The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

Fall 2023

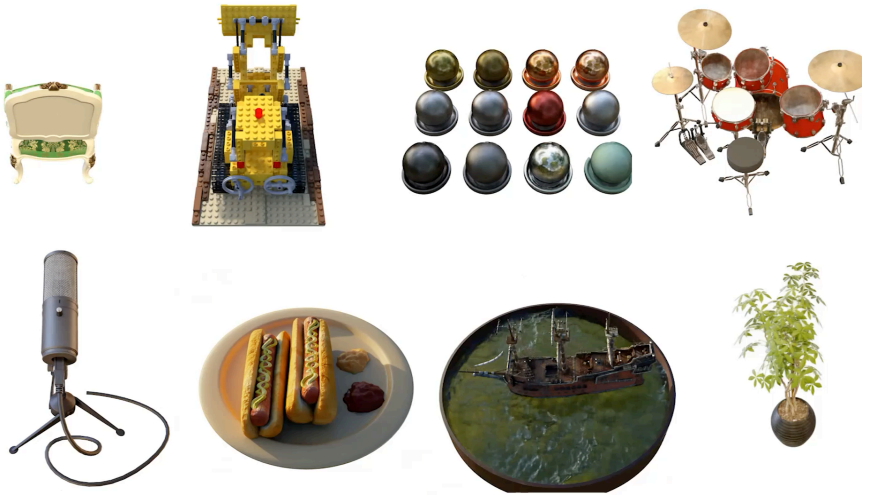
ADVANCED TOPICS IN COMPUTER VISION

Atlas Wang
Associate Professor, The University of Texas at Austin

Visual Informatics Group@UT Austin
https://vita-group.github.io/

Neural Radiance Field

Given 100 views



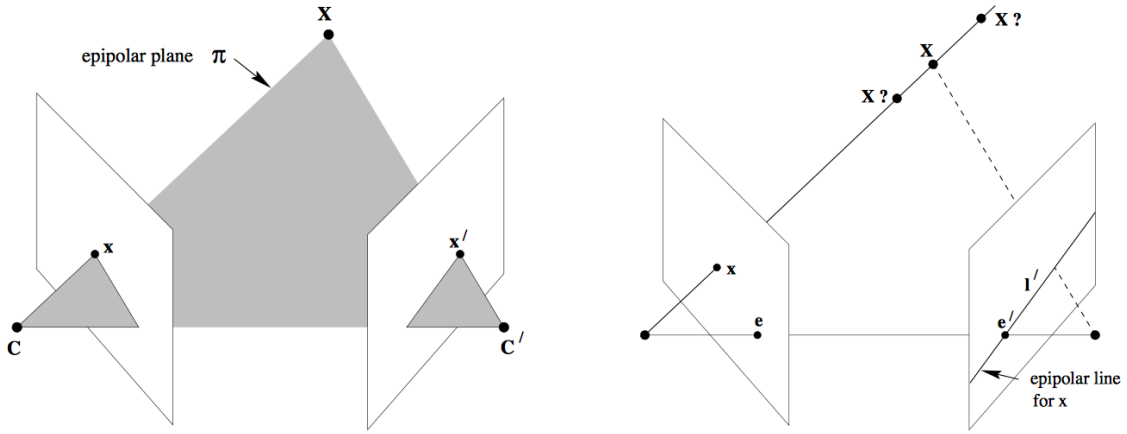
Neural Radiance Field - Pipeline

Given a set of sparse views of an object with known camera poses

Optimize a NeRF model

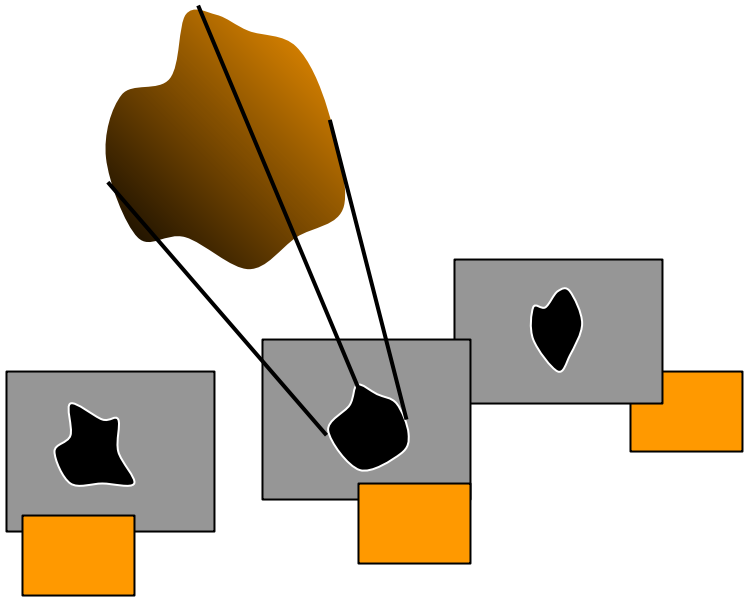
3D reconstruction viewable from any angle

3D Reconstruction - Inverse Problem



Point based Imaging Model

$$\arg\min_{\boldsymbol{X}\in\mathbb{R}^{N\times 3}} \sum_{i,j} \left\| \boldsymbol{P}^{(i)}(\boldsymbol{R}^{(i)}\boldsymbol{x}_j^{(i)} + \boldsymbol{t}^{(i)}) - \boldsymbol{y}_j^{(i)} \right\|_2^2$$



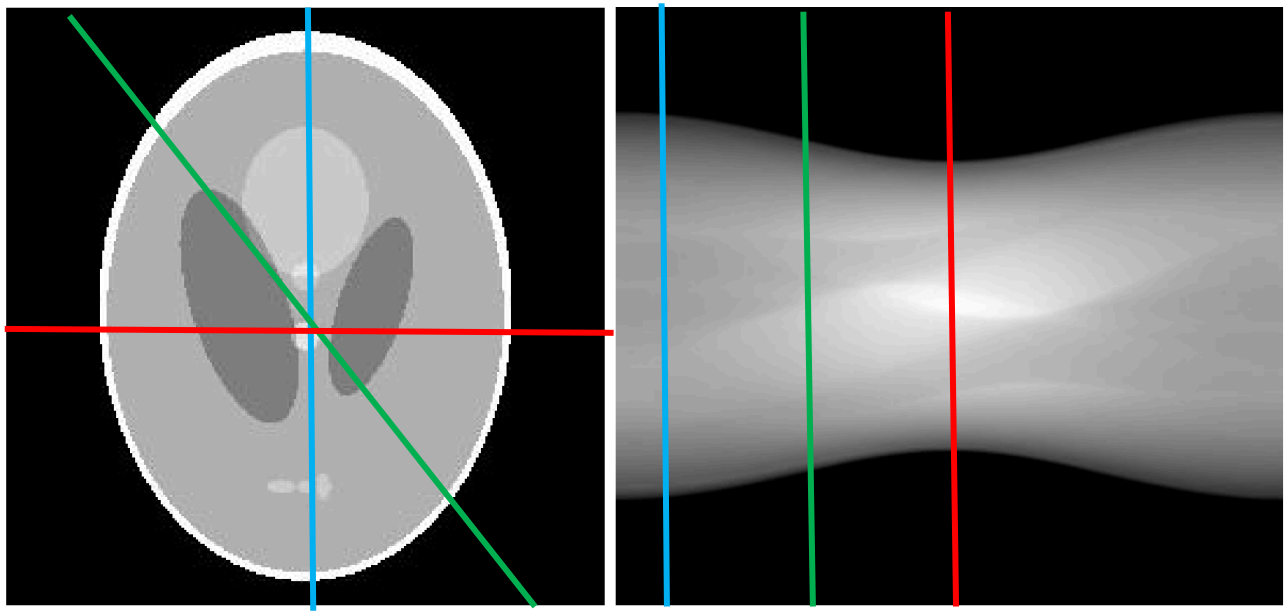
Rendering based Imaging Model

$$\arg\min_{\boldsymbol{V}} \sum_{i} \left\| \mathcal{R}(\boldsymbol{V}; \boldsymbol{R}^{(i)}, \boldsymbol{t}^{(i)}, \boldsymbol{P}^{(i)}) - \boldsymbol{y}^{(i)} \right\|_2^2$$

3D reconstruction is essentially solving an inverse problem!

An Illustrative Example: Computational Tomography

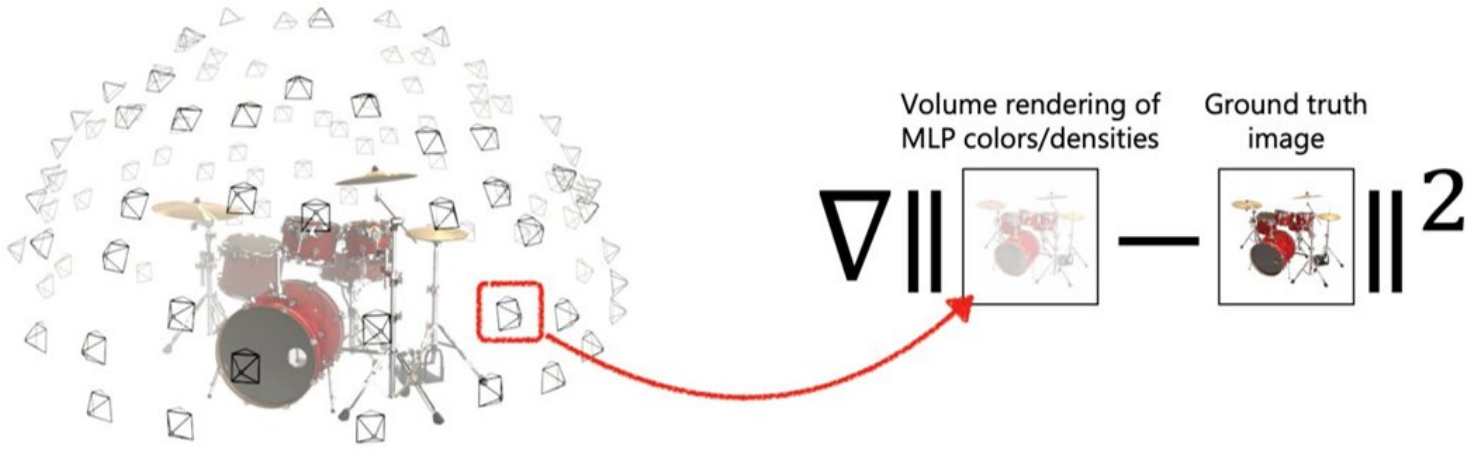
- In CT, the observations are CT slices from different angles.
- The imaging formation is modeled as a line integration (projection).
- We solve a least-square problem to recover CT images.



Rendering function:

$$\mathcal{R}(\boldsymbol{V};\theta) = \int V(x,y)\delta(x - \cos\theta x + y - \sin\theta)dxdy$$

NeRF as an Inverse Imaging Problem

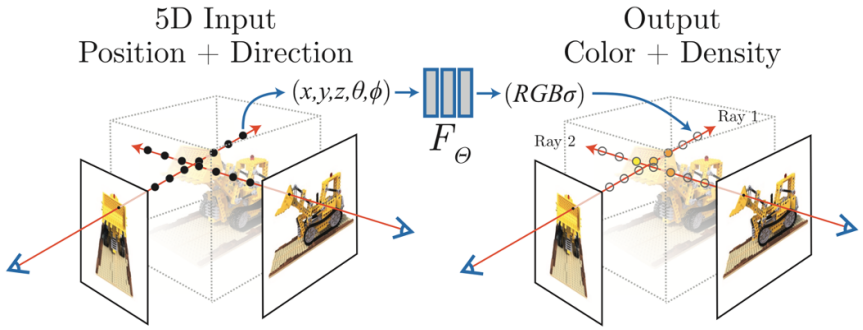


$$\arg\min_{\boldsymbol{V}} \sum_i \left\| \mathcal{R}(\boldsymbol{V}; \boldsymbol{R}^{(i)}, \boldsymbol{t}^{(i)}, \boldsymbol{P}^{(i)}) - \boldsymbol{y}^{(i)} \right\|_2^2$$

Neural Radiance Field - Two Knobs

$$\arg\min_{\boldsymbol{V}} \sum_i \left\| \mathcal{R}(\boldsymbol{V}; \boldsymbol{R}^{(i)}, \boldsymbol{t}^{(i)}, \boldsymbol{P}^{(i)}) - \boldsymbol{y} \right\|_2^2$$

1. Neural radiance 3D scene representation



2. Volume rendering

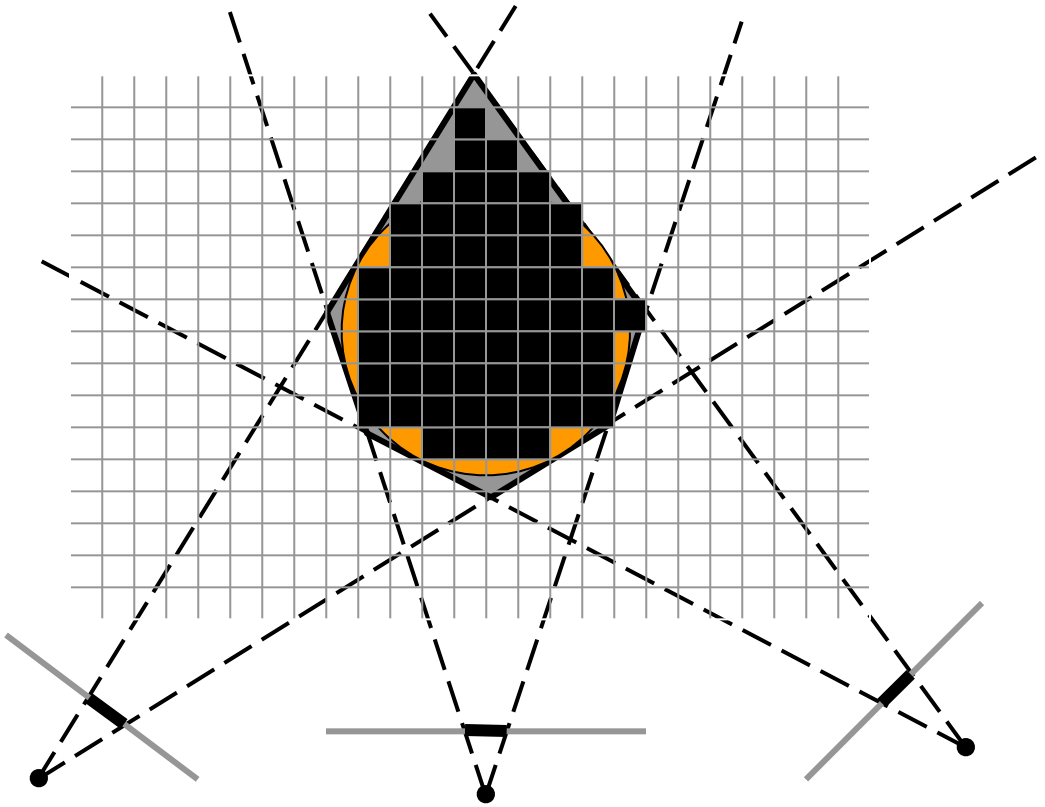
$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt\,, \text{ where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right)$$

Ray color | Transmittance | Density | Color

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

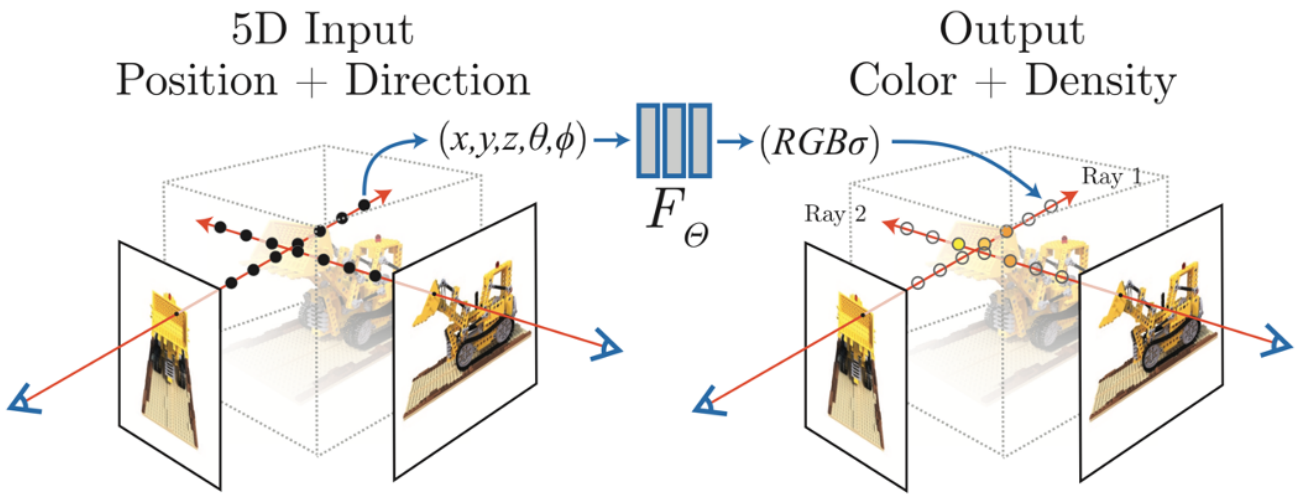
Position | Direction

Volumetric 3D Representation – Occupancy Field



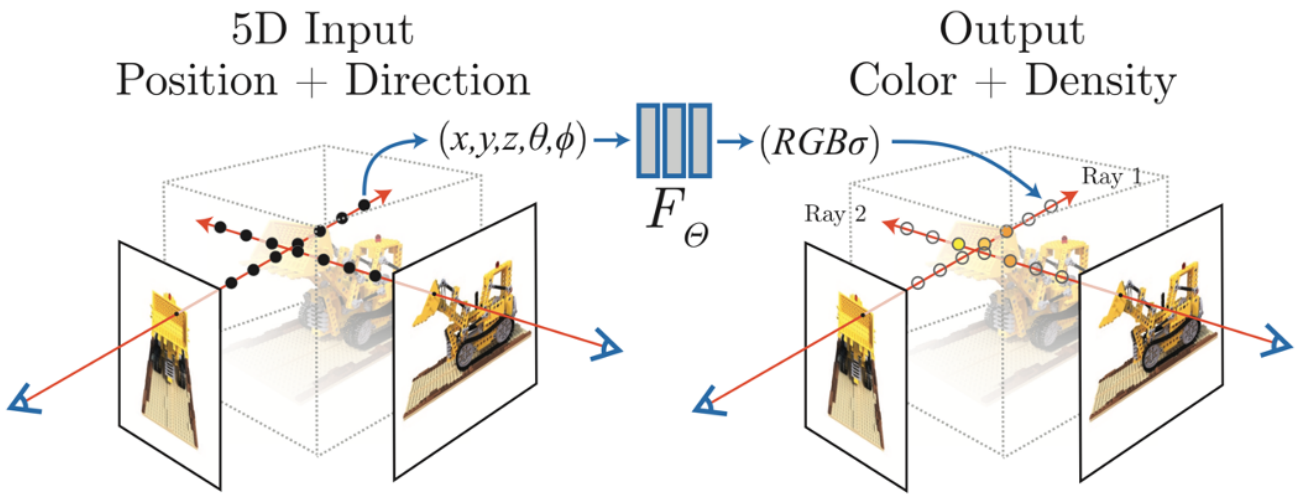
Color voxel black if occupied by the object

Radiance Field



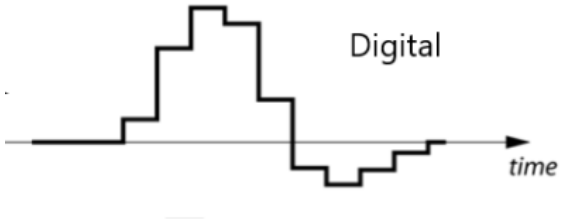
- Each point will store a density value between [0, 1]
  - How likely this point is occupied by the object
  - The opacity of this point (consider glasses, petal)

Radiance Field



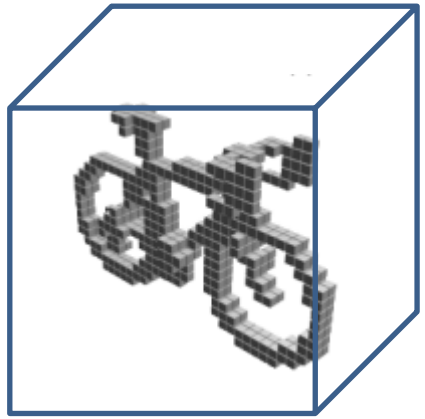
- Each voxel will also store an RGB value
  - The color of the voxel

Voxel Representation



Audio: vector

Image: pixel array



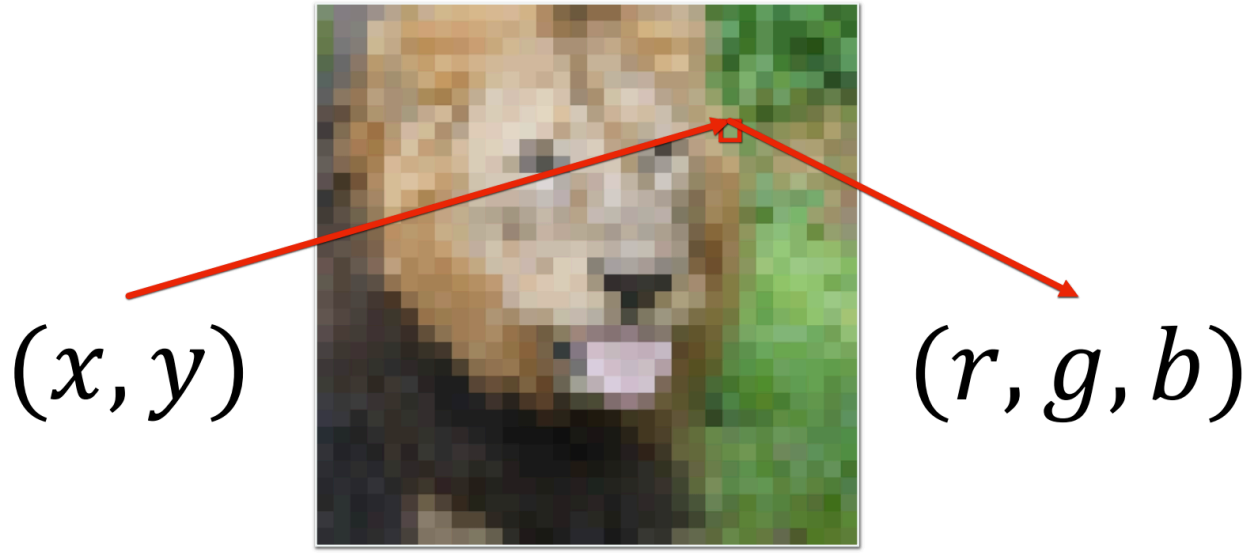
3D: voxel array

What's the problem with vectorized signals?

