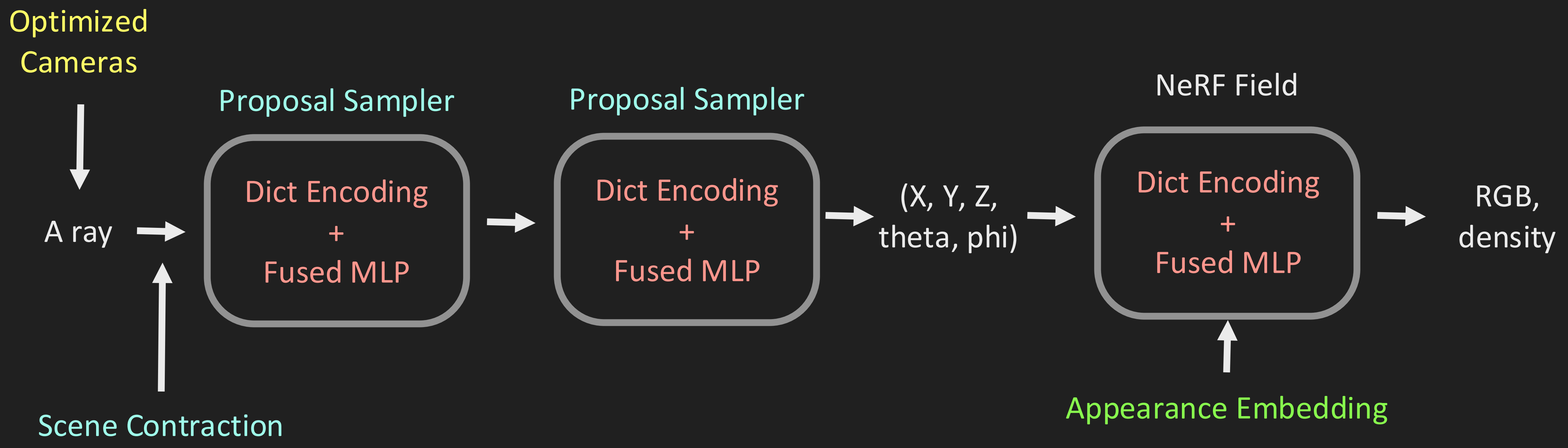
Volumetric Rendering

- RGB
- RGB-SH
- Depth
- Accumulation
- Normals

Pythonic and Modular