Finite resolution!

Concept of Implicit Neural Representation

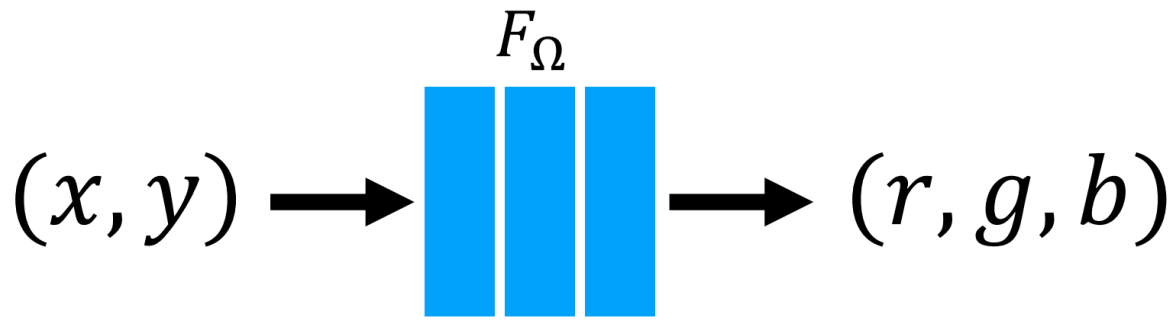
Toy problem: storing 2D image data



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

Concept of Implicit Neural Representation

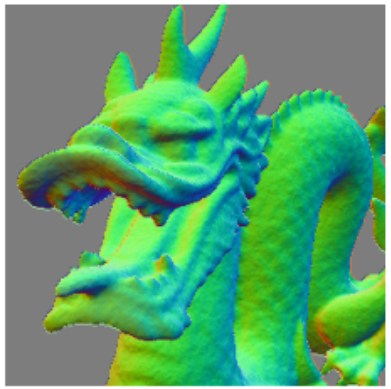
Toy problem: storing 2D image data



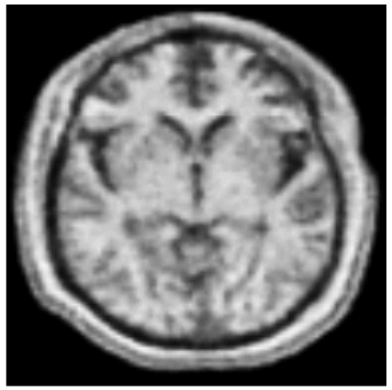
What if we train a simple fully-connected network (MLP) to do this instead?

More Examples

(b) Image regression
$(x, y) \rightarrow$ RGB



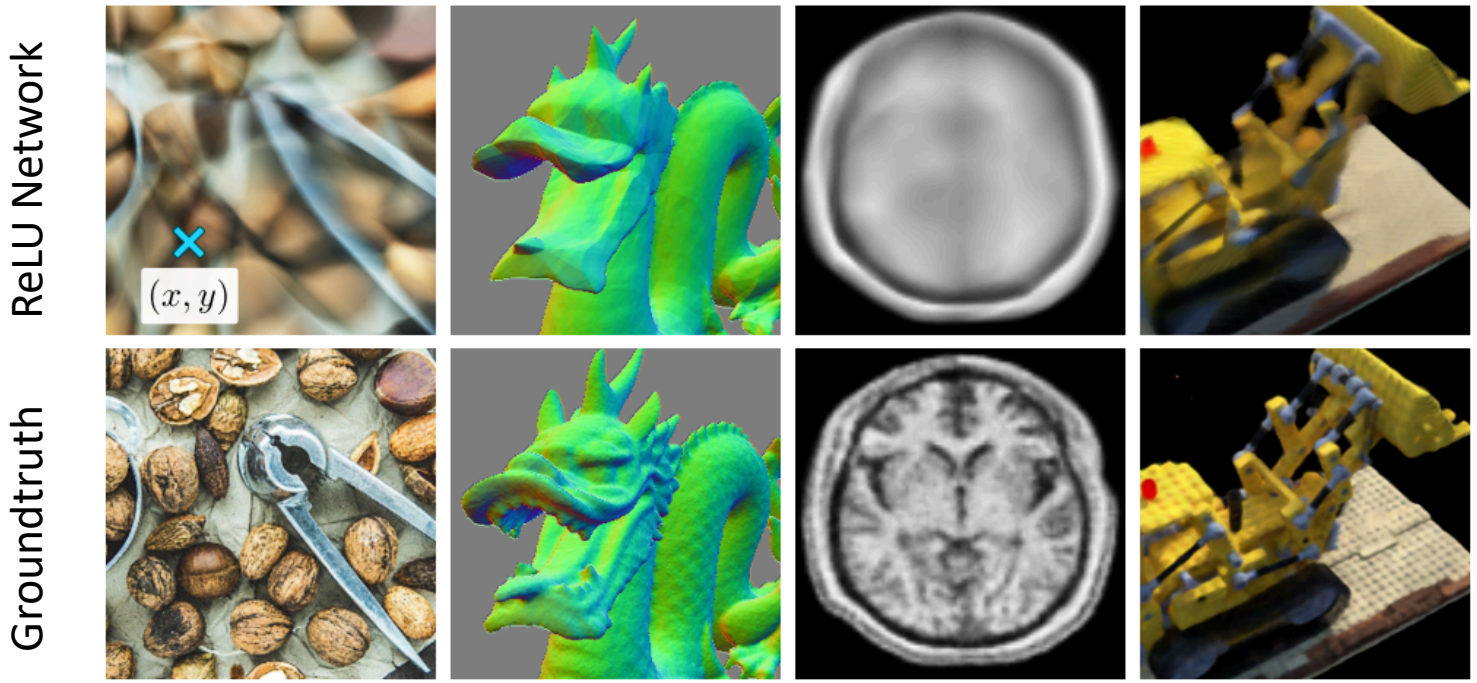
(c) 3D shape regression
$(x, y, z) \rightarrow$ occupancy



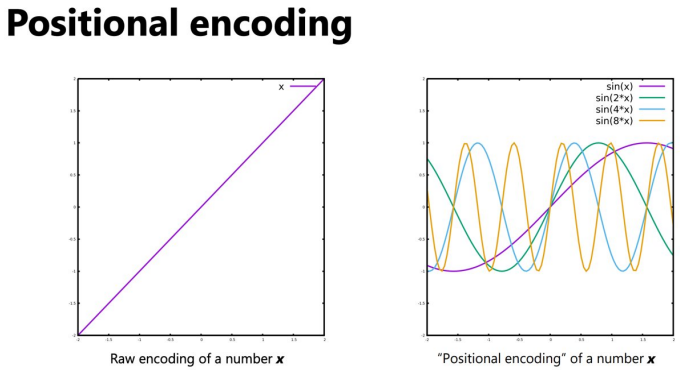
(d) MRI reconstruction
$(x, y, z) \rightarrow$ density

(e) Inverse rendering
$(x, y, z) \rightarrow$ RGB, density

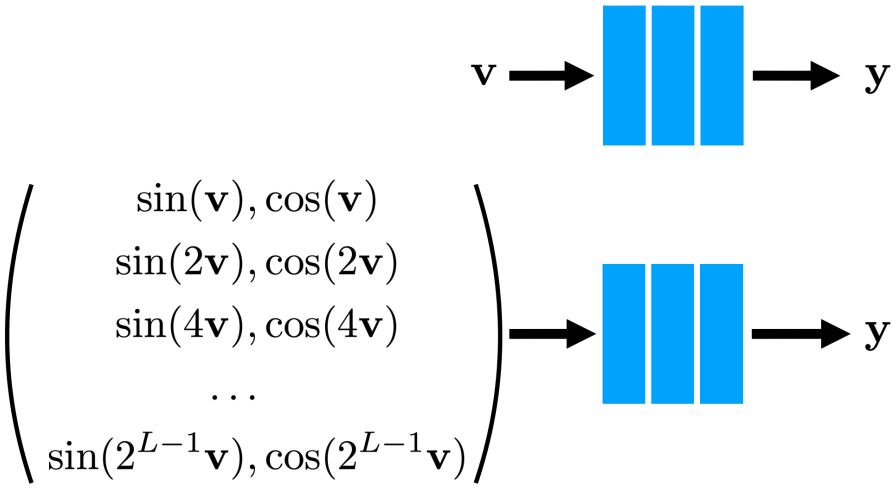
Naïve Approaches Fail

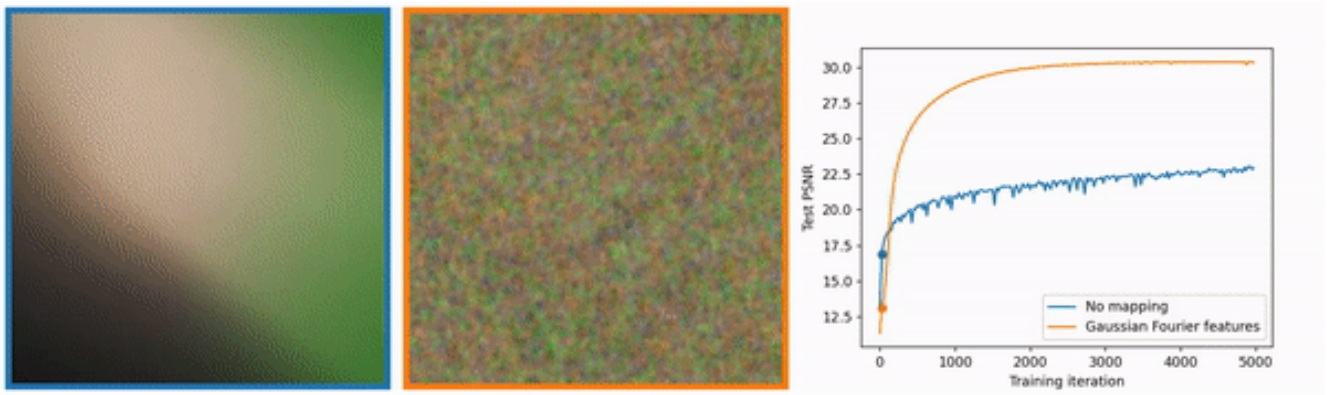


Solution I: Positional Encoding



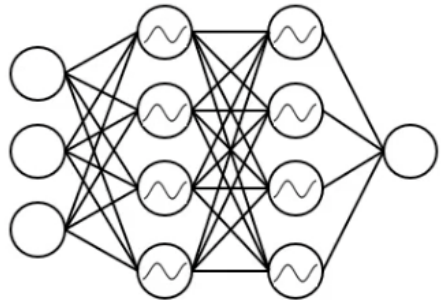
Example mapping: "positional encoding"



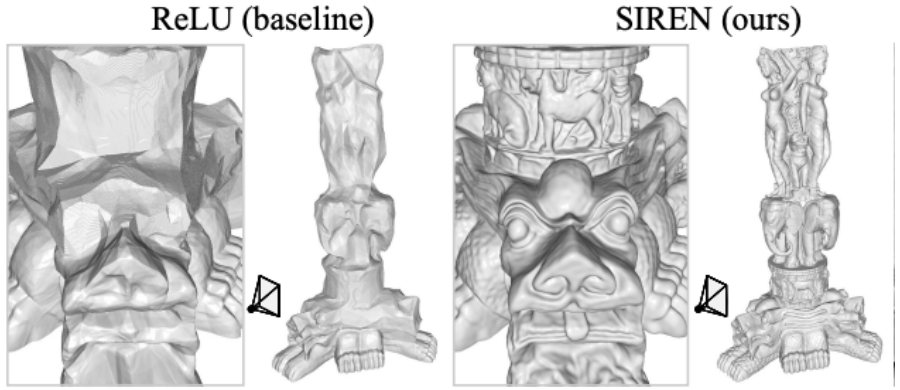


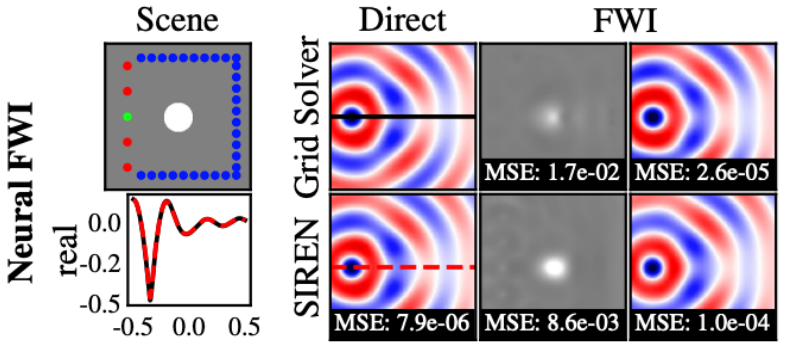
Solution II: Sinusoidal Network (SIREN)

1. Simply replace the activation function with a sinusoidal function
2. SIREN's derivatives are also a SIREN!
   a) It can be used to solve differential equations



Sinusoidal Network

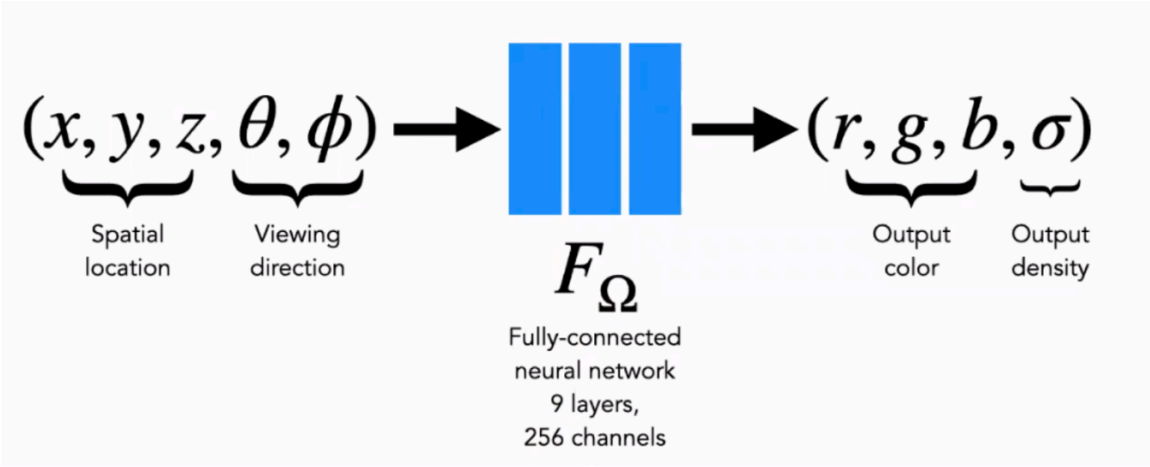




$$\mathcal{L}_{\text{sdf}} = \int_{\Omega} \big\| \, |\nabla_{\mathbf{x}} \Phi(\mathbf{x})| - 1 \big\| d\mathbf{x} + \int_{\Omega_0} \|\Phi(\mathbf{x})\| + \big(1 - \langle \nabla_{\mathbf{x}} \Phi(\mathbf{x}), \mathbf{n}(\mathbf{x}) \rangle\big) d\mathbf{x} + \int_{\Omega \setminus \Omega_0} \psi\big(\Phi(\mathbf{x})\big) d\mathbf{x}$$

$$\underset{m,\Phi}{\arg\min} \sum_{i=1}^{N} \int_{\Omega} |\text{Ш}_r(\Phi_i(\mathbf{x}) - r_i(\mathbf{x}))|^2 \, d\mathbf{x} \text{ s.t. } H(m)\, \Phi_i(\mathbf{x}) = -f_i(x),\ 1 \le i \le N,\ \forall \mathbf{x} \in \Omega$$

Neural Implicit Field in NeRF

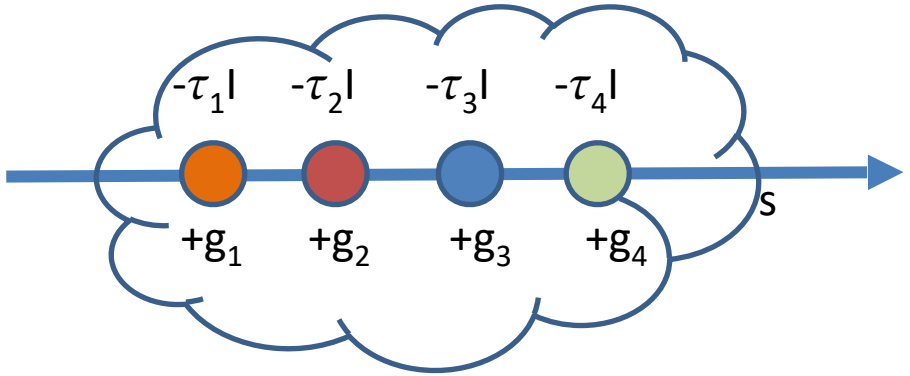


- Model the scene as a continuous mapping: $(\mathrm{x}, \mathrm{y}, \mathrm{z}) \rightarrow (r, g, b, \sigma)$
  - Parameterized by an MLP with position encoding
  - We can also easily incorporate view angles as input to model view-dependent effects! $(x, y, z, \theta, \phi) \rightarrow (r, g, b, \sigma)$

NeRF is yet another Differential System

$$\frac{dI}{ds} = g(s) - \tau(s) I(s)$$

Emission Attenuation



$$I(D) = I_0 \exp\left(-\int_0^D \tau(t)\,dt\right) + \int_0^D g(s) \exp\left(-\int_s^D \tau(t)\,dt\right) ds$$

Volume Rendering

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt\,, \text{ where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right)$$

Ray color | Transmittance | Density | Color

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

Position | Direction

Generate views with traditional volume rendering

Rendering model for ray r(t) = o + td:

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

colors

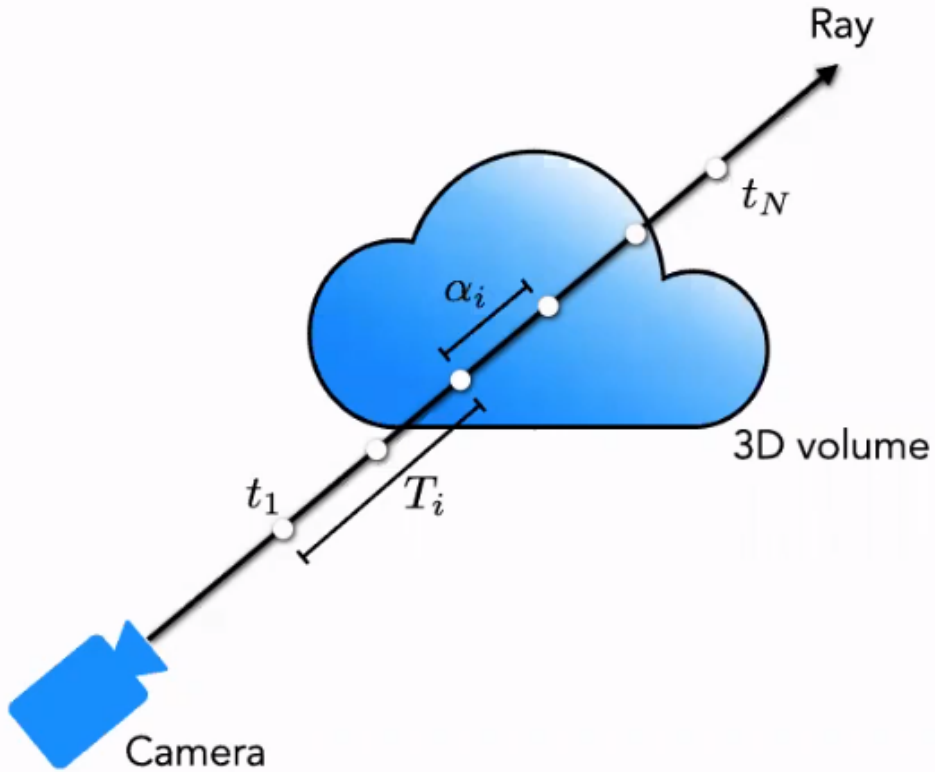
weights

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*:

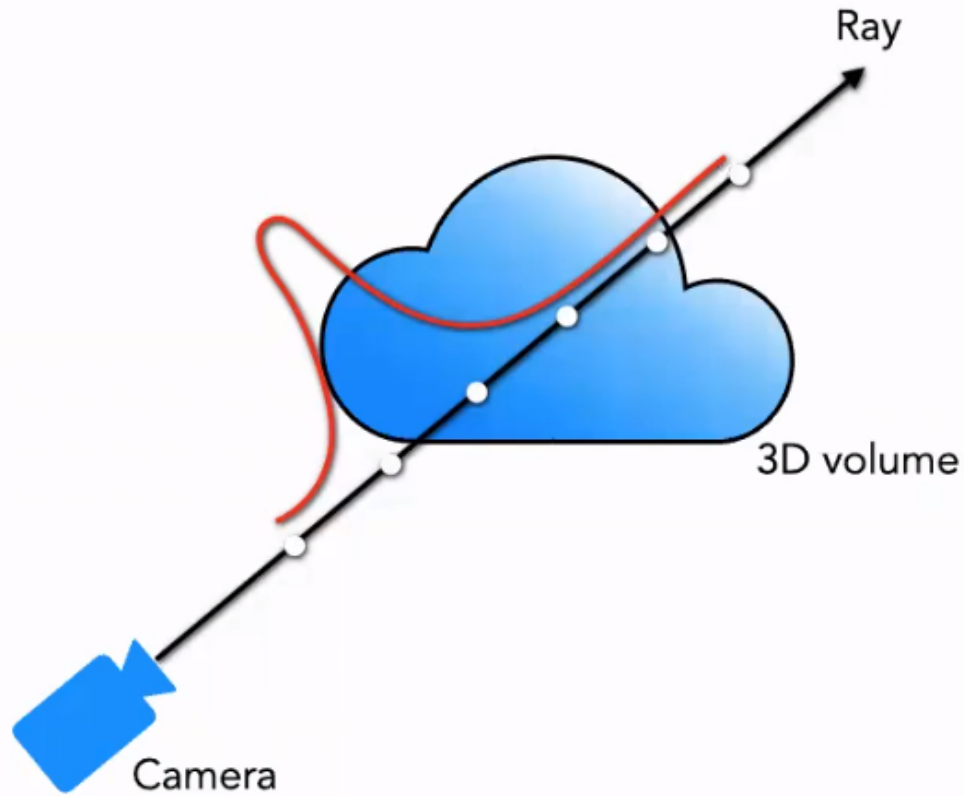
$$\alpha_i = 1 - e^{-\sigma_i \delta t_i}$$



Two pass rendering: coarse

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

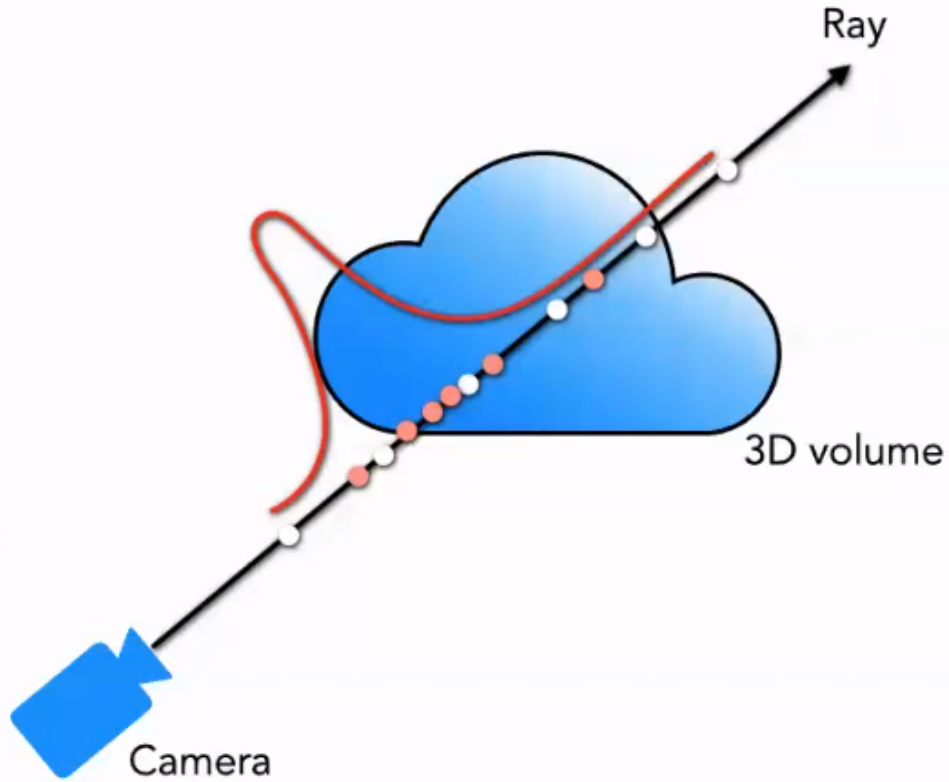
treat weights as probability distribution for new samples



Two pass rendering: fine

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

treat weights as probability distribution for new samples



Volume rendering is trivially differentiable

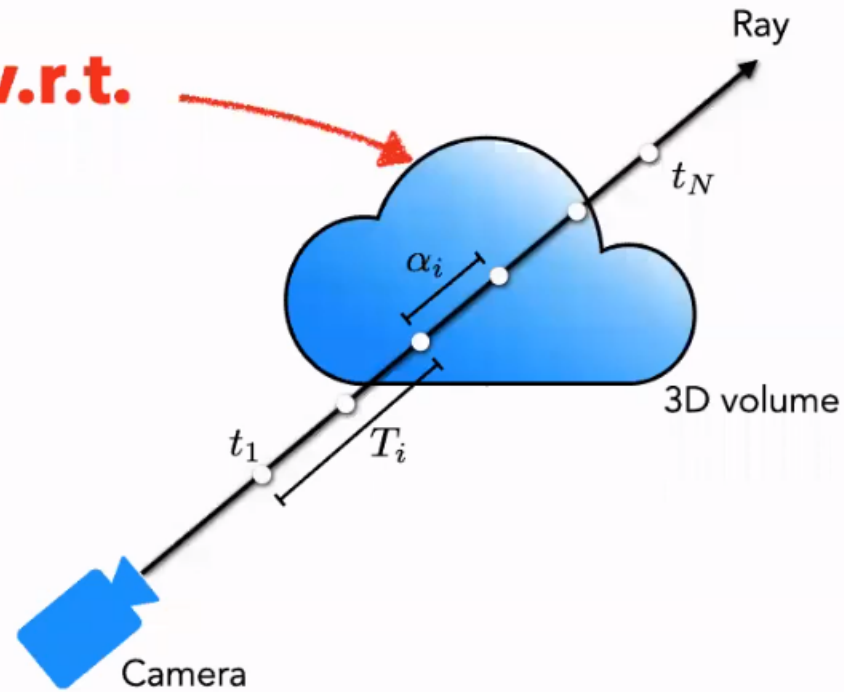
Rendering model for ray r(t) = o + td:

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

weights

colors

differentiable w.r.t.



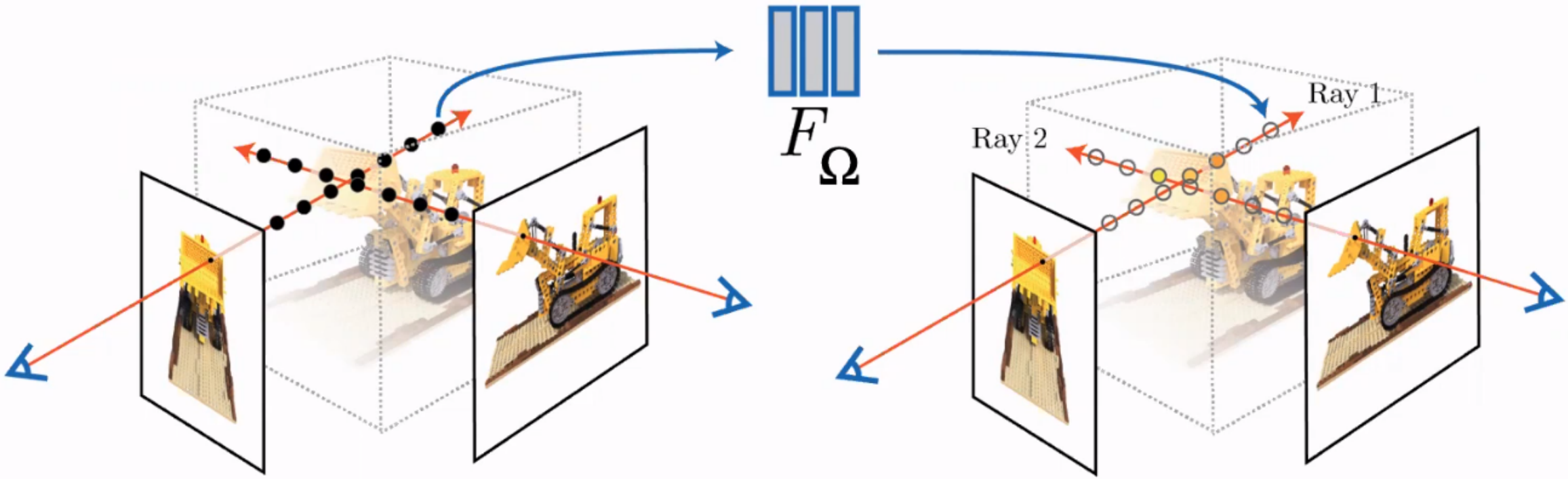
How much light is blocked earlier along ray:

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

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

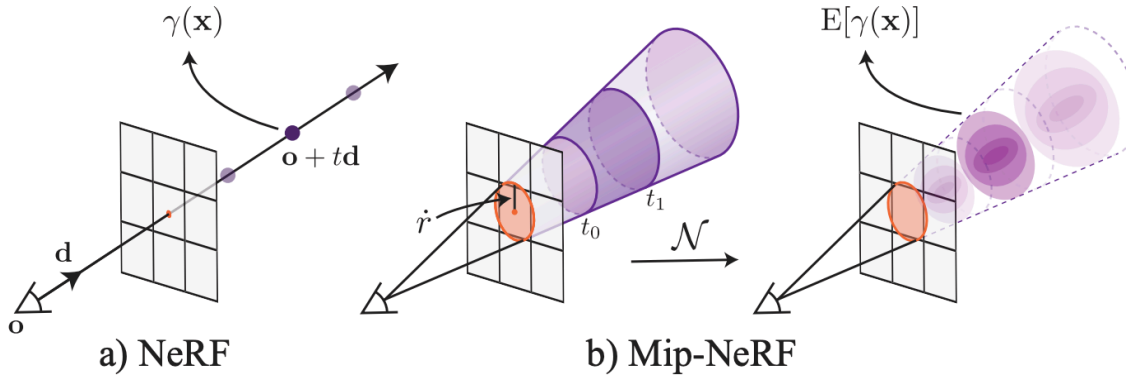
$$\alpha_i = 1 - e^{-\sigma_i \delta t_i}$$

Optimize with gradient descent on rendering loss



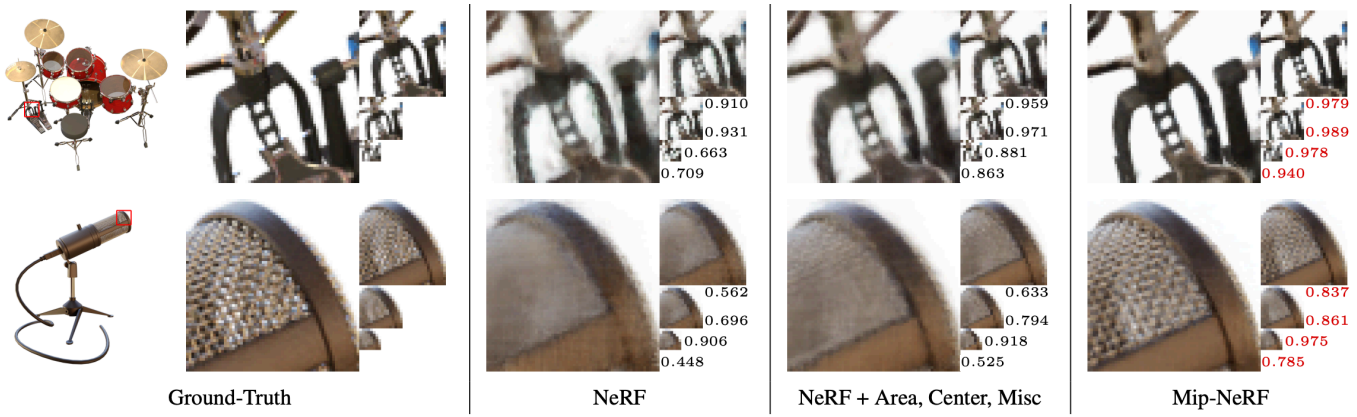
$$\min_{\Omega} \sum_{i} \|\text{render}^{(i)}(F_\Omega) - I_{\text{gt}}^{(i)}\|^2$$

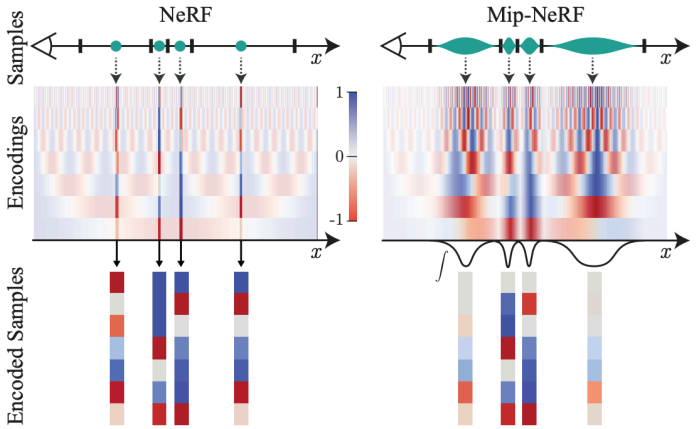
MipNeRF (ICCV'21): Multi-resolution NeRF



1. Considering a single ray as a cone, each pixel value should be the integral over the cone.
2. To simply the integral, integrating via a Gaussian measure leads to a new PE!

$$\gamma(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathrm{E}_{\mathbf{x}\sim\mathcal{N}(\boldsymbol{\mu}_\gamma,\boldsymbol{\Sigma}_\gamma)}[\gamma(\mathbf{x})]$$
$$= \begin{bmatrix}\sin(\boldsymbol{\mu}_\gamma)\circ\exp(-(1/2)\,\mathrm{diag}(\boldsymbol{\Sigma}_\gamma)) \\ \cos(\boldsymbol{\mu}_\gamma)\circ\exp(-(1/2)\,\mathrm{diag}(\boldsymbol{\Sigma}_\gamma))\end{bmatrix}$$





Towards Efficient NeRF

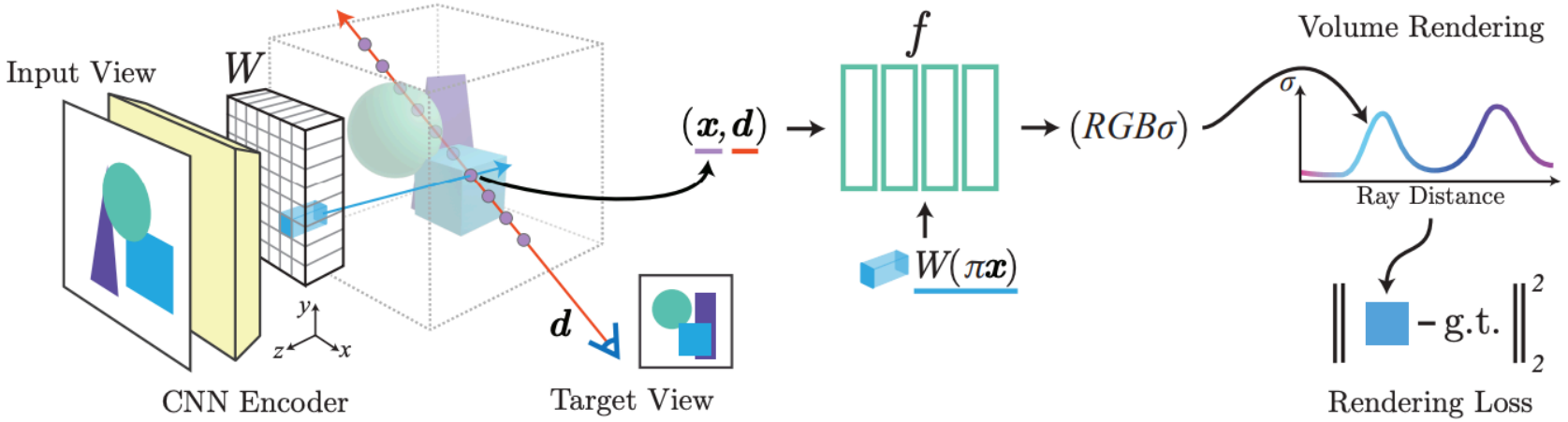
NeRF requires tedious per-scene optimization

- Training one scene equals to training a neural network!
- Usually requires one day.

Solution:

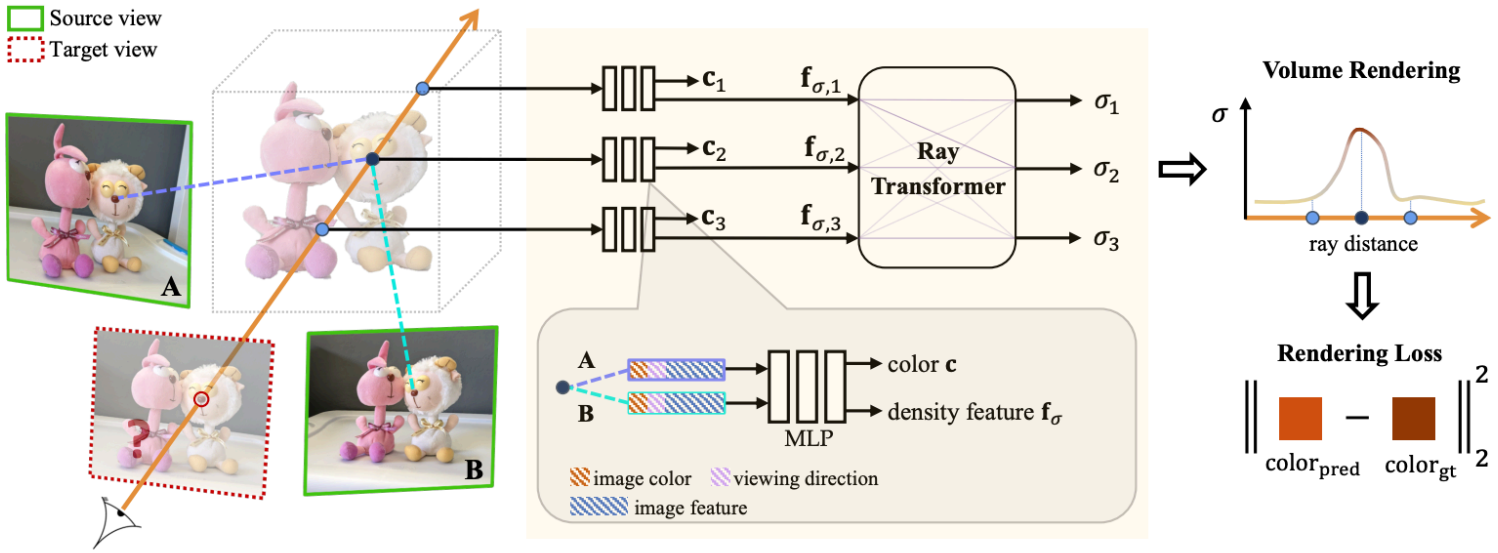
- Generalizable NeRF: reconstruct NeRF directly from images on the fly in a feedforward manner
- Fast per-scene training: still follow per-scene optimization, but significantly accelerate the optimization.