Nerfacto

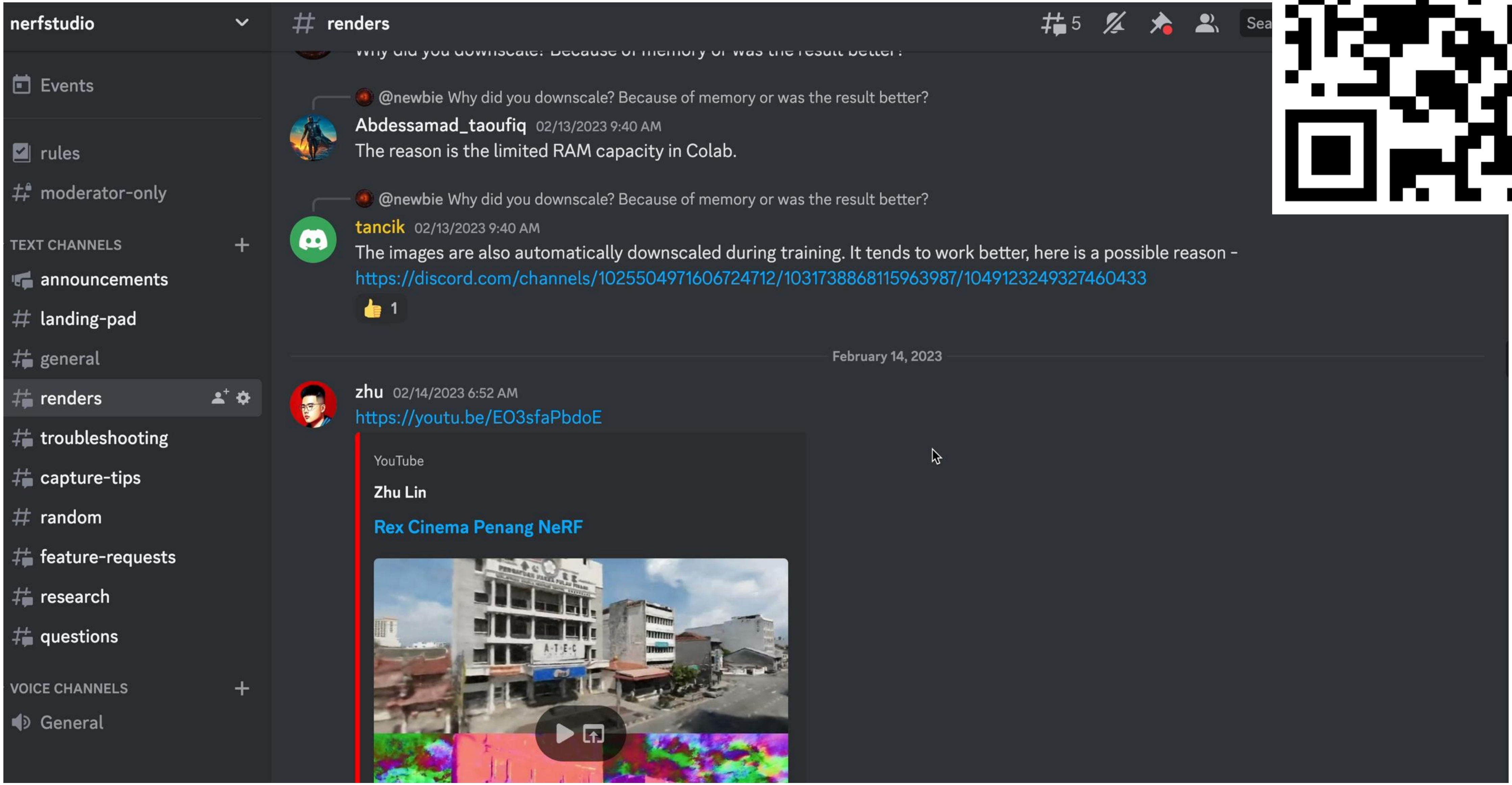
Striking the balance between performance & easy development