Pixel-NeRF: On-the-fly NeRF Reconstruction



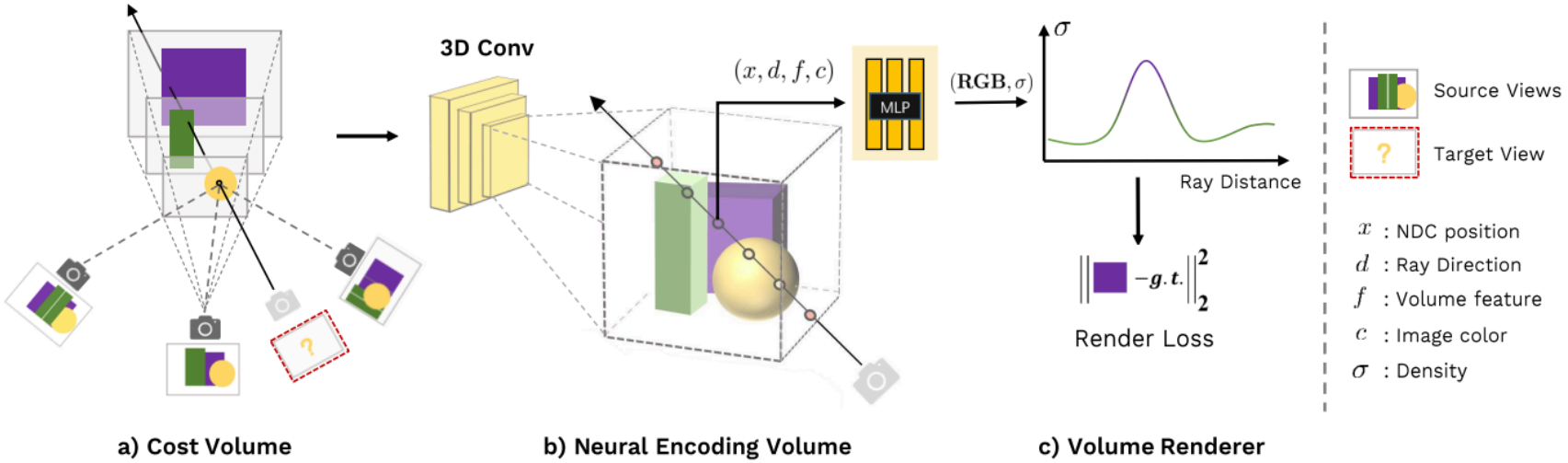
1. Learn an image encoder and a projector that map images to the feature volume **<u>on the fly</u>**
2. Decode the voxel feature to RGB + density
3. Call volume rendering to form images

IBRNet: Image Based Rendering NeRF



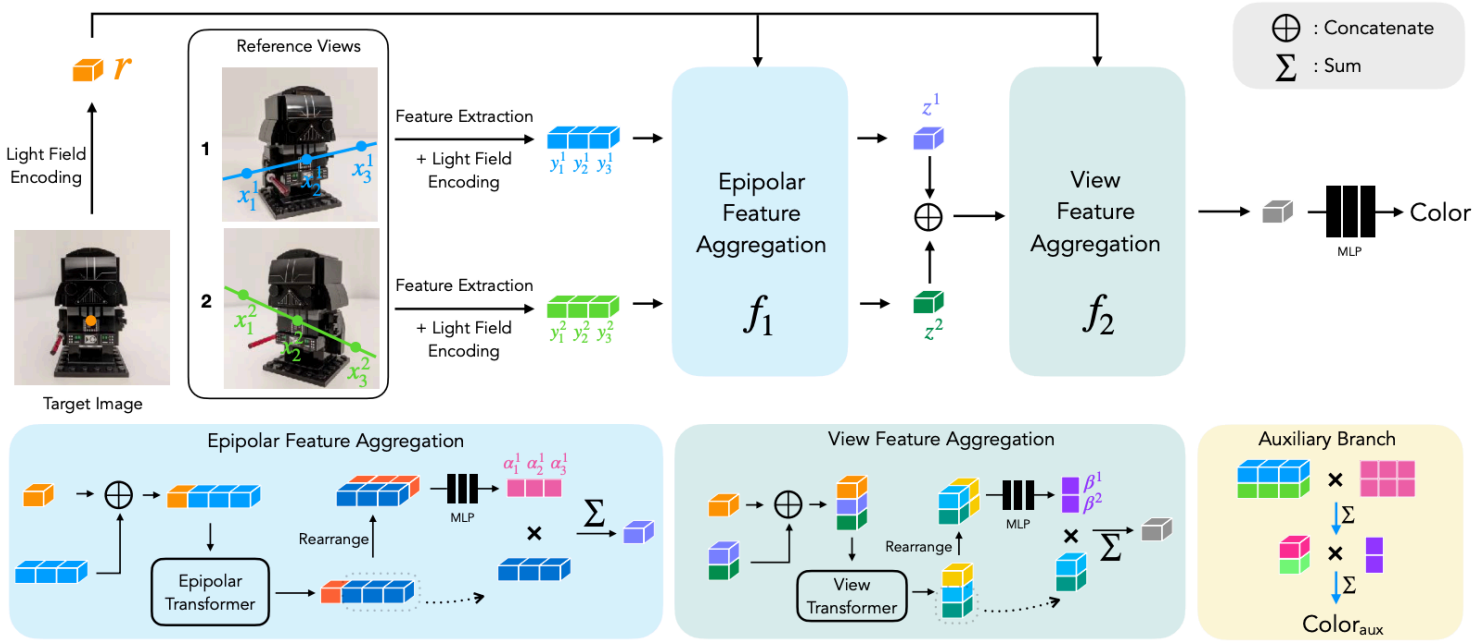
1. Same with PixelNeRF: Learn mapping between images and the feature volume
2. Use transformer to decode voxel features on a ray jointly

MVSNeRF: Multi-view Stereo Guided NeRF

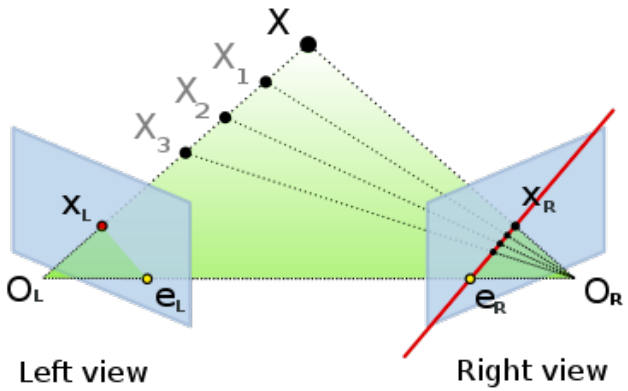


1. Same with PixelNeRF: Learn mapping between images and the feature volume
2. Instead of using feature volume, MVSNeRF borrows cost volume from MVSNet
   a) Cost volume estimates the stereo between views, which carries depth information

Neural Light Field Rendering

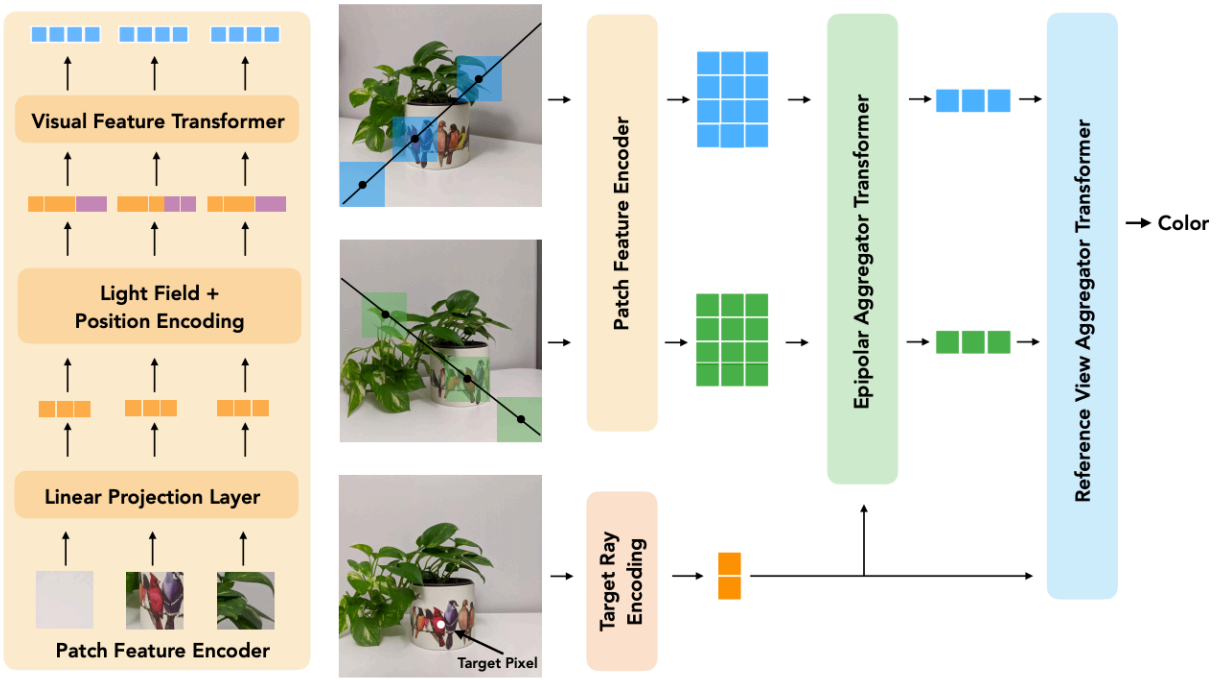


One pixel correspond to a line on a neighboring view!

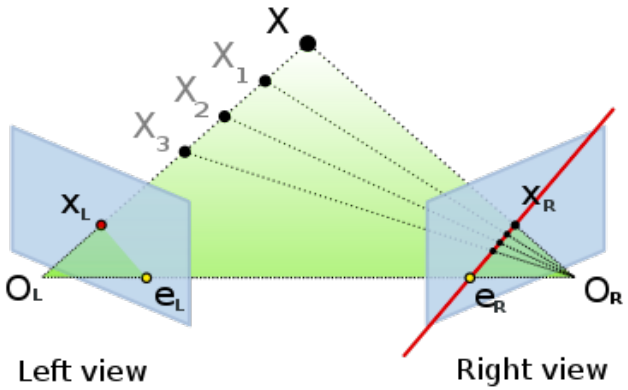


1. Light field rendering inspired rendering pipeline:
   a) Search an epipolar line, and use transformer to extract epipolar line features
   b) Use the second transformer to aggregate epipolar line features
   c) Add light field coordinates to encode view dependence

Generalizable Patch-Based Neural Rendering

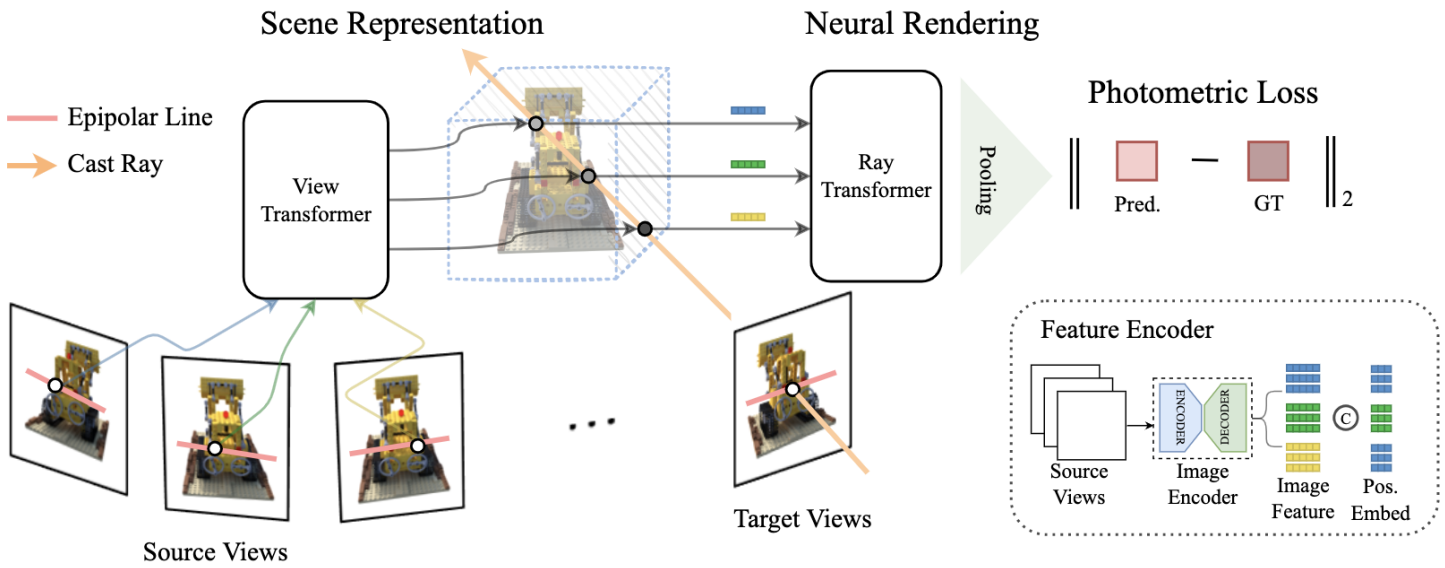


One pixel correspond to a line on a neighboring view!



1. Similar to light field rendering, but:
    1. Tokenize each image as patches: search correspondence with local windows
    2. Add visual feature transformer to exchange visual information between potentially corresponding patches on different reference images.

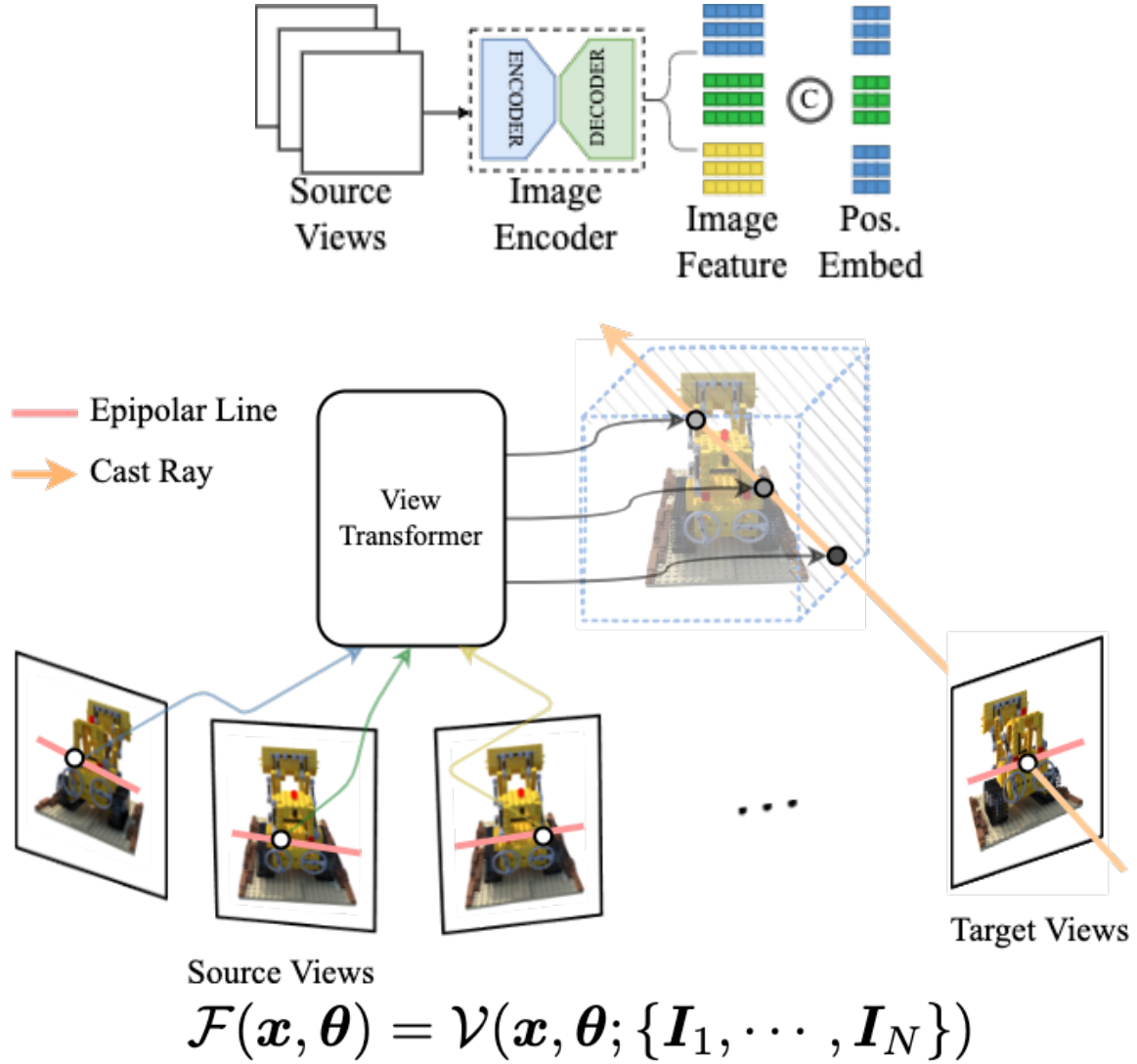
Generalizable NeRF Transformer (GNT)



1. Modeling "image -> feature volume" and "volume rendering" as two sequence functions!
   a. View Transformer: aggregate multi-view images for scene representation
   b. Ray Transformer: aggregate a sequence of points on a ray for ray-based rendering

View Transformer: Universal Scene Representation

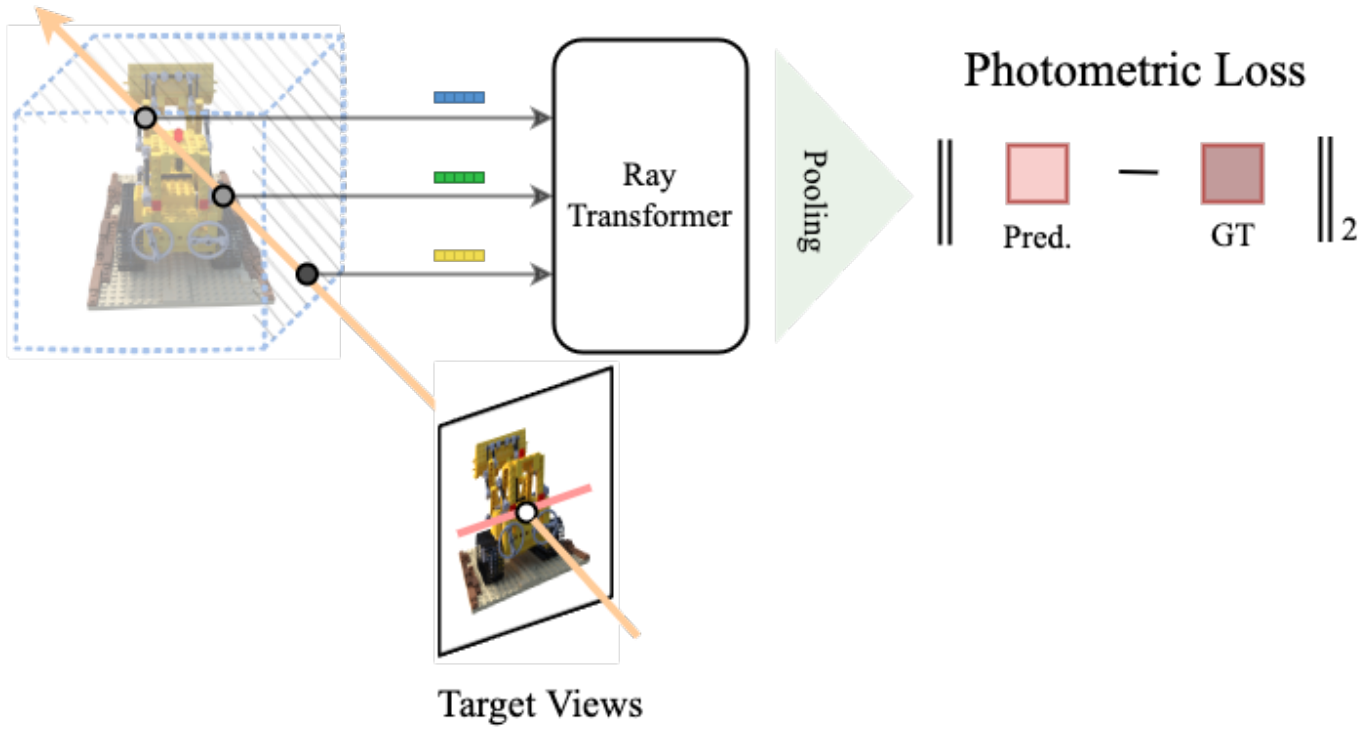
1. Extract image features for each source view via an image encoder.
2. Project 3D coordinates onto each source view plane following epipolar geometry.
3. Interpolate the features of 2D projected point on the image plane.
4. Feed collected image features as a sequence into the view transformer to output the 3D point features.



$$\mathcal{F}(\boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{V}(\boldsymbol{x}, \boldsymbol{\theta}; \{\boldsymbol{I}_1, \cdots, \boldsymbol{I}_N\})$$

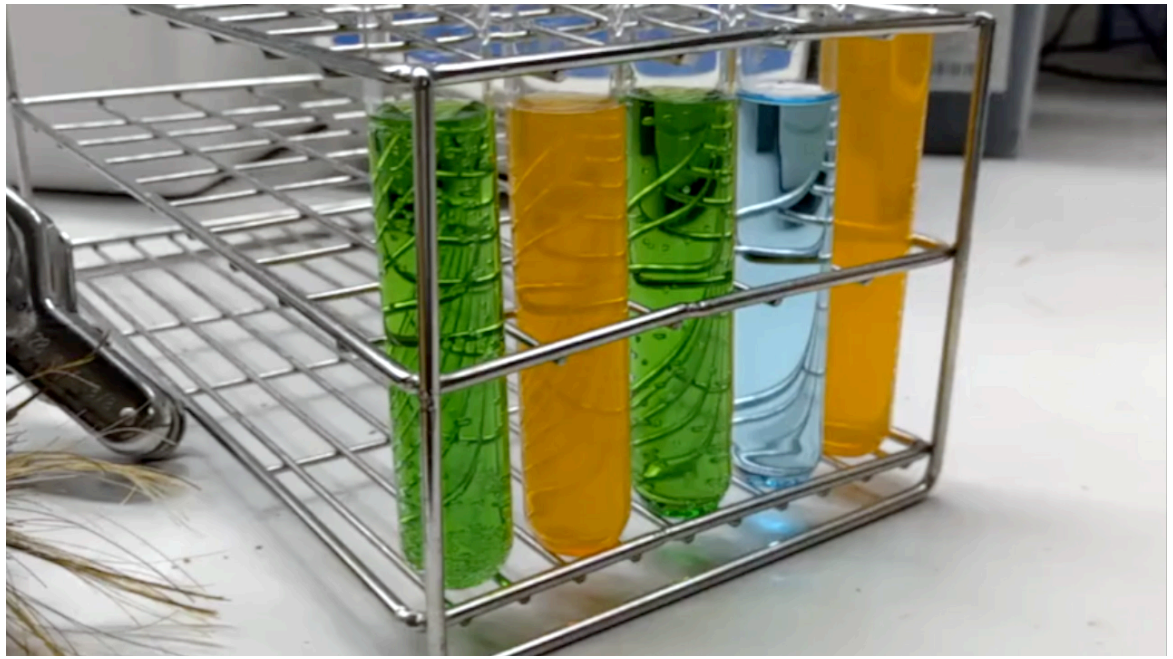
Ray Transformer: Universal Volume Rendering Approximator

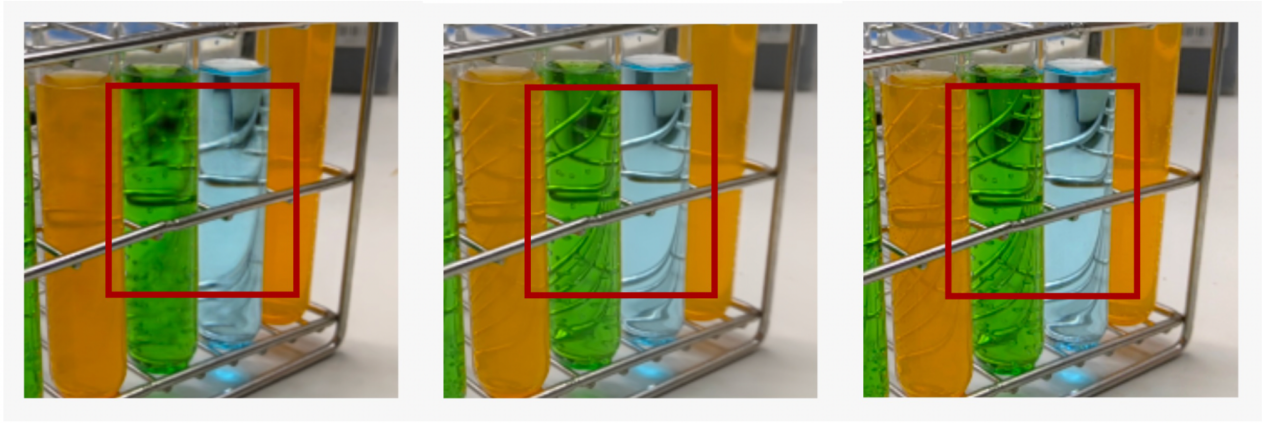
1. Cast a ray for each pixel on the target image plane.
2. Sample a sequence of points on the ray.
3. Get per-point feature vector by view transformer
4. Aggregate the sequence of 3D point feature by ray transformer.
5. Map the output feature to the RGB tuple.



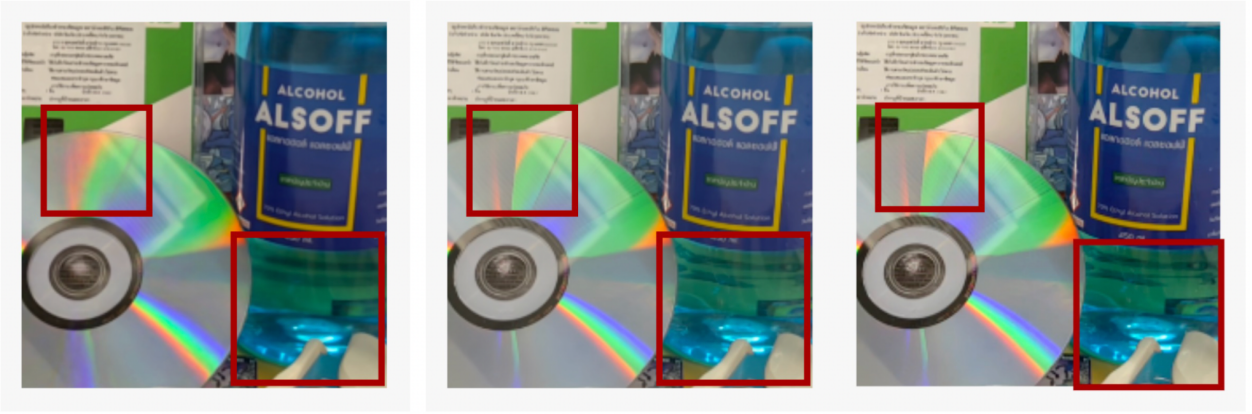
$$\boldsymbol{C}(\boldsymbol{r}) = \text{MLP} \circ \text{Mean} \circ \text{Ray-Transformer}(\mathcal{F}(\boldsymbol{o} + t_1\boldsymbol{d}, \boldsymbol{\theta}), \cdots, \mathcal{F}(\boldsymbol{o} + t_M\boldsymbol{d}, \boldsymbol{\theta}))$$

Videos on Challenging Scenes





(a) NeX (b) GNT (c) GT



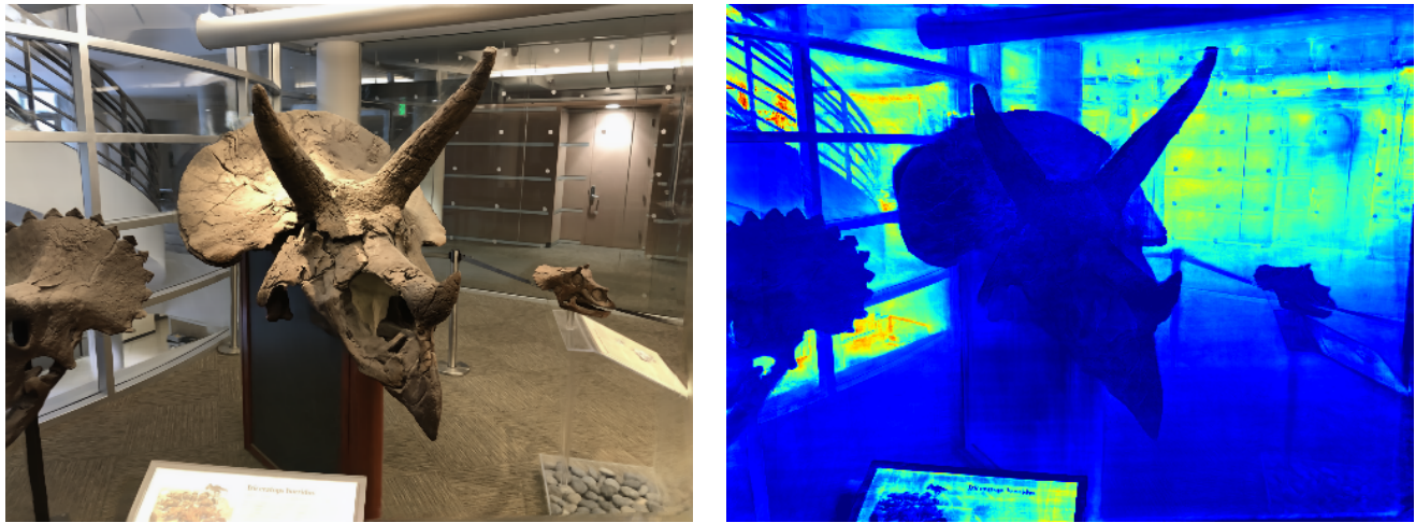
(a) NeX (a) GNT (b) GT

Visualization of Depth

Cue depth from learned attention weights from ray transformer.

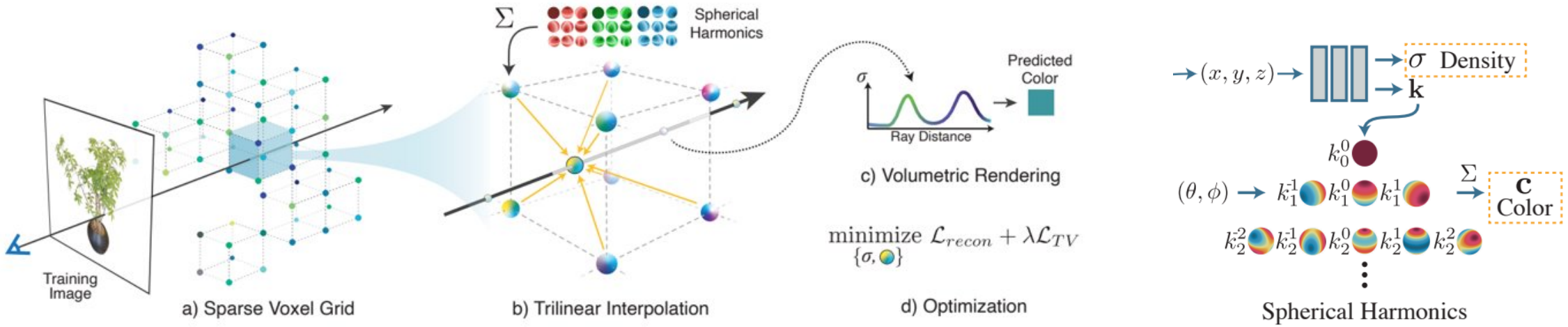
GNT learns to physically ground its attention maps, with no explicit depth supervision

(a) Trex



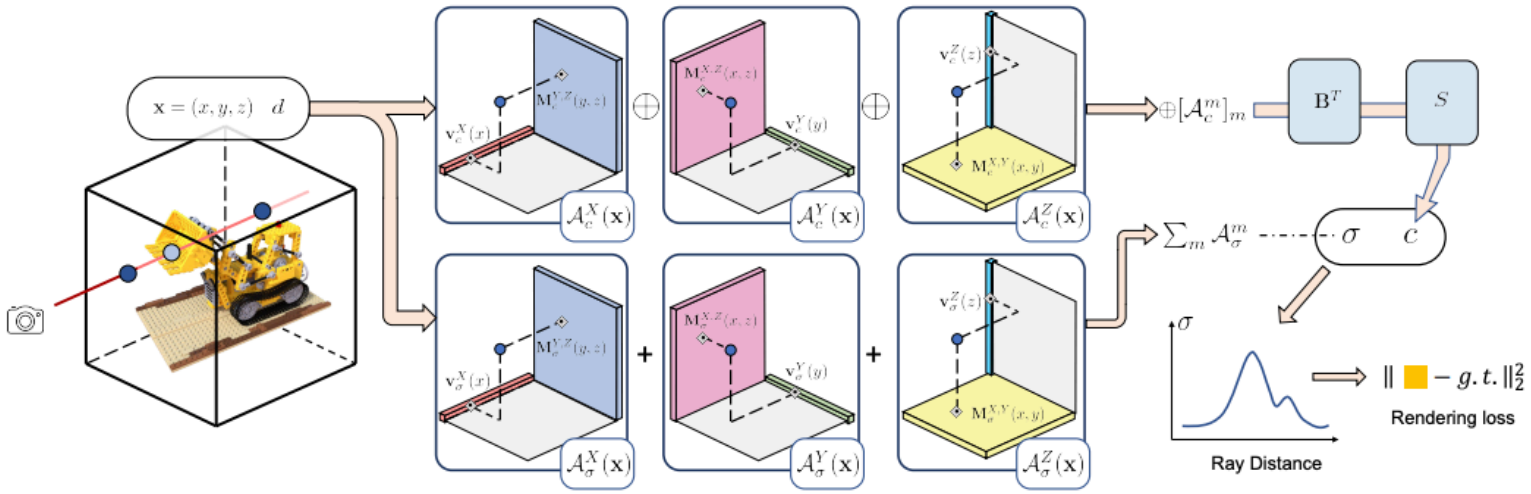
(b) Horns

Plenoxel: NeRF without Neural Networks



1. Drop neural networks and directly use voxels for radiance field optimization
2. Model view dependency via spherical harmonics
3. Reconstruct NeRFs in minutes

TensoRF: Tensorial Radiance Fields

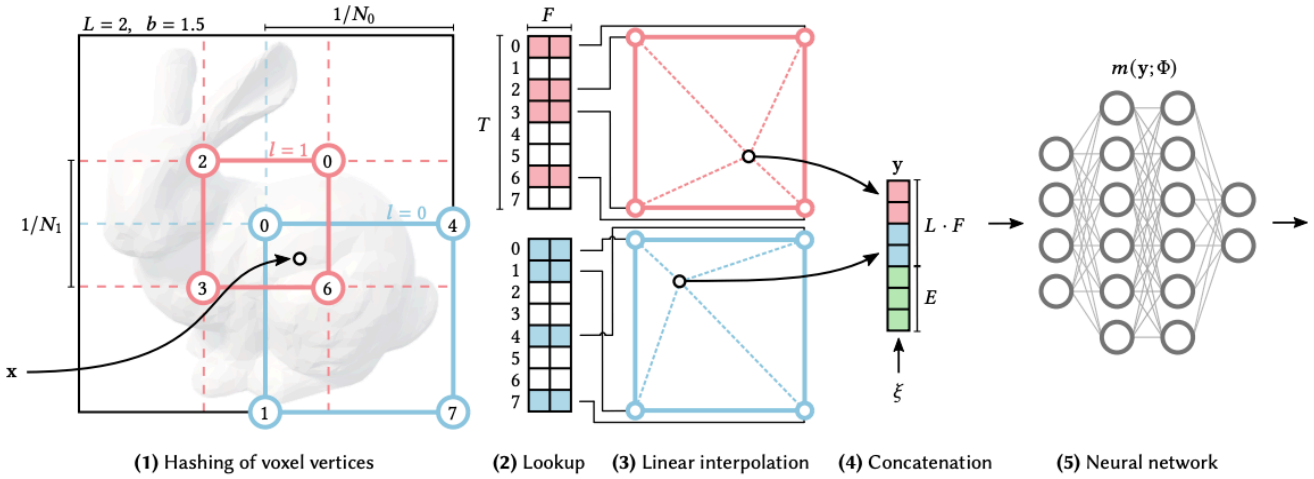


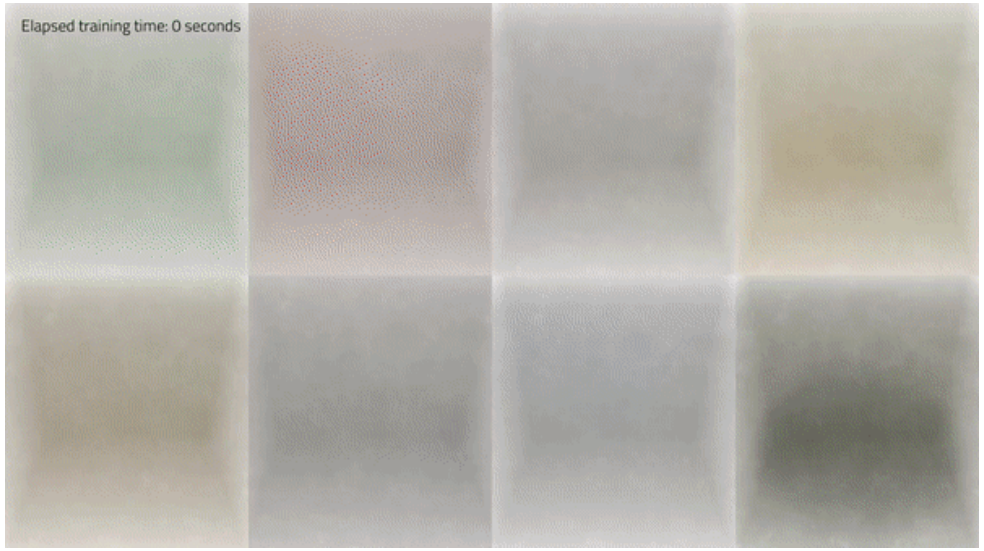
1. Decompose voxels by tensor factorization

$$\mathcal{G}_\sigma = \sum_{r=1}^{R_\sigma} \mathbf{v}_{\sigma,r}^X \circ \mathbf{M}_{\sigma,r}^{YZ} + \mathbf{v}_{\sigma,r}^Y \circ \mathbf{M}_{\sigma,r}^{XZ} + \mathbf{v}_{\sigma,r}^Z \circ \mathbf{M}_{\sigma,r}^{XY} = \sum_{r=1}^{R_\sigma} \sum_{m \in XYZ} \mathcal{A}_{\sigma,r}^m$$

2. Fast reconstruction in minutes.

Instant Neural Graphics Primitives with Hash Encoding



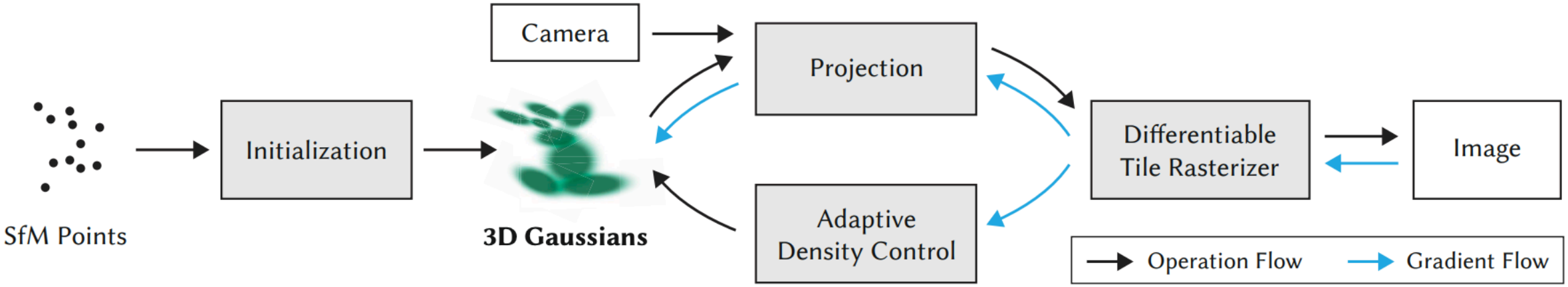


1. Compress voxel representation with a hash table (O(1) complexity)
2. Put the small hash table into the GPU cache!
3. Use a tiny MLP to achieve view dependency
4. **Fast reconstruction in seconds!**

https://nvlabs.github.io/instant-ngp/

3D Gaussian Splatting for Real-Time Radiance Field Rendering

$$G(x) = e^{-\frac{1}{2}(x)^T \Sigma^{-1}(x)} \qquad \Sigma' = JW\,\Sigma\,W^T J^T$$