Current Model

mip-NeRF 360, NeRF-W, NeRF--/BaRF, InstantNGP

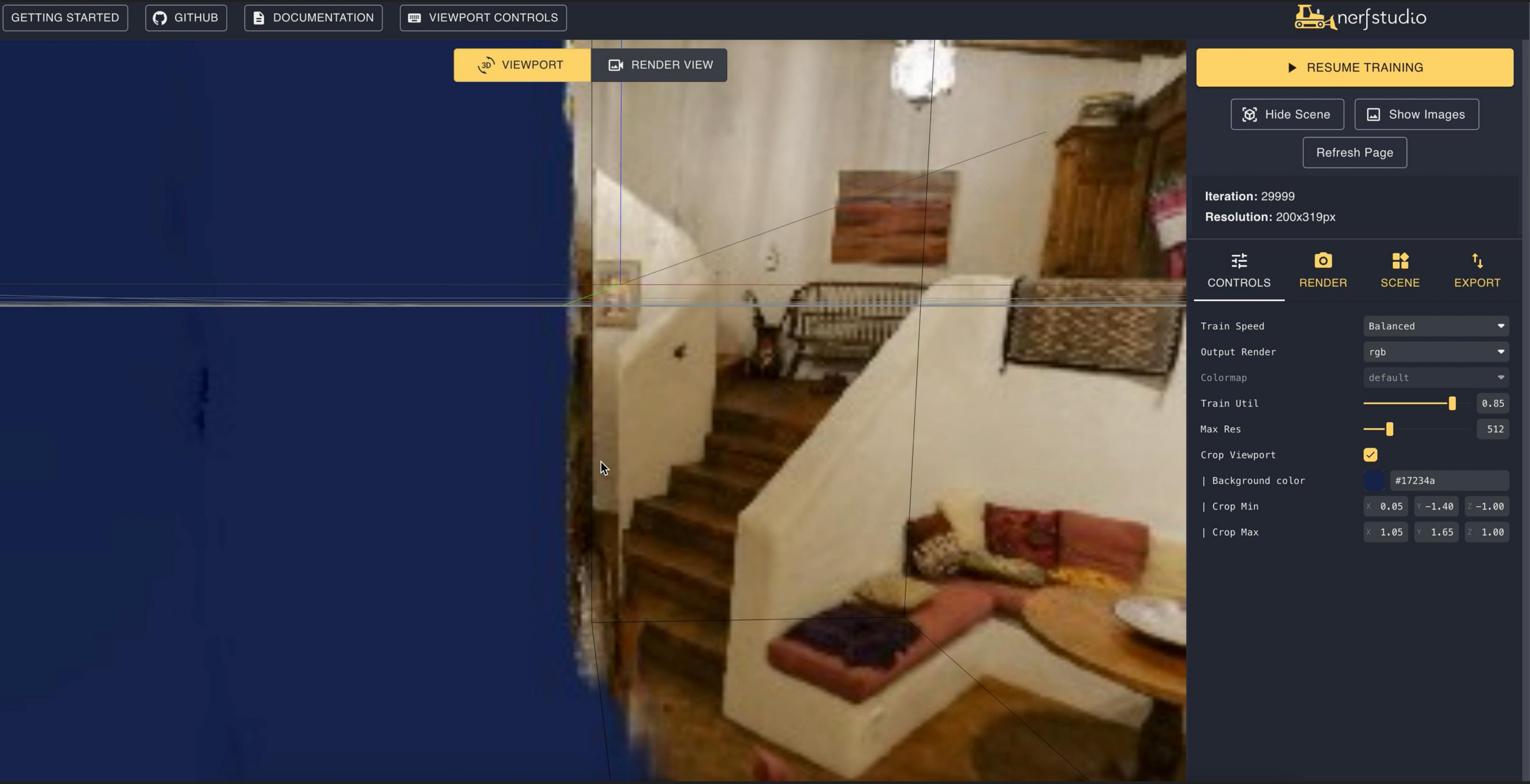
An Active Discord Commur




Nerfacto


Nerfacto-huge

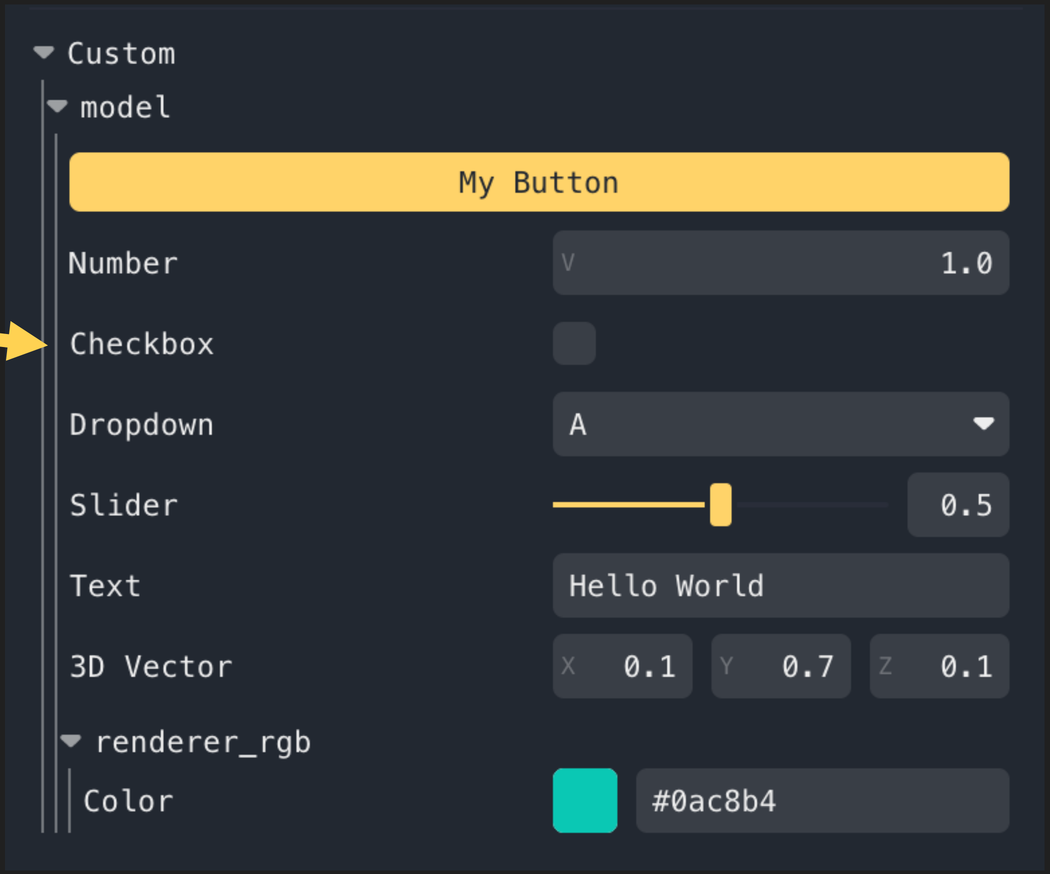
Viewer


GETTING STARTED
GITHUB
DOCUMENTATION
VIEWPORT CONTROLS
nerfstudio
VIEWPORT
RENDER VIEW
RESUME TRAINING
Hide Scene
Show Images
Refresh Page
Iteration: 29999
Resolution: 200x319px
CONTROLS
RENDER
SCENE
EXPORT
Train Speed
Balanced
Output Render
rgb
Colormap
default
Train Util
0.85
Max Res
512
Crop Viewport
| Background color
#17234a
| Crop Min
X 0.05
Y -1.40
Z -1.00
| Crop Max
X 1.05
Y 1.65
Z 1.00


GETTING STARTED
GITHUB
DOCUMENTATION
VIEWPORT CONTROLS
nerfstudio
VIEWPORT
RENDER VIEW
RESUME TRAINING
Hide Scene
Show Images
Refresh Page
Iteration: 29999
Resolution: 320x512px
CONTROLS
RENDER
SCENE
EXPORT
Train Speed
Balanced
Output Render
rgb
Colormap
default
Train Util
0.85
Max Res
512
Crop Viewport
| Background color
#7091fd
| Crop Min
| Crop Max