1. Use 3D gaussian as the 3D scene approximation
    1. Mixture of Gaussian are universal approximator.
2. Projection of Gaussians can be computed analytically.
3. **Rasterization based rendering instead of ray-tracing based.**

https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

Beyond NeRFs: Magics implemented with NeRF

Outline:

- NeRF from Internet Images
- NeRF from few views(3-5)
- NeRF from single view
- NeRF editing

Neural Radiance Field - Pipeline

Given a set of sparse views of an object with known camera poses

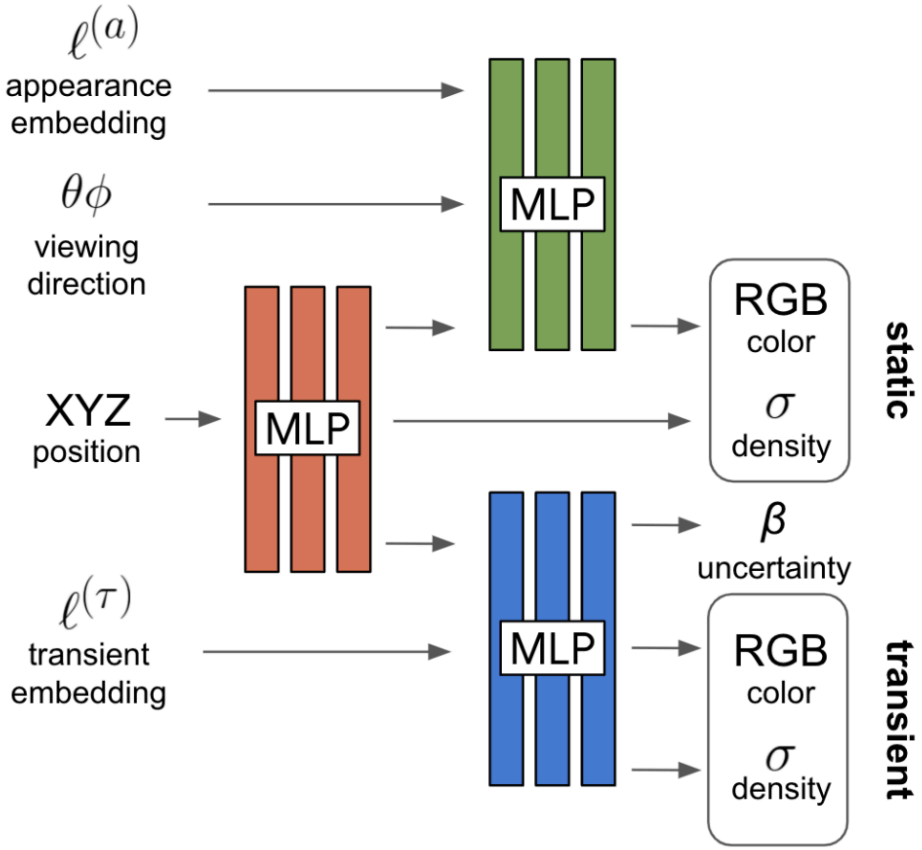
Optimize a NeRF model

3D reconstruction viewable from any angle

Can we train a NeRF from this?

NeRF in the wild (CVPR'21)

Figure 2: Example in-the-wild photographs from the Phototourism dataset [13] used to train NeRF-W. Due to variable illumination and post-processing (top), the same object's color may vary from image to image. In-the-wild photos may also contain transient occluding

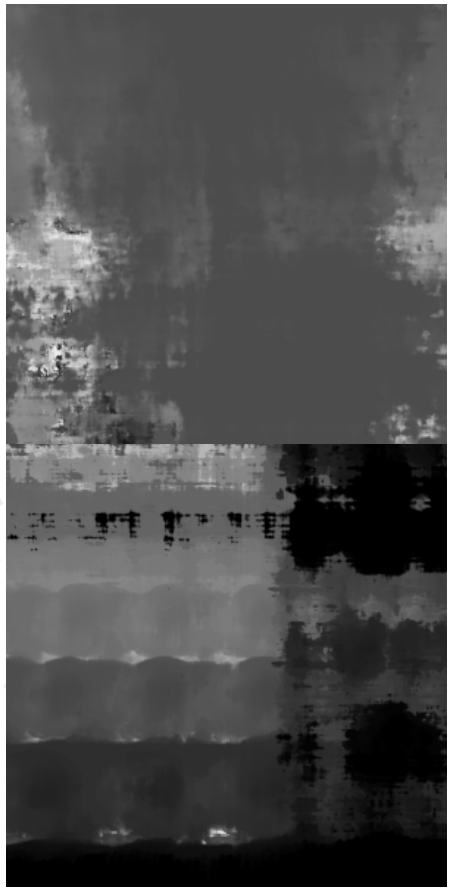


NeRF in the wild (CVPR'21)

Few-view NeRF: What if NeRF is trained with 3-5 viewpoints?

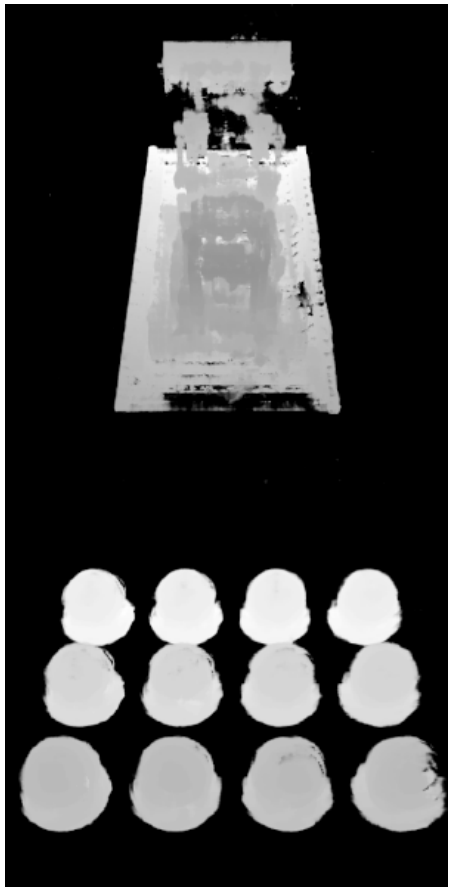
NeRF

NeRF(depth)



RegNeRF

RegNeRF(depth)

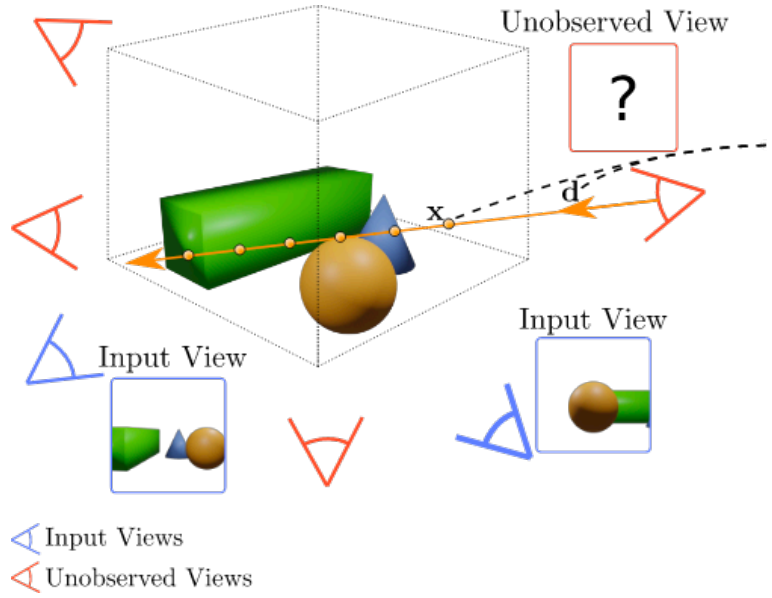


Main Idea: Provide regularization on novel views

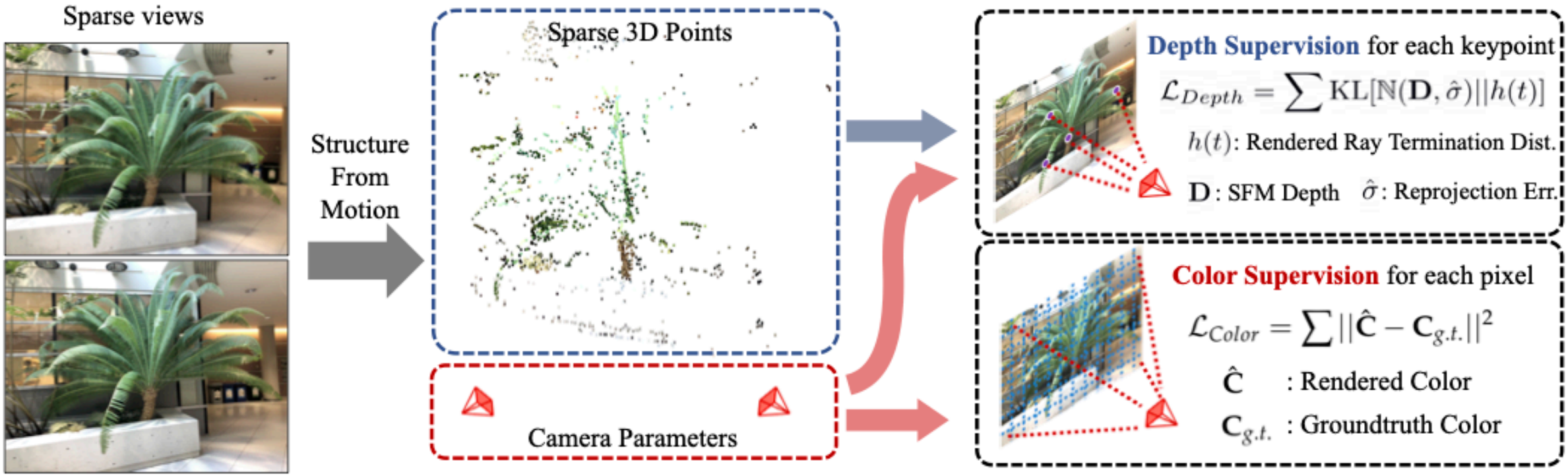
If NeRF is trained on available views only, the learned representation will collapse (overfit).

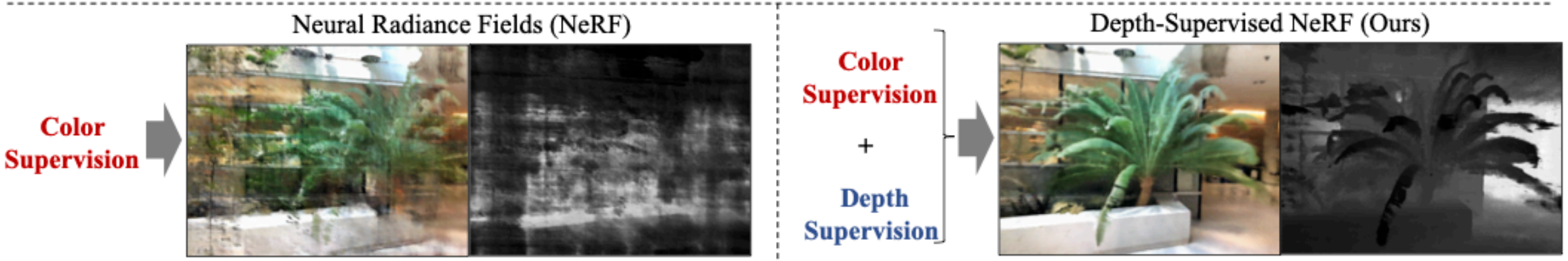
But how to train on novel views?
There's no ground truth annotation.

Instead of providing exact pixel supervision,
We can guide the novel views with priors.



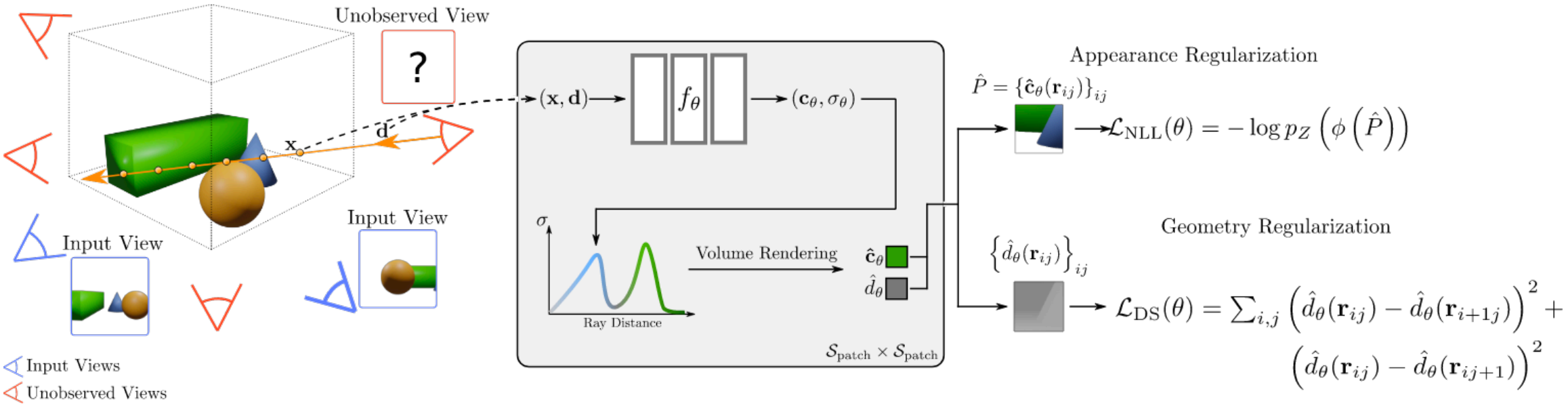
DSNeRF (CVPR'22): Depth Supervised NeRF



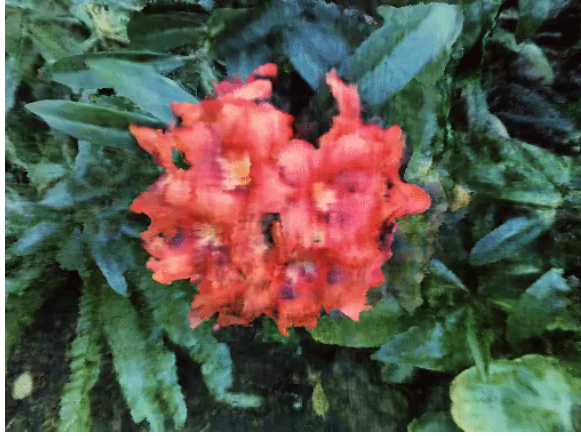


DietNeRF (ICCV'21): high-level semantics consistency

RegNeRF (CVPR'22): depth should be smooth



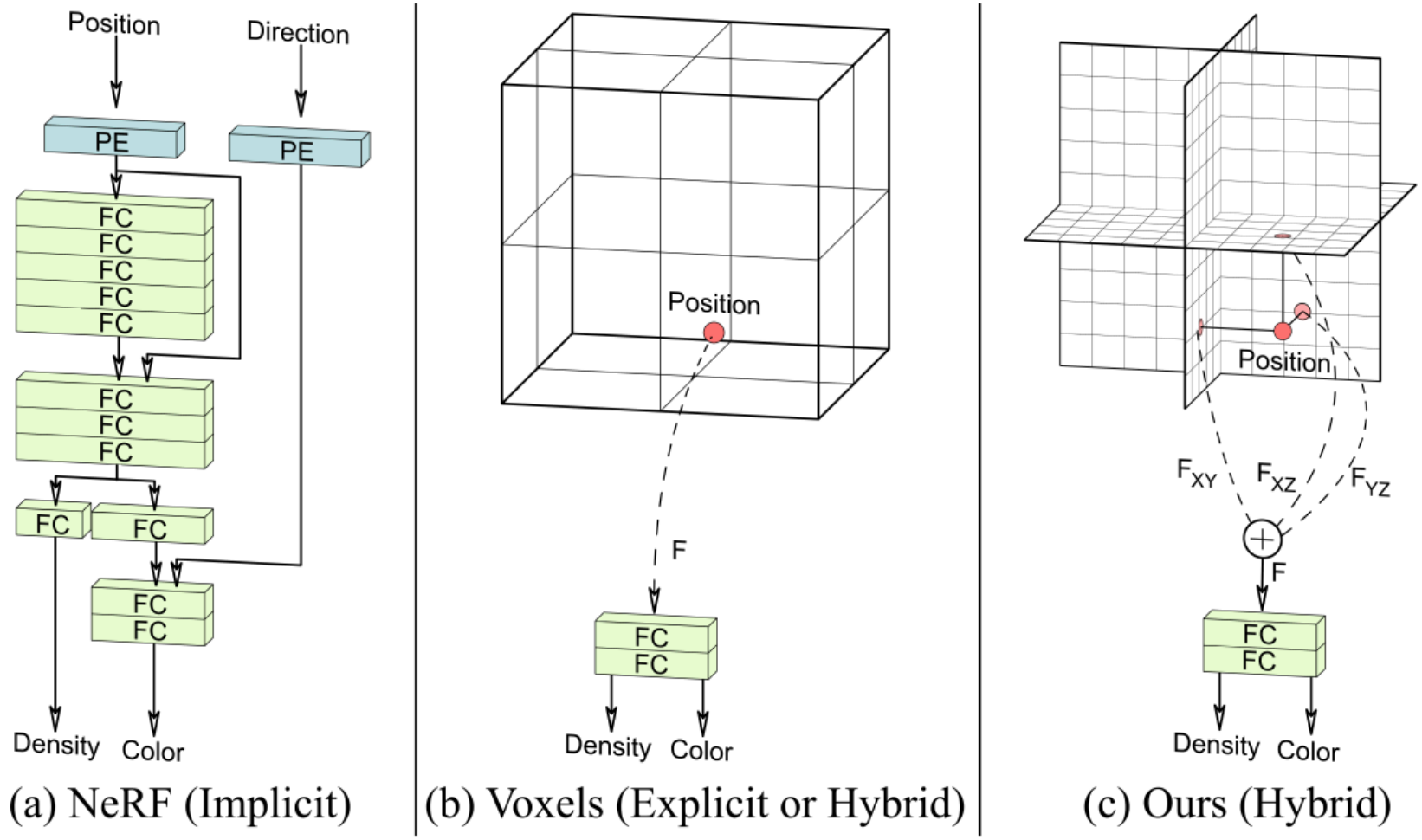
How to extend to single view setting?



- Cross scene fitting: Learn single view to 3D mapping using large dataset. Works for category-level only.
  E.g. Pixel-NeRF, LOL-NeRF

- Single scene fitting: Train with RGB(D) data and use additional priors for novel view renderings. Works for in-the-wild images but object-centric.
  E.g. SinNeRF, NeuralLift-360

EG3D (CVPR'22): 3D GAN to Generate Tri-planes



EG3D (CVPR'22): 3D GAN to Generate Tri-planes

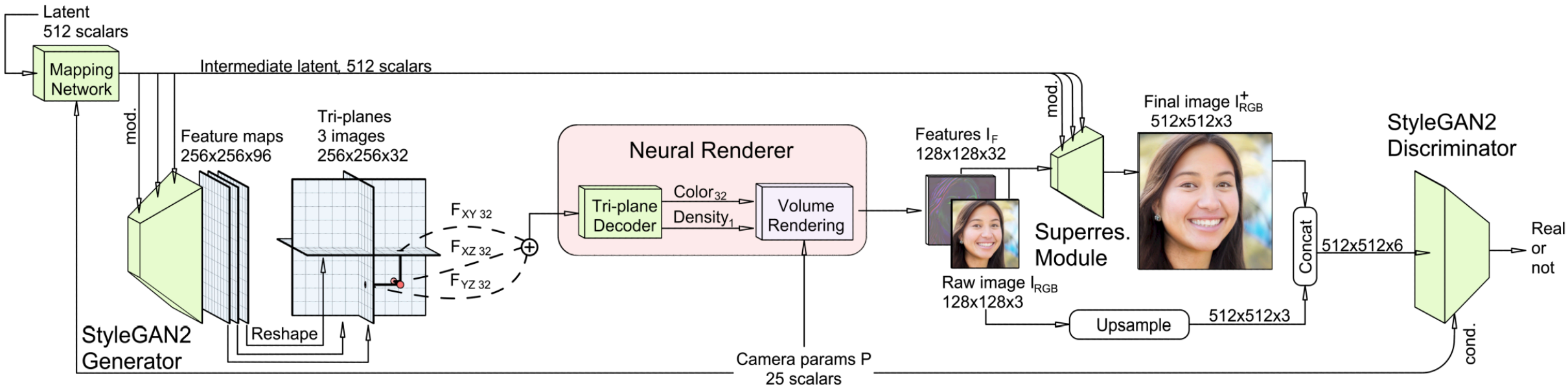
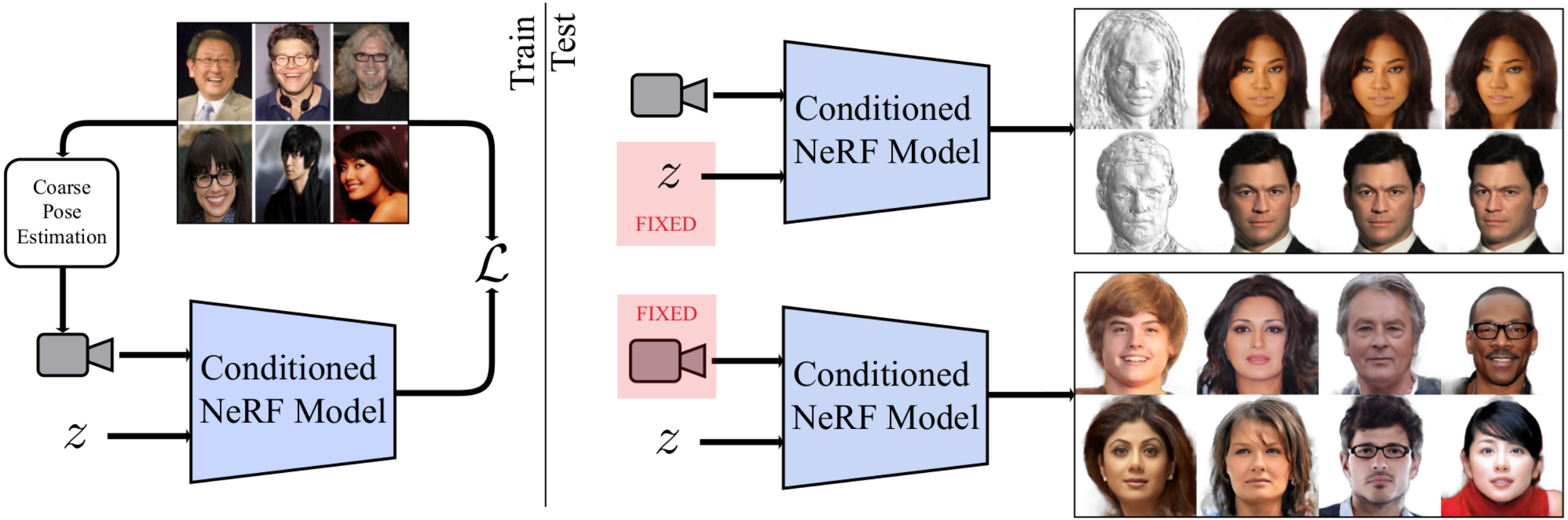
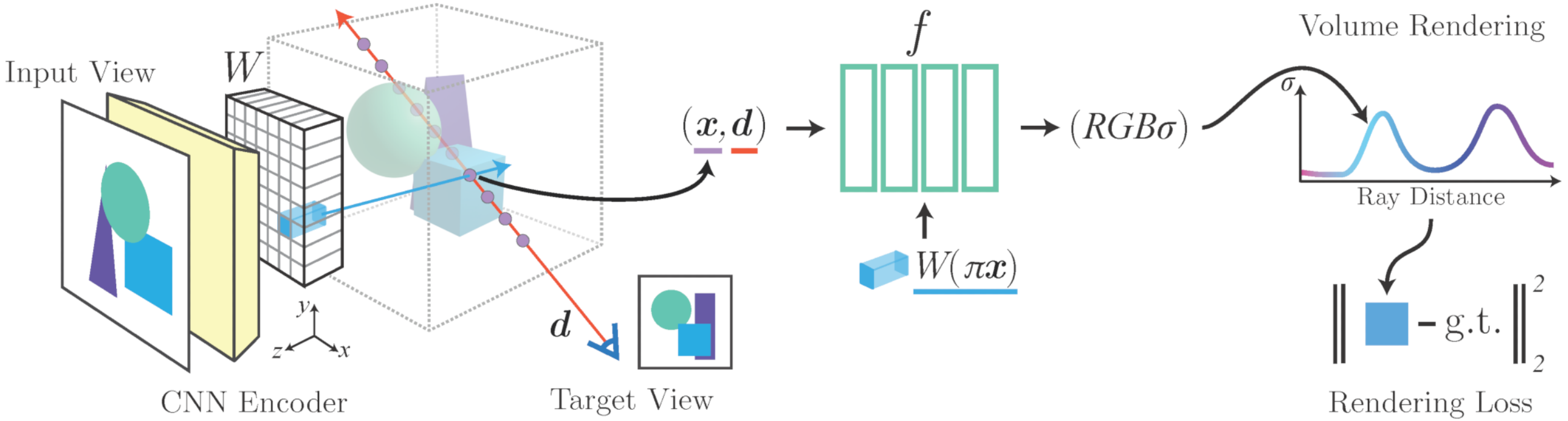


Figure 4. Our 3D GAN framework comprises several parts: a pose-conditioned StyleGAN2-based feature generator and mapping network, a tri-plane 3D representation with a lightweight feature decoder, a neural volume renderer, a super-resolution module, and a pose-conditioned StyleGAN2 discriminator with dual discrimination. This architecture elegantly decouples feature generation and neural rendering, allowing the use of a powerful StyleGAN2 generator for 3D scene generalization. Moreover, the lightweight 3D tri-plane representation is both expressive and efficient in enabling high-quality 3D-aware view synthesis in real-time.

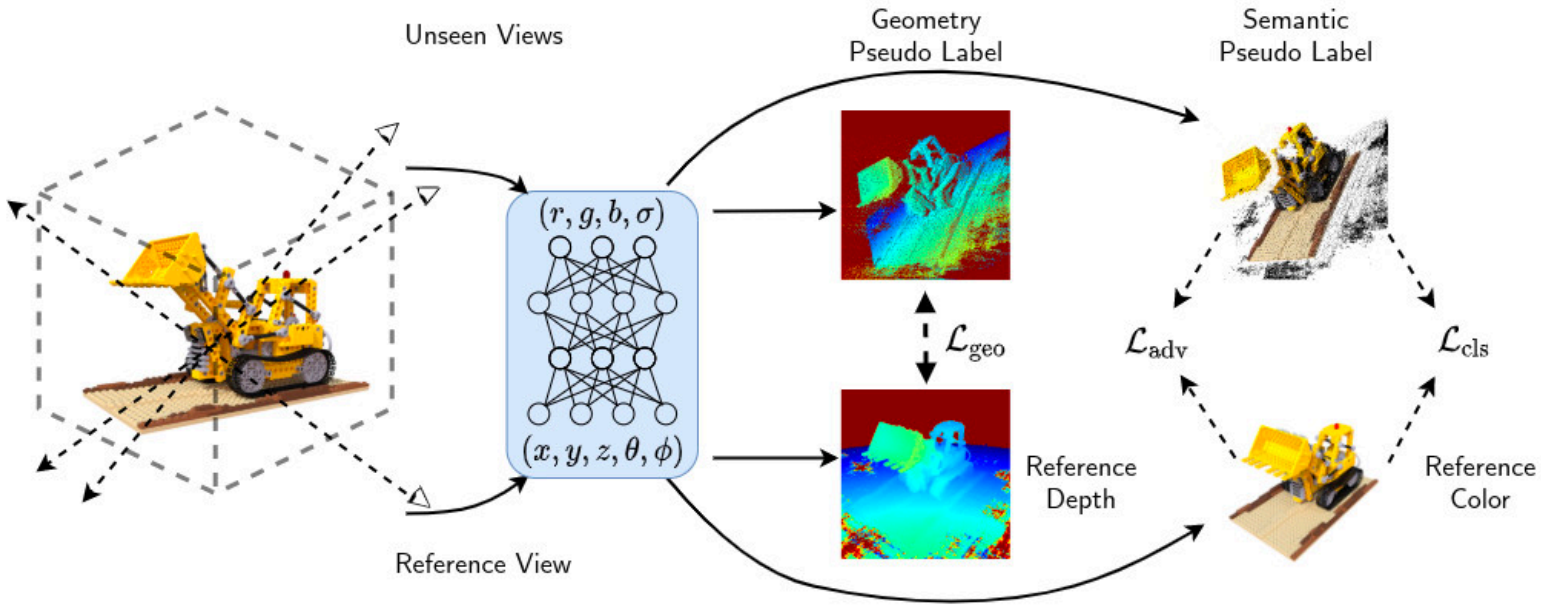
LOL-NeRF (CVPR'22): Conditional NeRF Generator



Pixel NeRF (CVPR'21): Using CNN to Extract Features



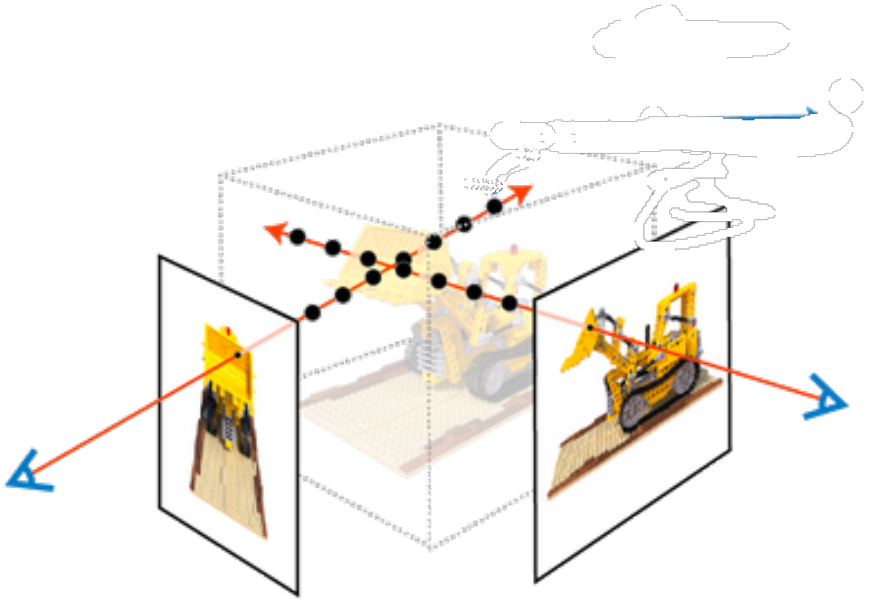
SinNeRF (ECCV'22): Training only on 1 reference view



- Propagate depth information using warping
- Maintain semantic consistency using DINO-ViT and GAN

Geometry Pseudo Label

- How to propagate depth information to novel views?
  - Unproject 2D depth on reference view back into 3D points
  - Project 3D points to 2D depth on novel views



$$p_j = K_{\text{unseen}} T (K_{\text{ref}}^{-1} Z_i p_i),$$

Semantic Pseudo Label

- How to propagate RGB information?

Major contents should be similar between reference view and novel views

ViT has been proven to be an expressive semantic prior

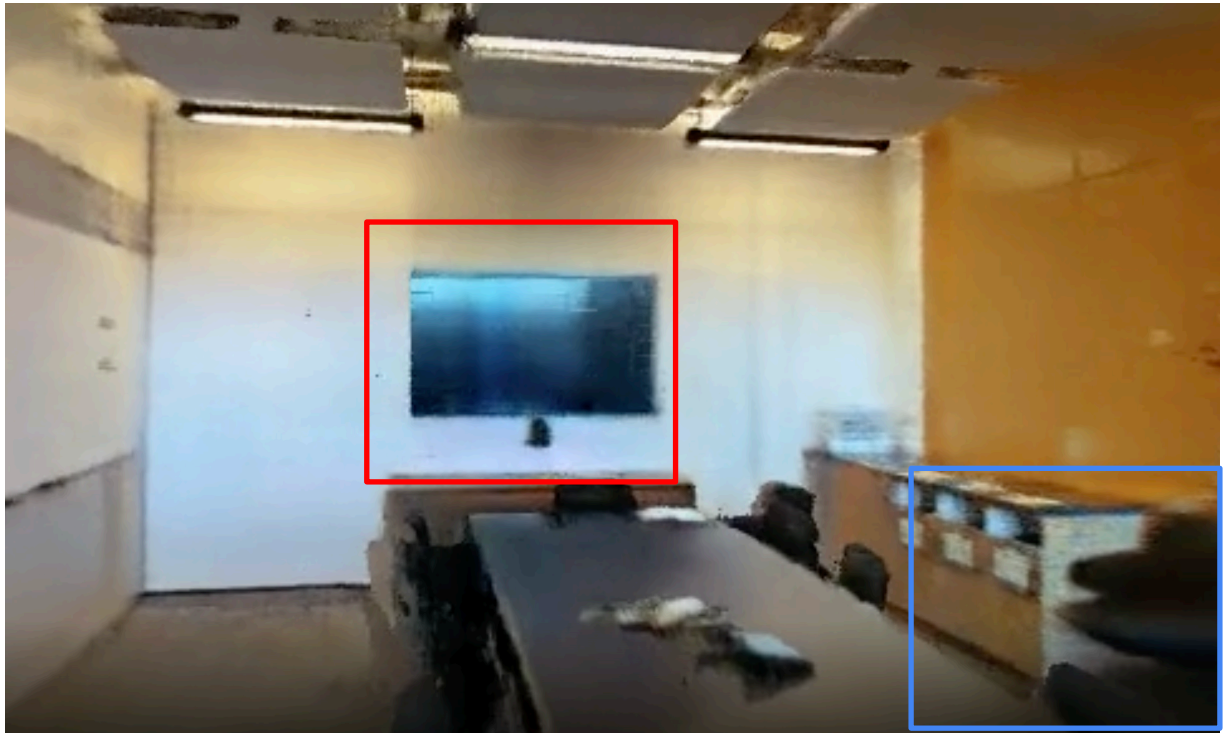
The [CLS] token from ViT output is a global representation for the input

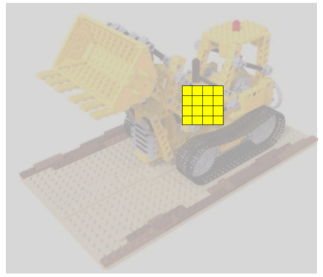
$$\mathcal{L}_{\text{cls}} = ||f_{\text{vit}}(A) - f_{\text{vit}}(B)||^2$$

Here, A, B are patches from reference view and novel views

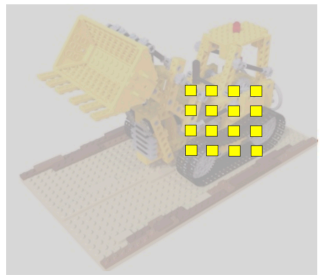
[CLS] token only works when seeing the overall scene

- Red box and blue box don’t have a similar [CLS] token
- We have to cover the whole scene, but rendering is expensive?
- Strided Sampling

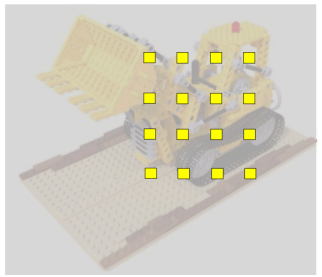




Sampling Stride = 1



Sampling Stride = 2



Sampling Stride = 4

Large stride get blurry results

- but [CLS] loss doesn't help when we reduce the stride
- Adding a discriminator can help! (NeRF is the generator)

$$\mathcal{L}_{\mathrm{D}} = \mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{data}}\,(\boldsymbol{x})}\left[f_D(-D(T(\boldsymbol{x})))\right] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[f_D(D(T(G(\boldsymbol{z}))))\right],$$
$$\mathcal{L}_{\mathrm{G}} = \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[f_G(-D(T(G(\boldsymbol{z}))))\right],$$
$$\mathcal{L}_{\mathrm{adv}} = \mathcal{L}_{\mathrm{D}} + \mathcal{L}_{\mathrm{G}},$$

- But there's only real sample available?
- T is differential augmentation that help data efficient GAN training

Text to 3D: Optimize 3D model so random 2D views look good

DreamFields (CVPR'22)

Bouquest of flowers sitting in a clear glass vase

Image_text_loss = CLIP embedding similarity

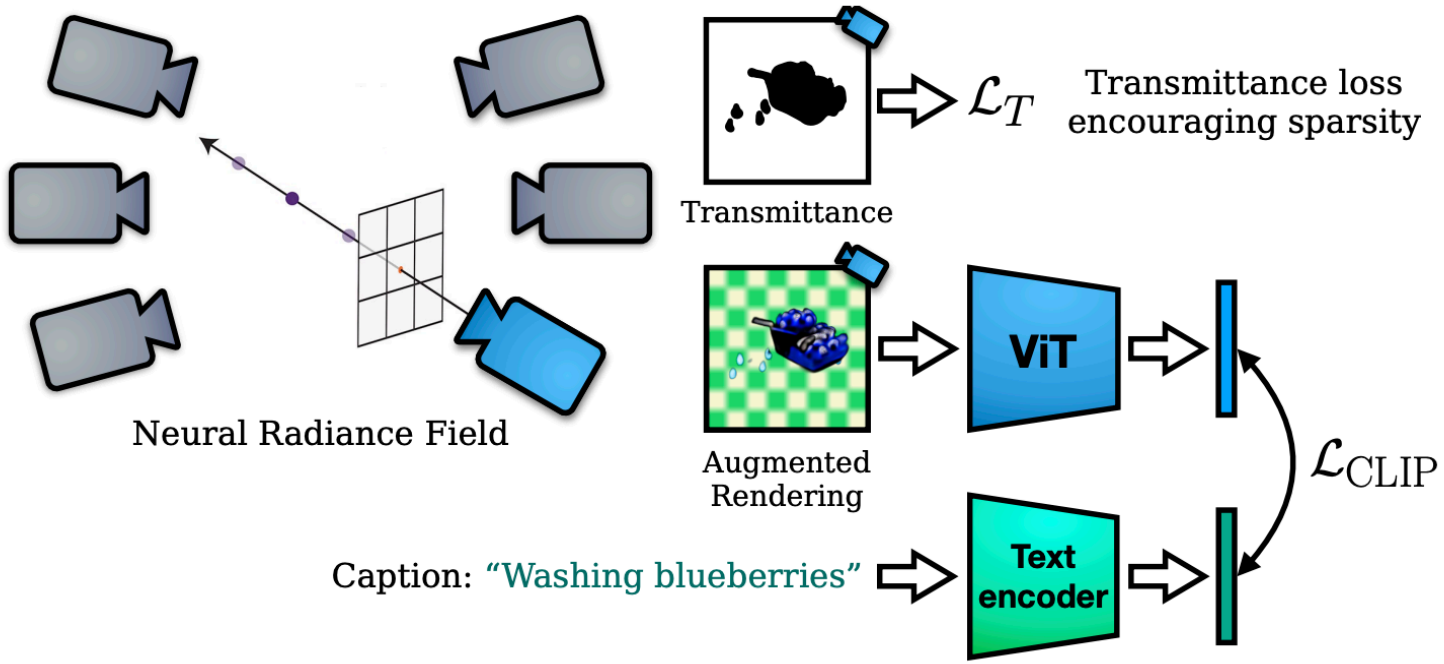


Figure 1. Given a caption, we learn a **Dream Field**, a continuous volumetric representation of an object's geometry and appearance learned with guidance from a pre-trained model. We optimize the Dream Field by rendering images of the object from random camera poses that are scored with *frozen* pre-trained **image** and **text** encoders trained on web images and alt-text. 2D views share the same underlying radiance field for consistent geometry.