Custom Interactivity

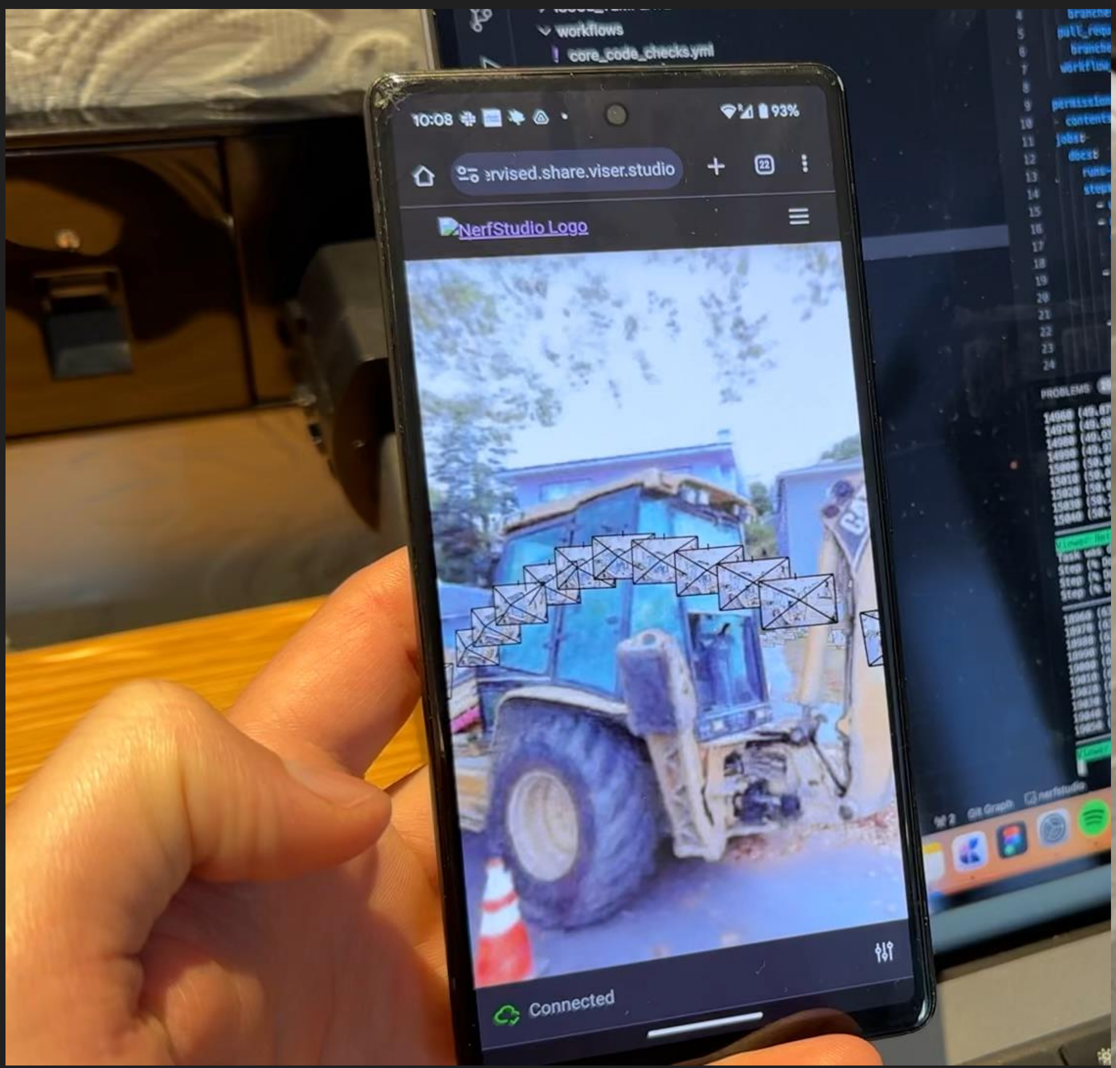
```
self.checkbox = ViewerCheckbox(name="Checkbox", default_value=False)
⋮
current_value = self.checkbox.value
```



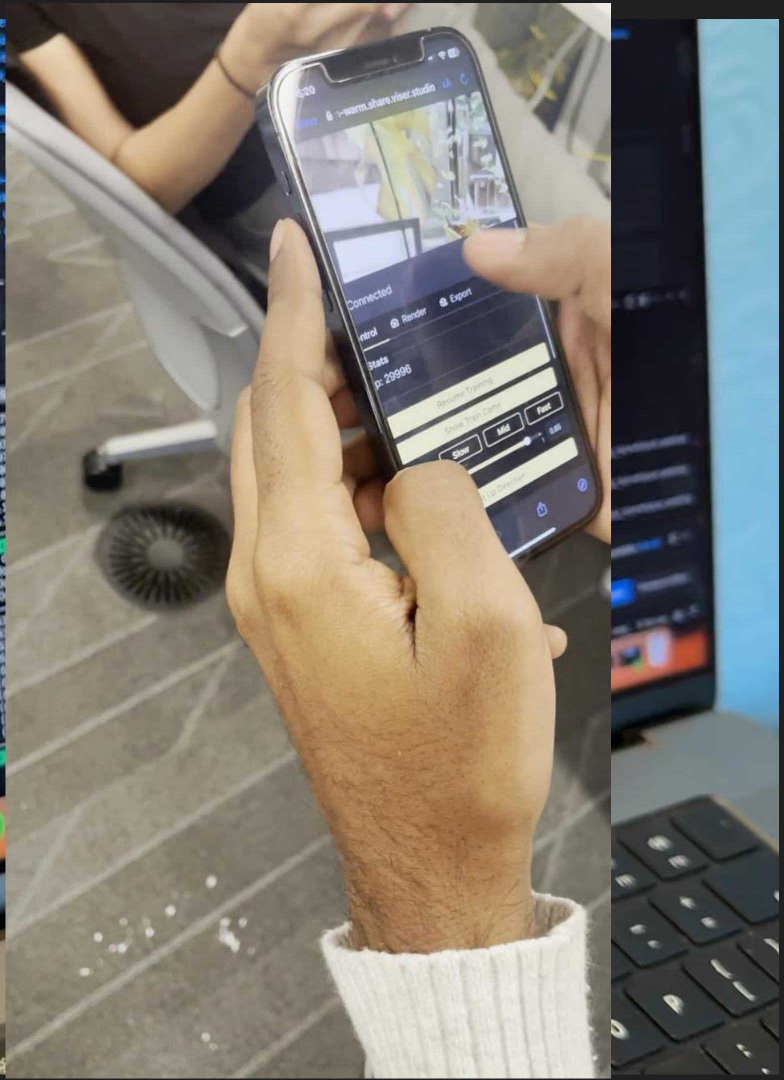
Coming Soon: Viser Integration

viser.studio

Python library for web-based 3D visualization



Shareable Links

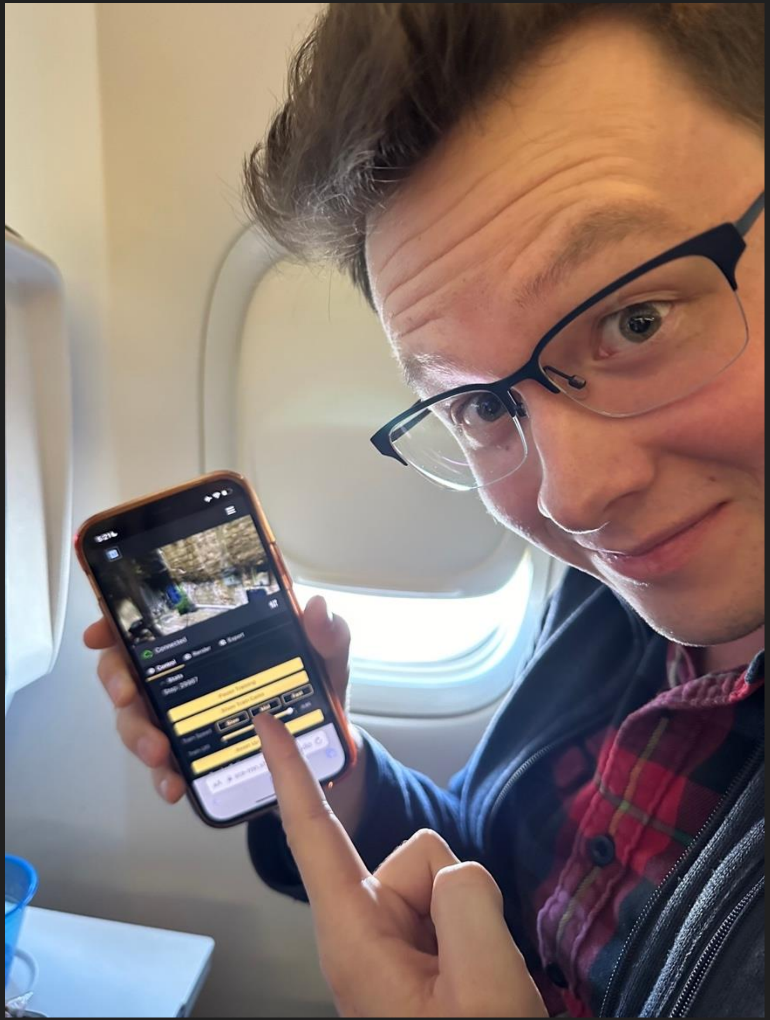


Mobile Support

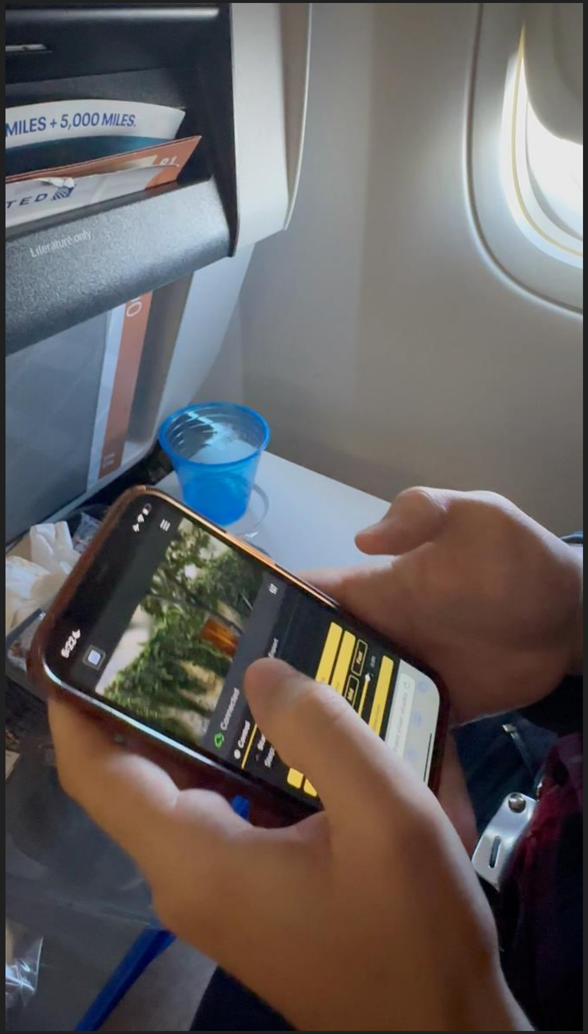
Coming Soon: Viser Integration

Python library for web-based 3D visualization

viser.studio



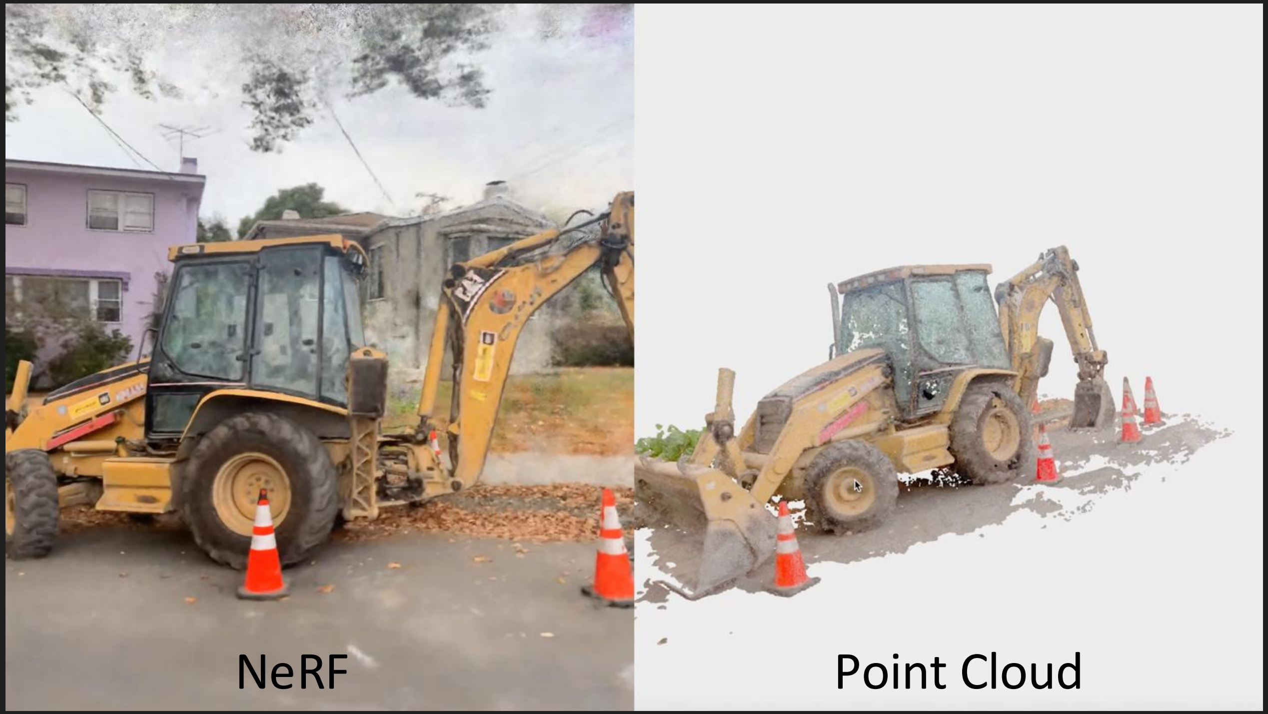
Shareable Links



Mobile Support

Export Options

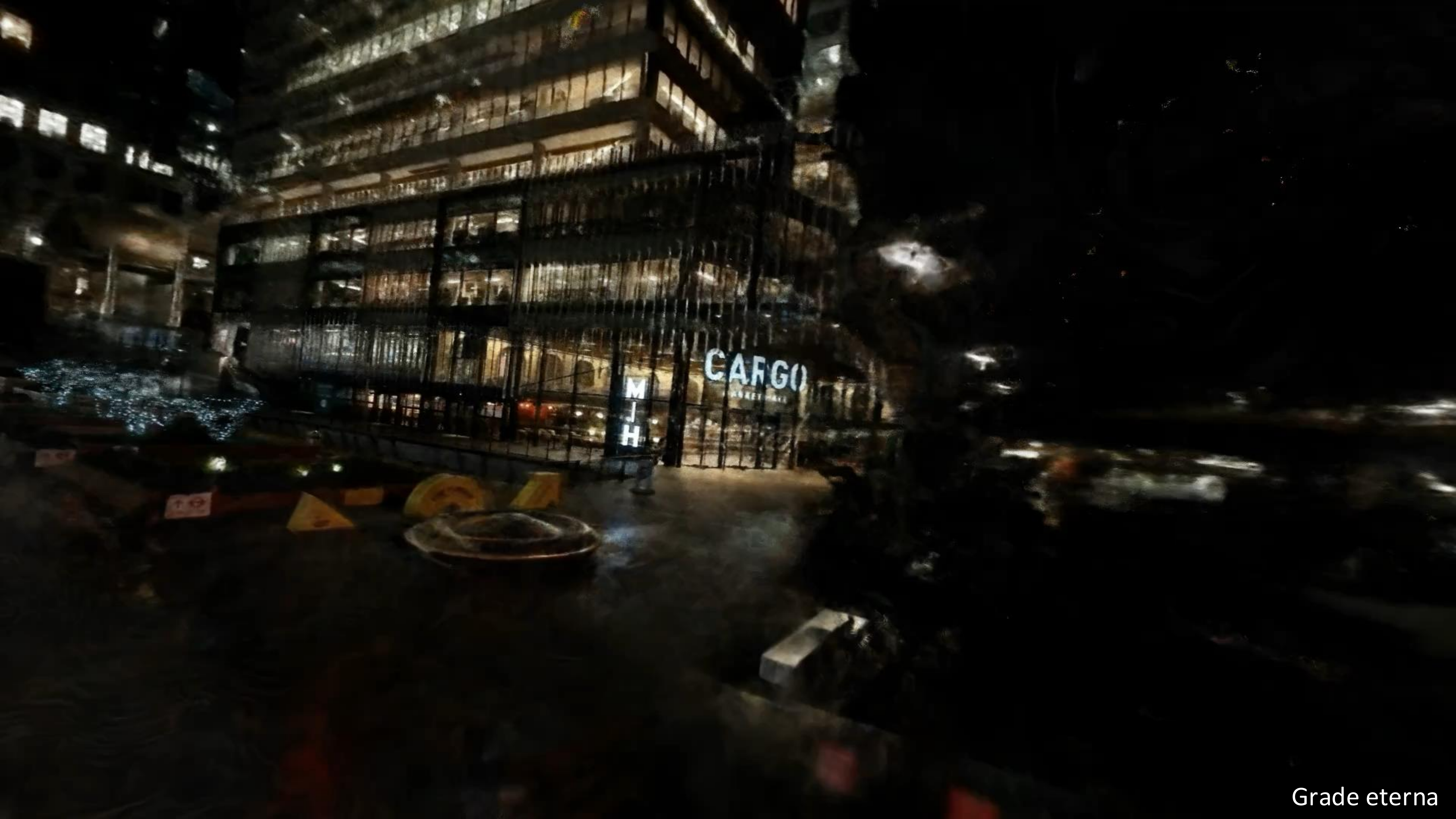
Geometry Conversion



Camera Effects

VFX: Blender Integration


nerfstudio



Grade eterna

Grade eterna