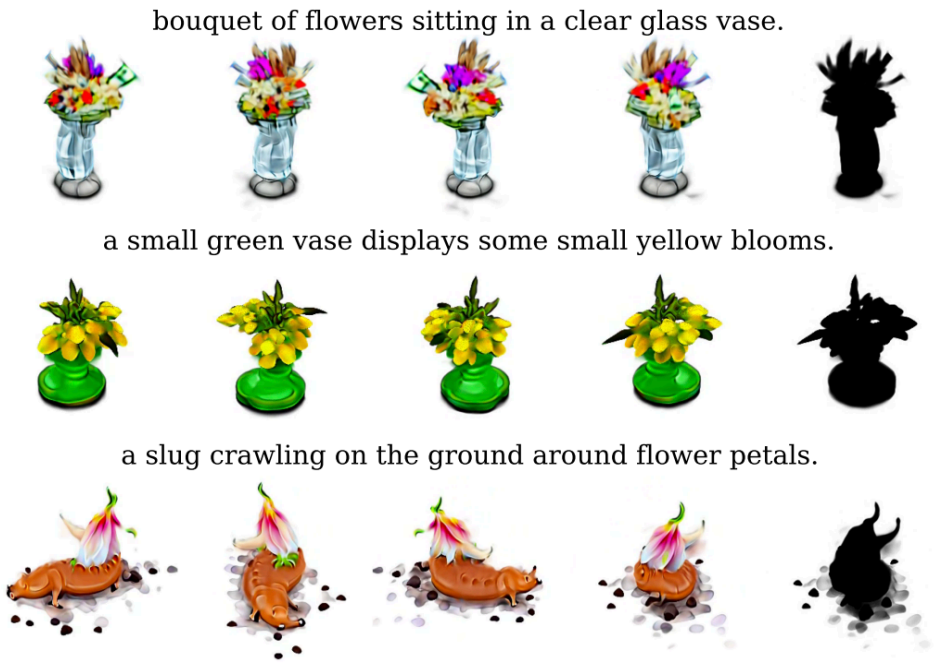


Figure 2. Example Dream Fields rendered from four perspectives. On the right, we show transmittance from the final perspective. We create diverse outputs using the compositionality of language; these captions from MSCOCO describe three flower arrangements with different properties like context and color.

DreamFusion (ICLR'23)

Image_text_loss = Score Distillation (Imagen)

Gradient of the distillation loss

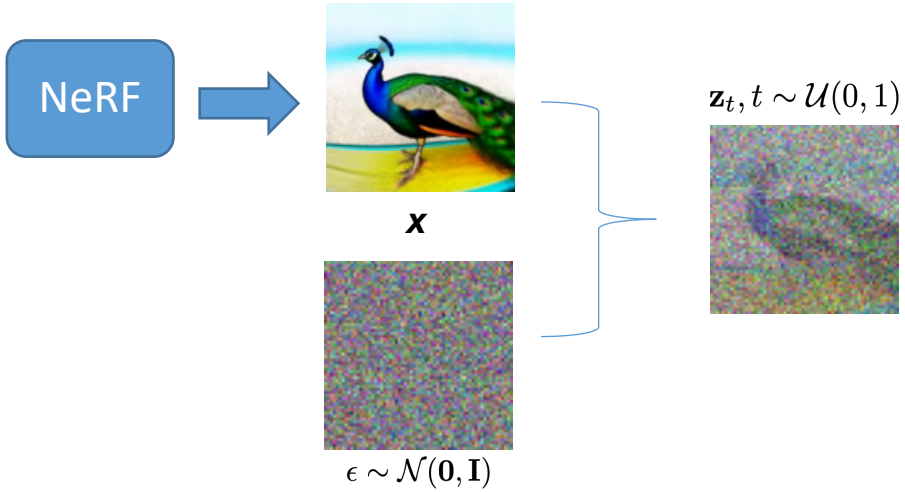
$$\nabla_\theta \mathcal{L}_{\text{SDS}}(\phi, [redacted]) \triangleq \mathbb{E}_{t,\epsilon}\left[w(t)\left(\hat{\epsilon}_\phi(\mathbf{z}_t; y, t) - \epsilon\right)\frac{\partial \mathbf{x}}{\partial \theta}\right]$$

NeRF

x

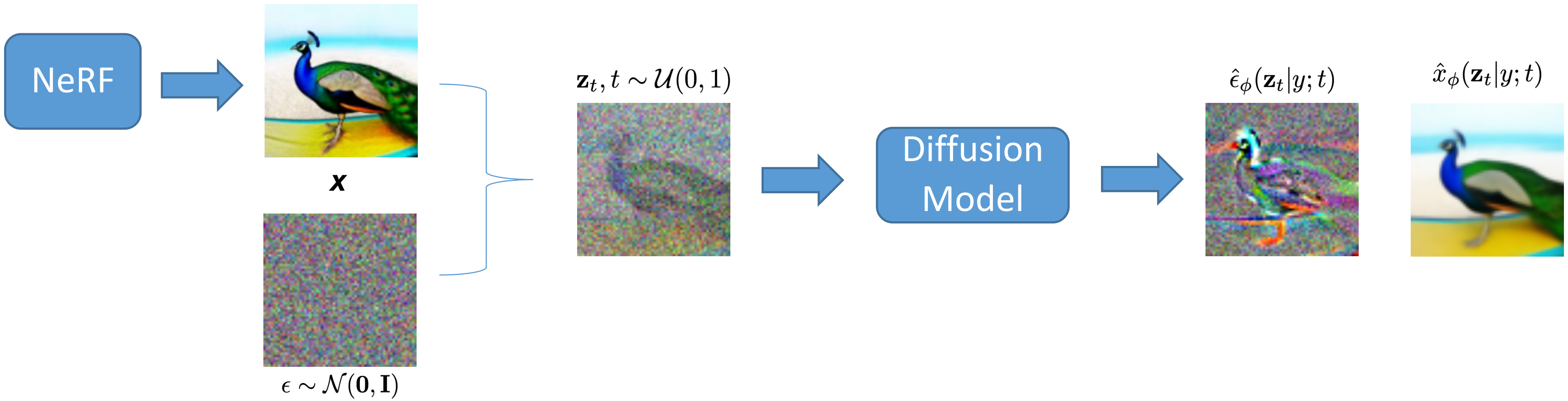
Gradient of the distillation loss

$$\nabla_\theta \mathcal{L}_{\text{SDS}}(\phi, \mathbf{x} = g(\theta)) \triangleq \mathbb{E}_{t,\epsilon}\left[w(t)\left(\hat{\epsilon}_\phi(\mathbf{z}_t; y, t) - \epsilon\right)\frac{\partial \mathbf{x}}{\partial \theta}\right]$$



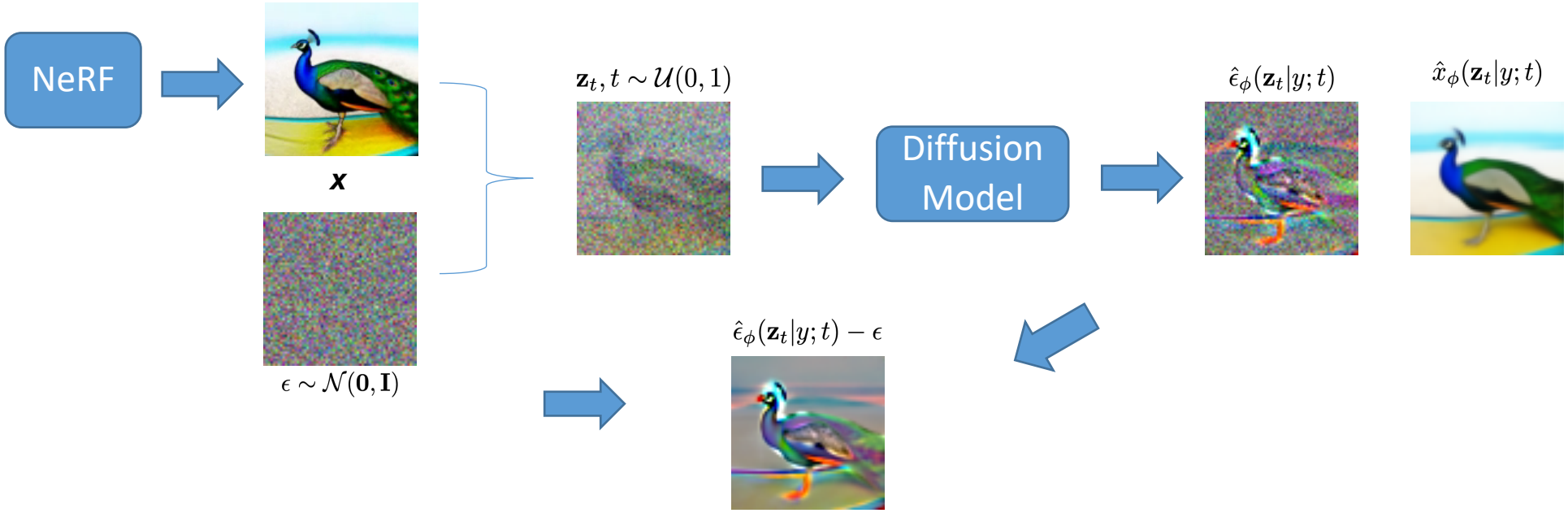
Gradient of the distillation loss

$$\nabla_\theta \mathcal{L}_{\mathrm{SDS}}(\phi, \mathbf{x} = g(\theta)) \triangleq \mathbb{E}_{t,\epsilon}\left[w(t) \left(\text{[illegible]} - \epsilon \right) \frac{\partial \mathbf{x}}{\partial \theta} \right]$$



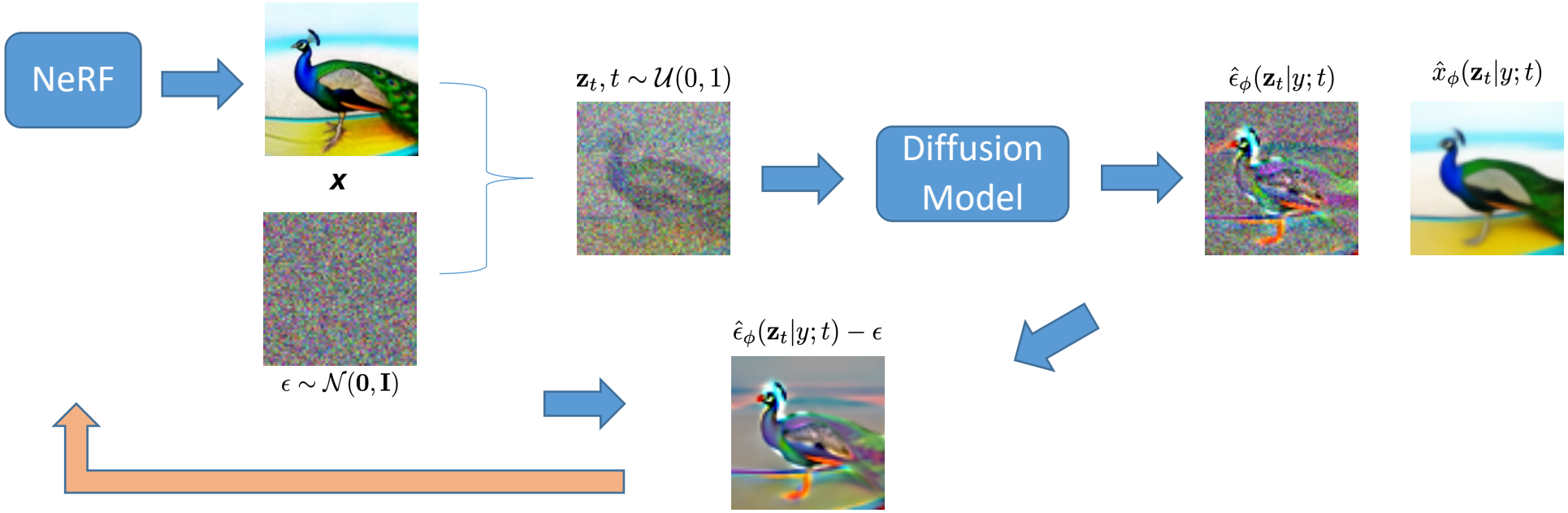
Gradient of the distillation loss

$$\nabla_\theta \mathcal{L}_{\text{SDS}}(\phi, \mathbf{x} = g(\theta)) \triangleq \mathbb{E}_{t,\epsilon}\left[w(t) \left(\hat{\epsilon}_\phi(\mathbf{z}_t; y; t) - \epsilon \right) \frac{\partial \mathbf{x}}{\partial \theta} \right]$$

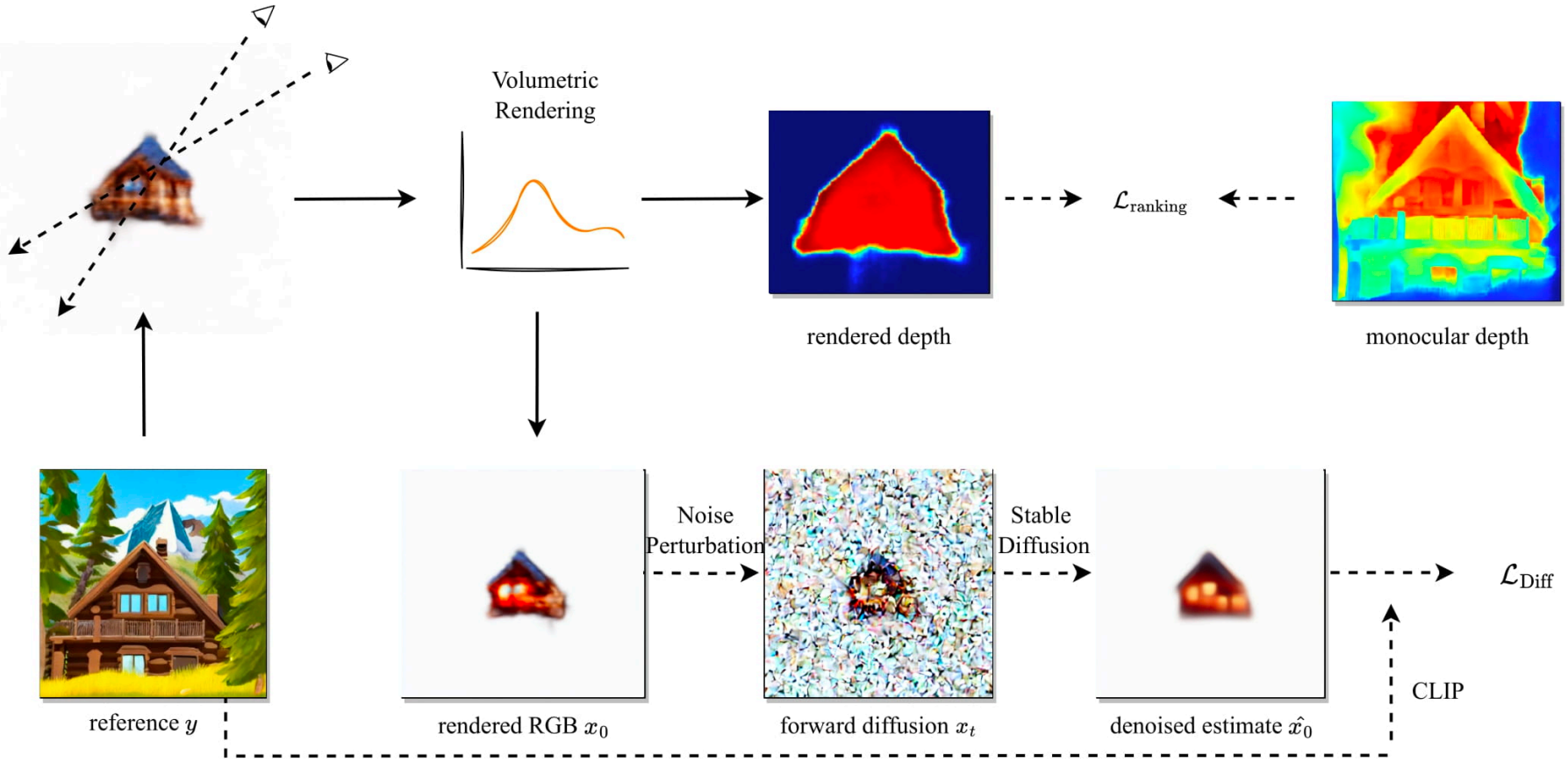


Gradient of the distillation loss

$$\nabla_\theta \mathcal{L}_{\text{SDS}}(\phi, \mathbf{x} = g(\theta)) \triangleq \mathbb{E}_{t,\epsilon}\left[w(t)\left(\hat{\epsilon}_\phi(\mathbf{z}_t; y, t) - \epsilon\right)\frac{\partial \mathbf{x}}{\partial \theta}\right]$$

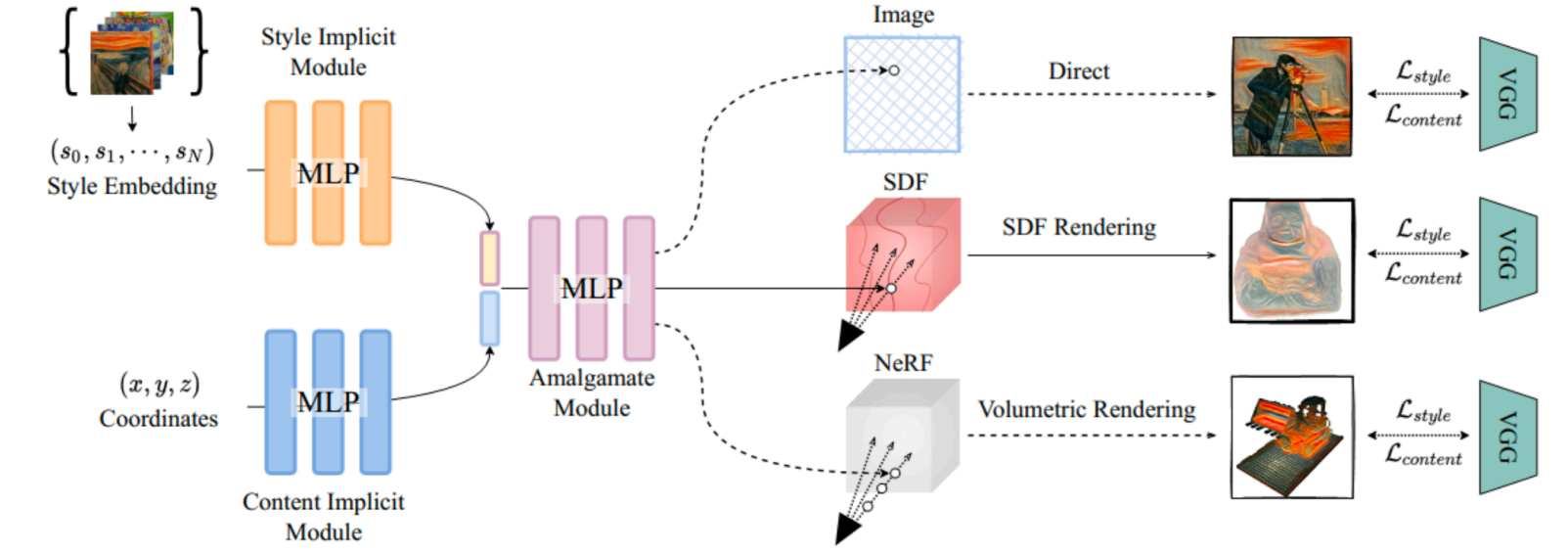


NeuralLift-360 (CVPR'23): image-to-3D in the wild



3D style transfer

INS (ECCV'22)



The main pipeline of unified implicit neural stylization (INS) framework and its components. We took NeRF with the proposed INS, for example, it inputs with implicit coordinates alone with ray directions and style implicit conditional code. Style Implicit Module (SIM) and Content Implicit Module (CIM) are used to extract conditional implicit style features and implicit scene features. Amalgamate Module (AM) is applied to fuse features in the two spaces. An implicit rendering step is applied on the top of AM(i.e. Volume rendering for NeRF, surface rendering for SDF). VGG used to generate style supervision is omitted in this figure for simplicity.

INS (ECCV'22)

SIREN [70]

SIREN+**INS**

Style Image

SDF [83]

SDF+**INS**

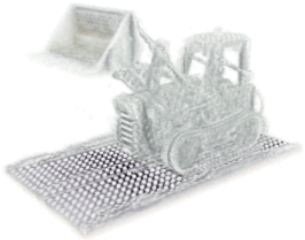
Style Image

NeRF [53]

Style Image

NeRF+**INS**

NeRF+**INS**



NeRF+**INS**

NeRF+**INS**

Instruct-NeRF2NeRF: Editing 3D Scenes with Instructions

Original

"Turn the bear into a panda"

Summary

- NeRF is a powerful 3D representation. Can be considered as a container supporting various 3D applications.
- NeRF store 3D information in the MLP parameters, editing directly is hard but we can edit via optimization.
- Training NeRF using one or few view is challenging but achievable with various priors.

The Rising Field Welcomes YOU!

The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering