3D-aware Synthesis (part II)

Jun-Yan Zhu

16-726, Spring 2025

Many slides from Eric Chan

© EG3D [Chan et al., 2022]

NeRF (neural radiance fields):
Neural networks as a *volume* representation, using volume rendering to do view synthesis. $(x, y, z, \theta, \phi) \rightarrow color, opacity$

Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

Representing a scene as a continuous 5D function

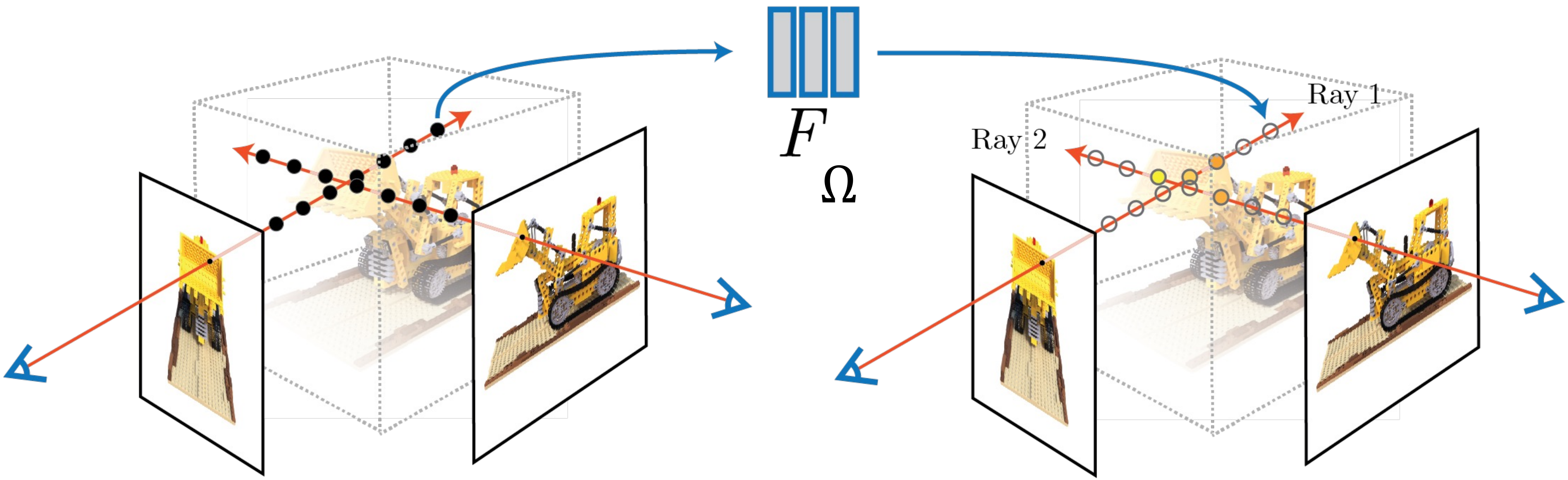
$$(\underbrace{x, y, z}_{\text{Spatial location}}, \underbrace{\theta, \phi}_{\text{Viewing direction}}) \longrightarrow F_{\Omega} \longrightarrow (\underbrace{r, g, b}_{\text{Output color}}, \underbrace{\sigma}_{\text{Output density}})$$

F_{Ω}

Fully-connected neural network
9 layers,
256 channels

Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

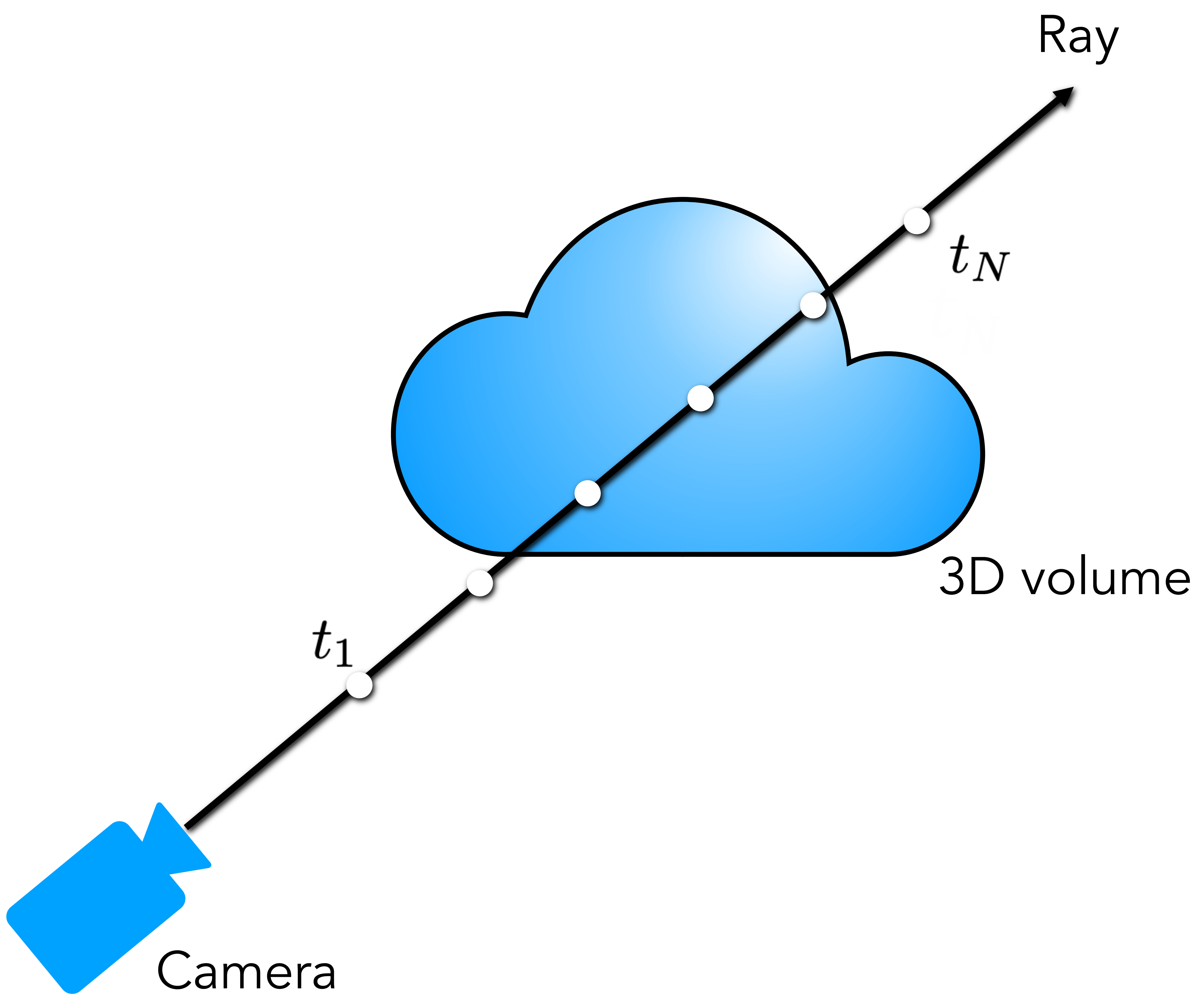
Generate views with traditional volume rendering



Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

Generate views with traditional volume rendering

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

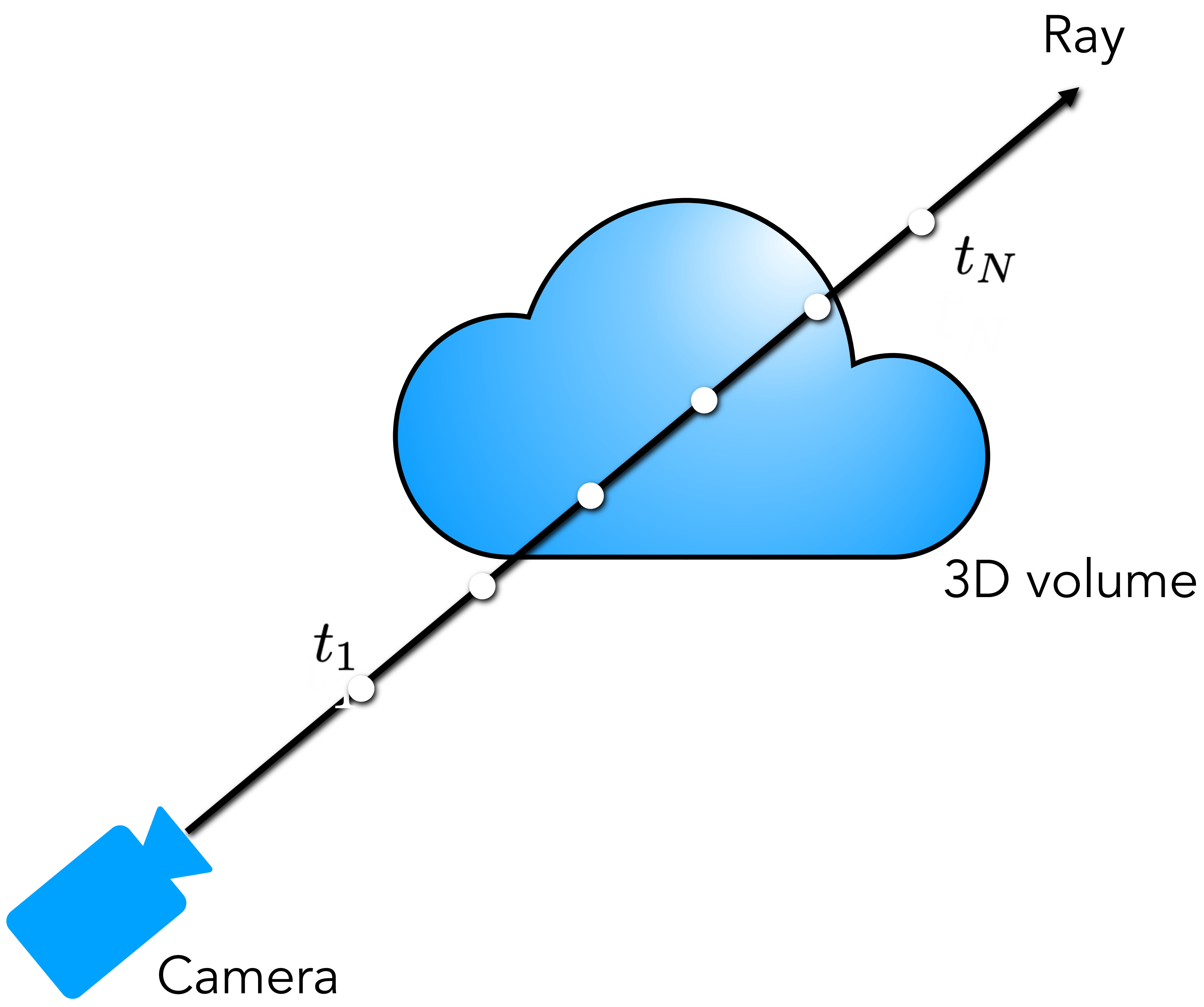


Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

Generate views with traditional volume rendering

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

$$C \approx \sum_{i=1}^{N} \underbrace{T_i \alpha_i}_{\text{weights}} \underbrace{c_i}_{\text{colors}}$$



Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

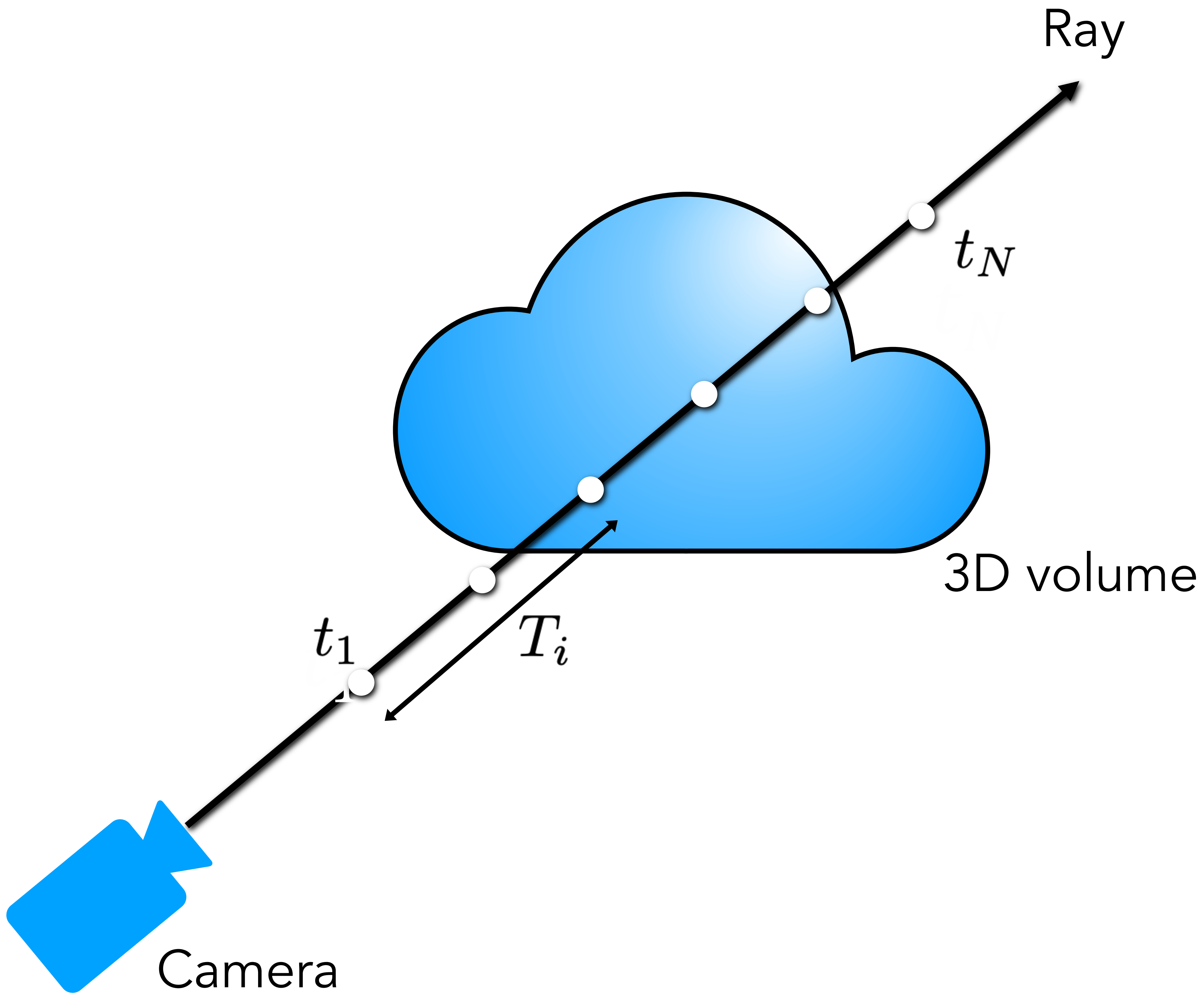
Generate views with traditional volume rendering

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

$$C \approx \sum_{i=1}^{N} \underbrace{T_i \alpha_i}_{\text{weights}} \underbrace{c_i}_{\text{colors}}$$

How much light is blocked earlier along ray:

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



Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

Generate views with traditional volume rendering

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

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

weights

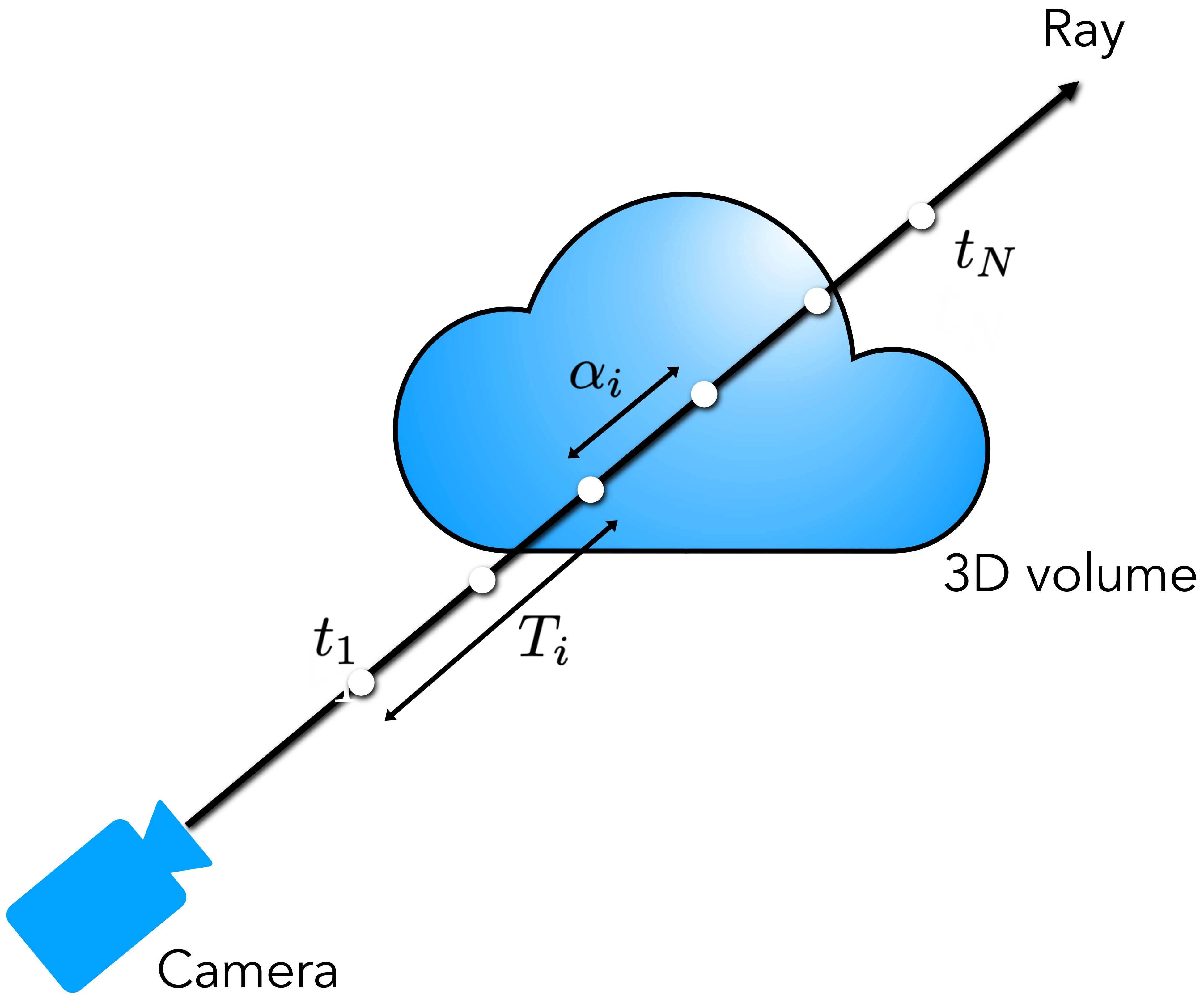
colors

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*:

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



Sigma parametrization for continuous opacity

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

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

weights

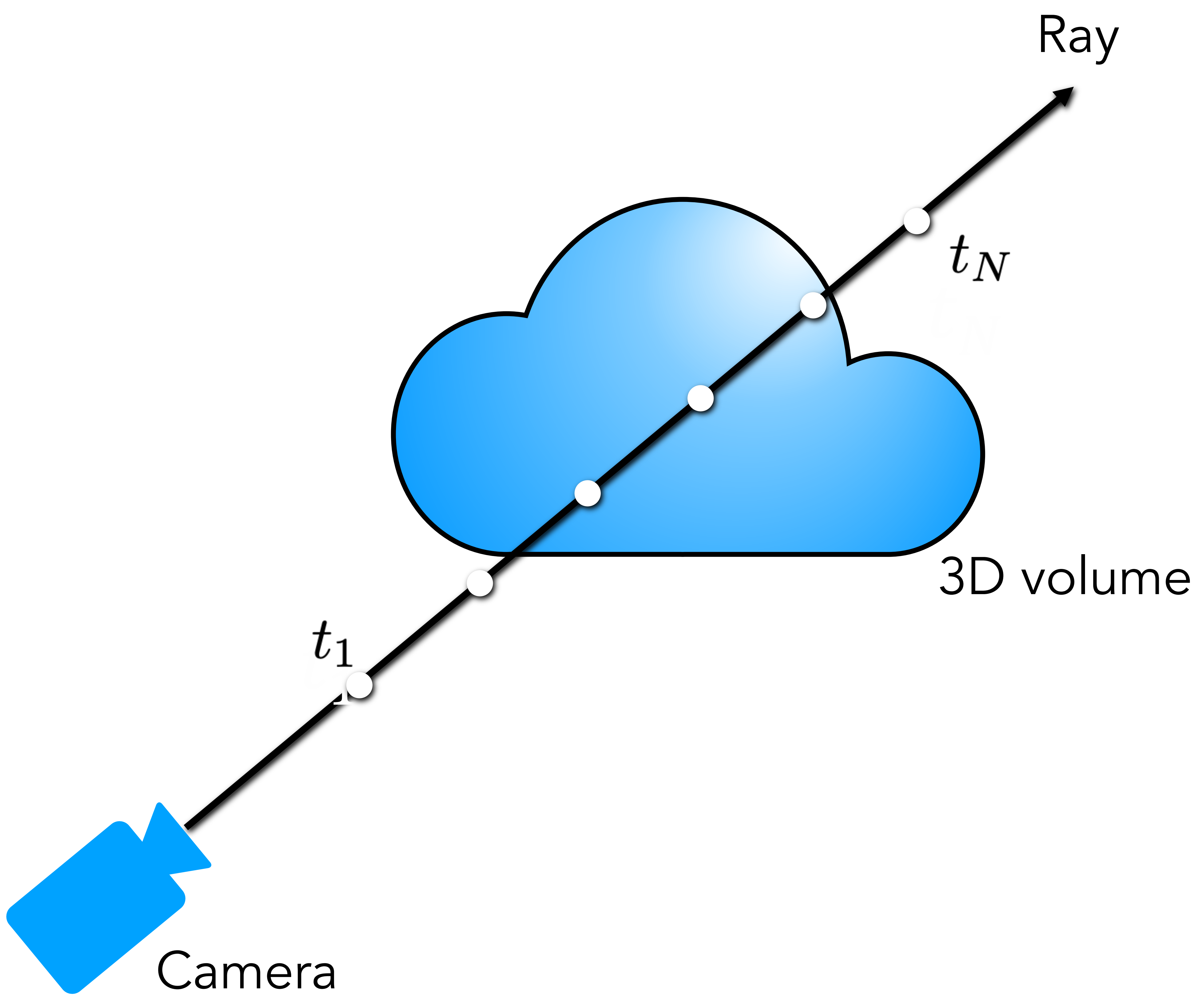
colors

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*:

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



Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

Effective resolution is tied to distance between samples

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

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

weights

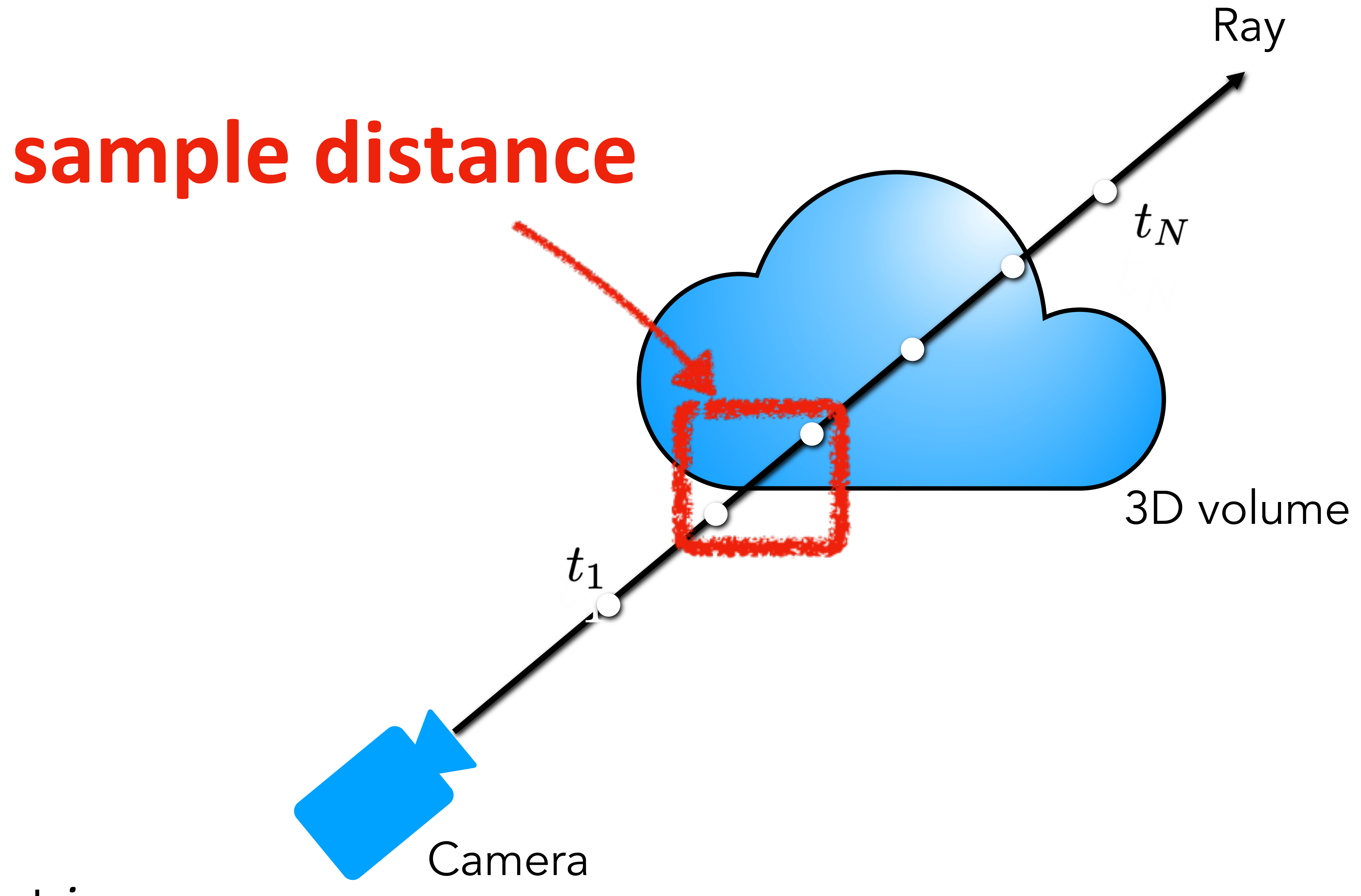
colors

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*:

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



Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

Volume rendering is trivially differentiable

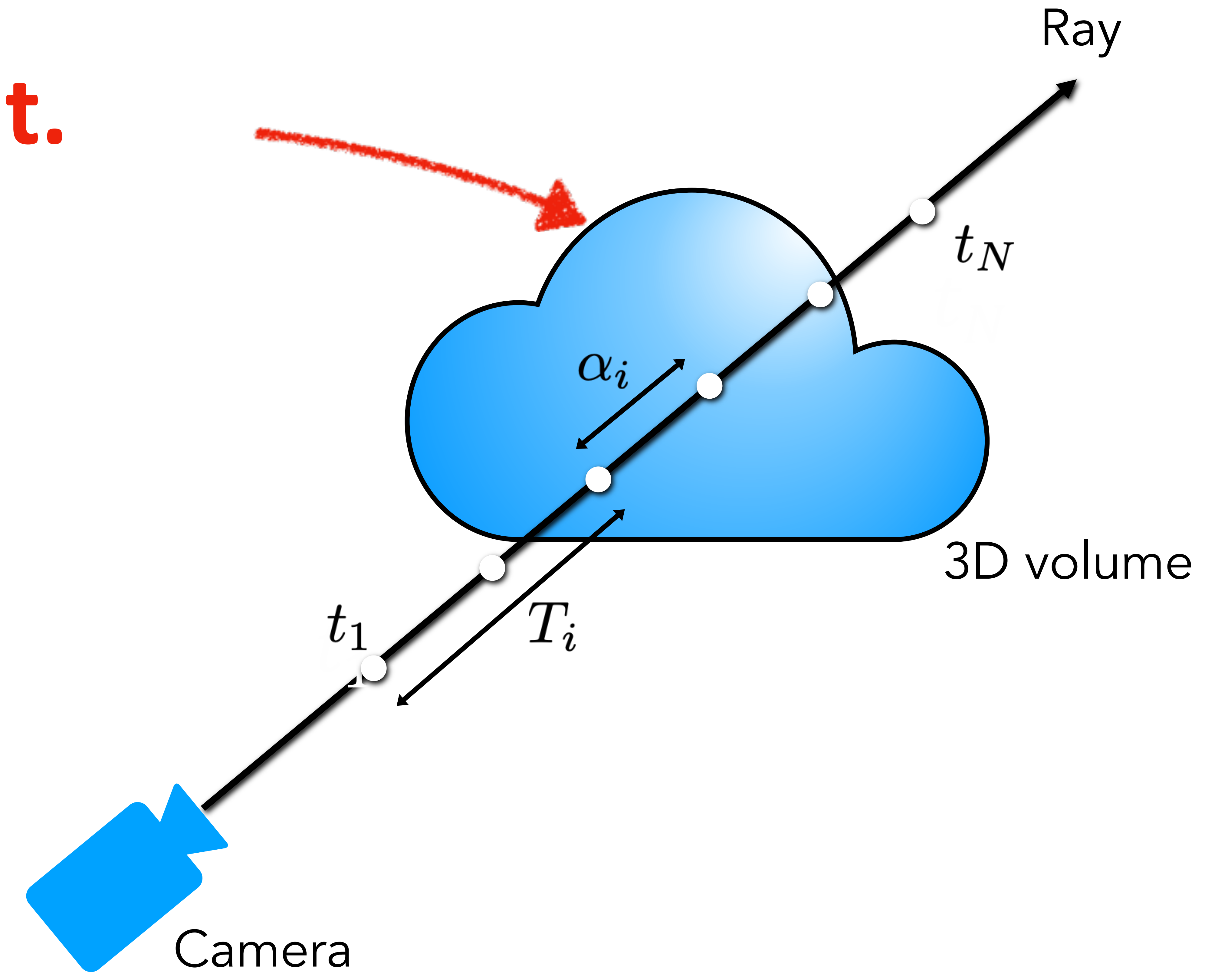
Rendering model for ray r(t) = o + td:

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

differentiable w.r.t.

weights

colors



How much light is blocked earlier along ray:

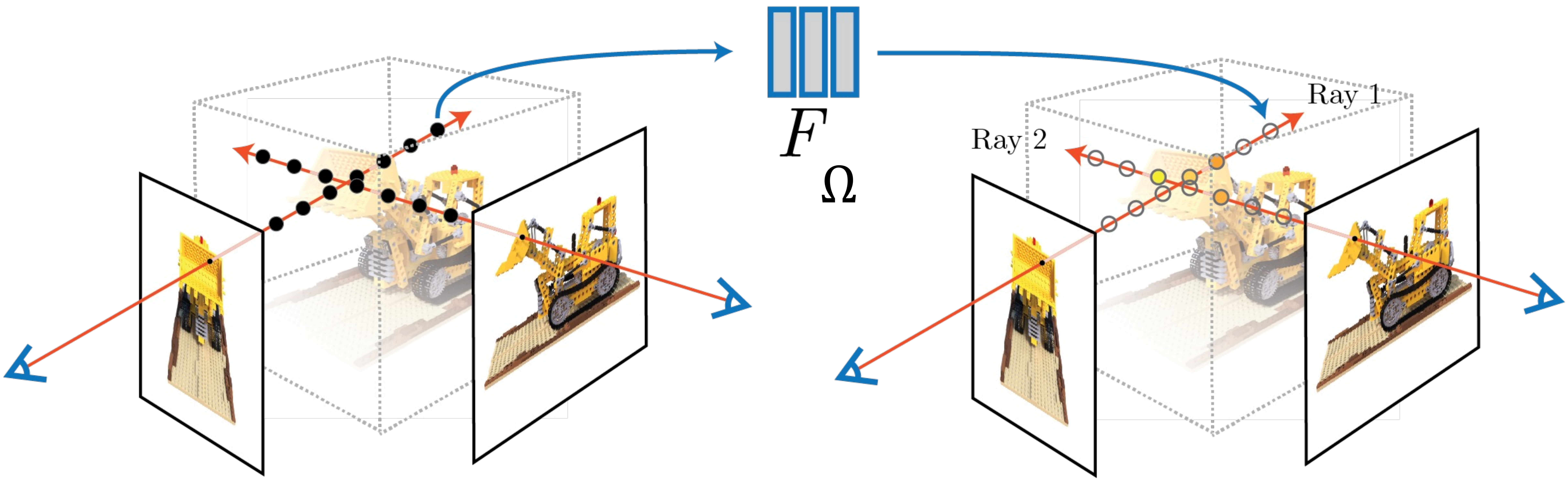
$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$

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

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

Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

Optimize with gradient descent on rendering loss



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

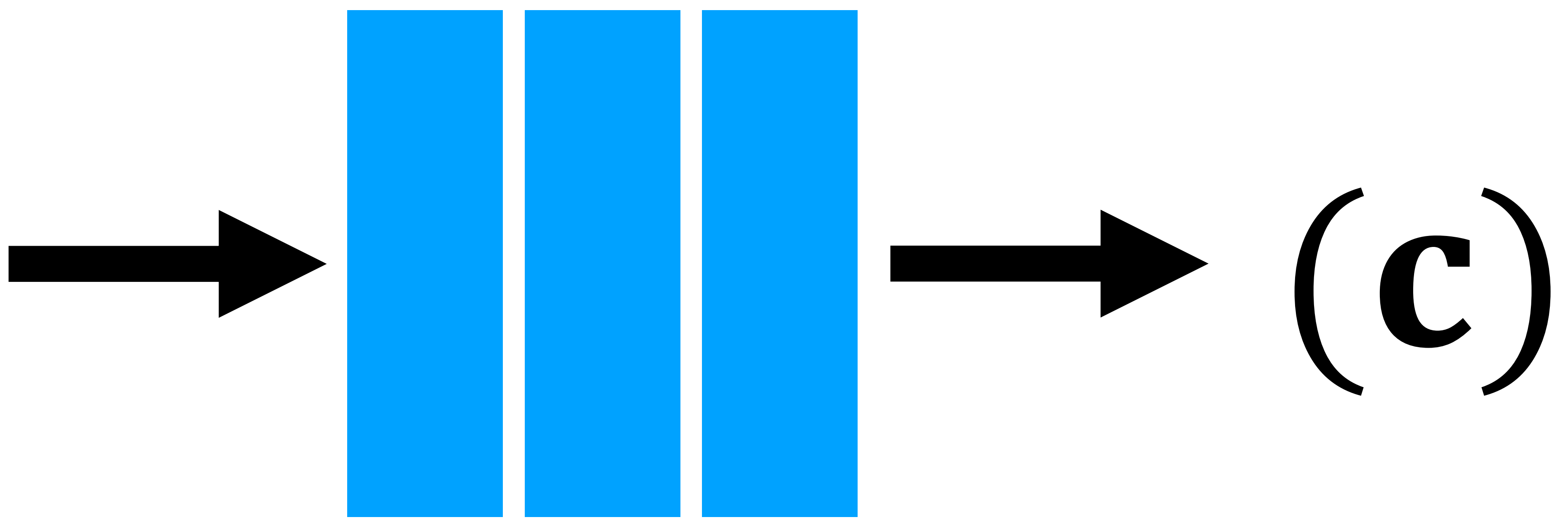
Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

Training network to reproduce all input views of the scene

Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

Positional encoding: high frequency embedding of input coordinates

$$\left(\begin{array}{c} \sin(\mathbf{x}), \cos(\mathbf{x}) \\ \sin(2\mathbf{x}), \cos(2\mathbf{x}) \\ \sin(4\mathbf{x}), \cos(4\mathbf{x}) \\ \vdots \\ \sin(2^N\mathbf{x}), \cos(2^N\mathbf{x}) \end{array}\right)$$



Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

Simple trick enables network to memorize images

Ground truth image

Standard fully-connected net

With "embedding"

Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

Positional encoding also directly improves our scene representation!

NeRF (Naive)

NeRF (with positional encoding)

Slides credit: Ben Mildenhall, Pratul Srinivasan et al.,

Implementation Details

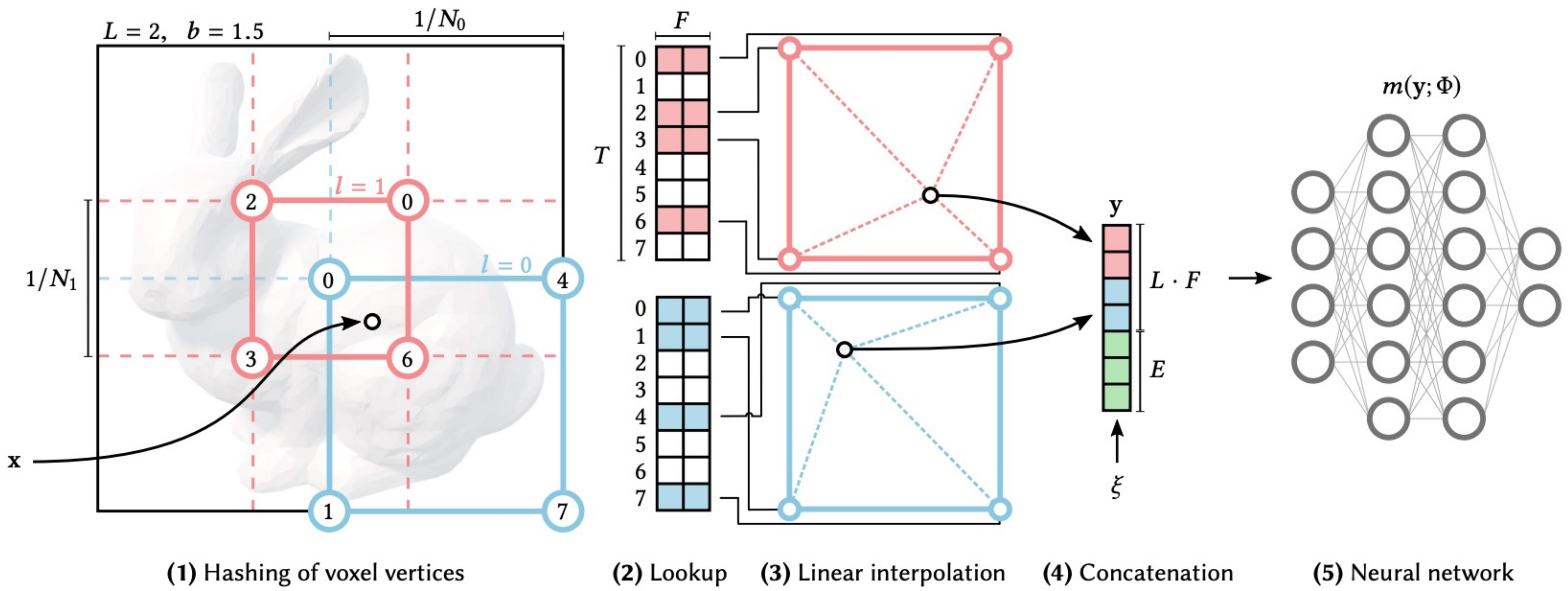
Camera Locations and Poses

- Use Structure from Motion (e.g., [COLMAP]) to initialize camera poses
- Incorrect camera poses lead to bad results
- Joint optimization of camera poses and scene presentation.

Photo credit: https://colmap.github.io/

Implementation Details

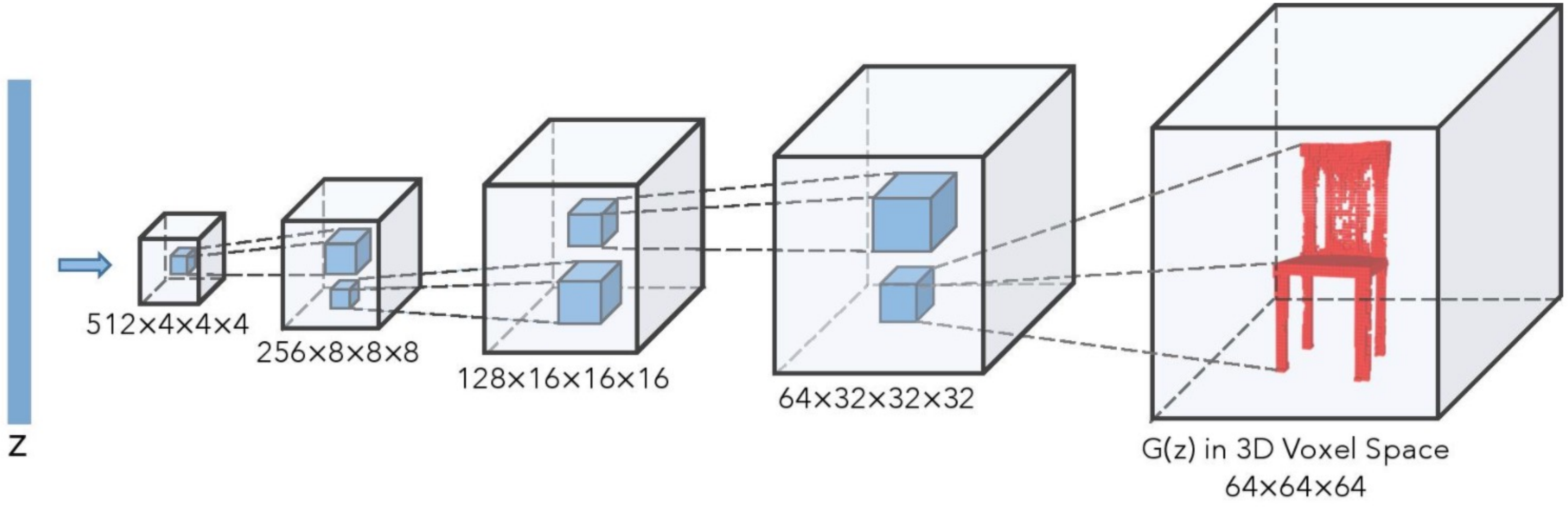
Training and inference speed:
- Original NeRF is quite slow.
- Faster training and inference is an active research topic.
- Optimized CUDA kernel for small MLP network (10x faster)
- Efficient data structure: multi-resolution hashing (10+ faster)



Instant Neural Graphics Primitives [Müller et al.,]

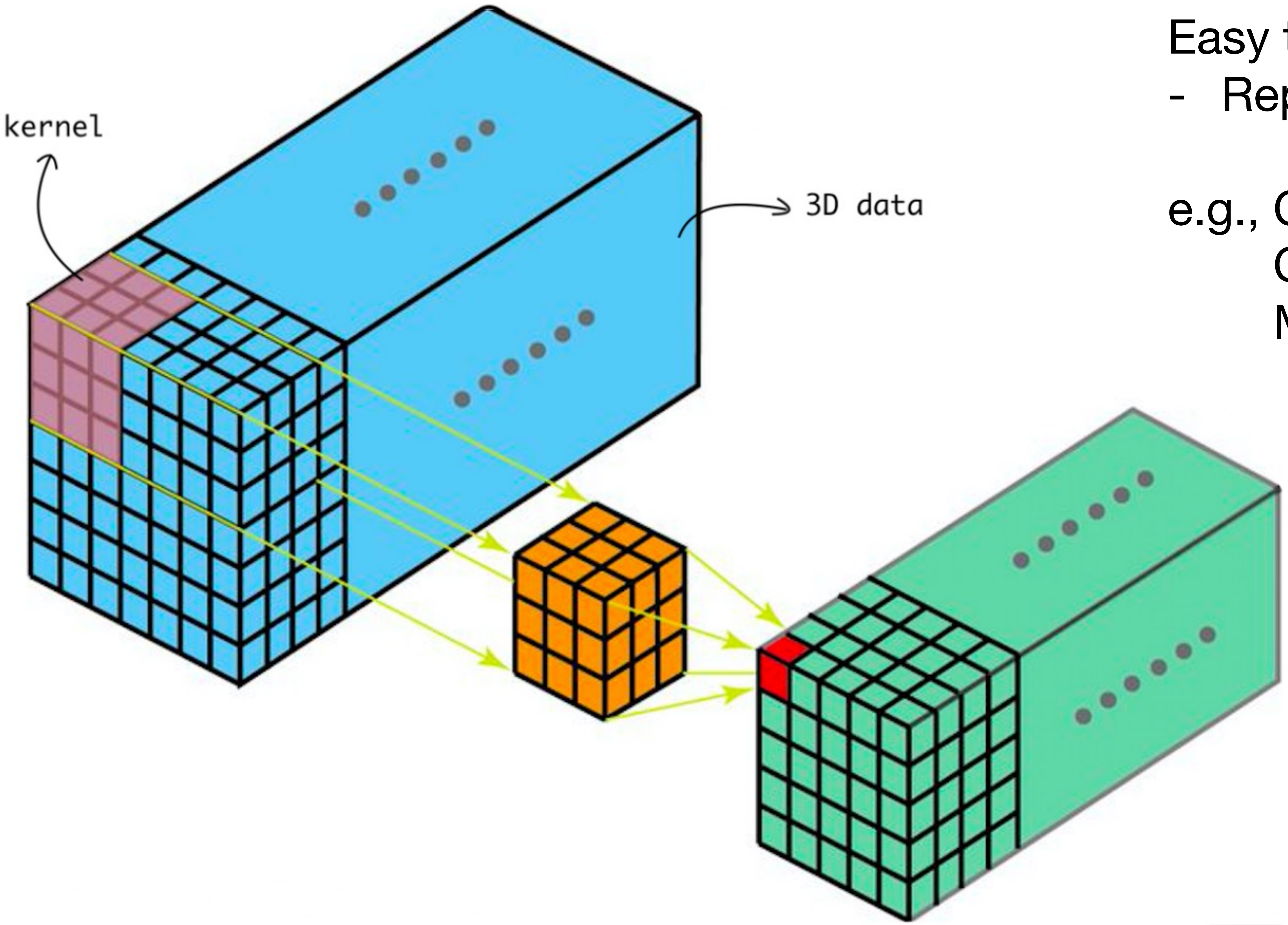
Toward 3D-aware Generative Models

3D Generative Adversarial Networks



Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. [Wu*, Zhang*, et al., NeurIPS 2016]

3D Convolutional Layers



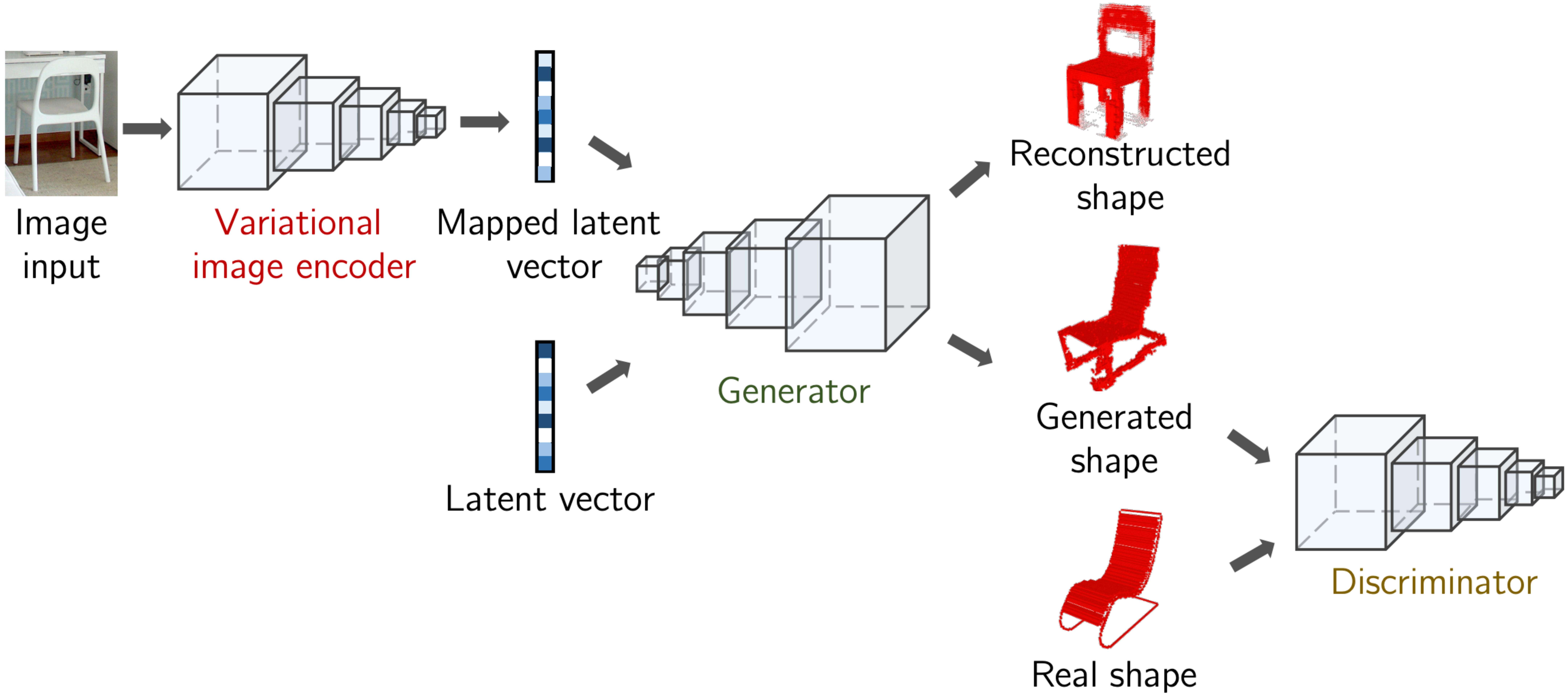
Easy to implement:
- Replace 2D by 3D in your code

e.g., Conv2D -> Conv3D
ConvTranspose2d->ConvTranspose3d
MaxPool2d -> MaxPool3d

```
CLASS torch.nn.Conv3d(in_channels, out_channels, kernel_size, stride=1, padding=0,
      dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None) [SOURCE]
```

Photo credit: Shiva Verma

3D Generative Adversarial Networks



Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. [Wu*, Zhang*, et al., NeurIPS 2016]

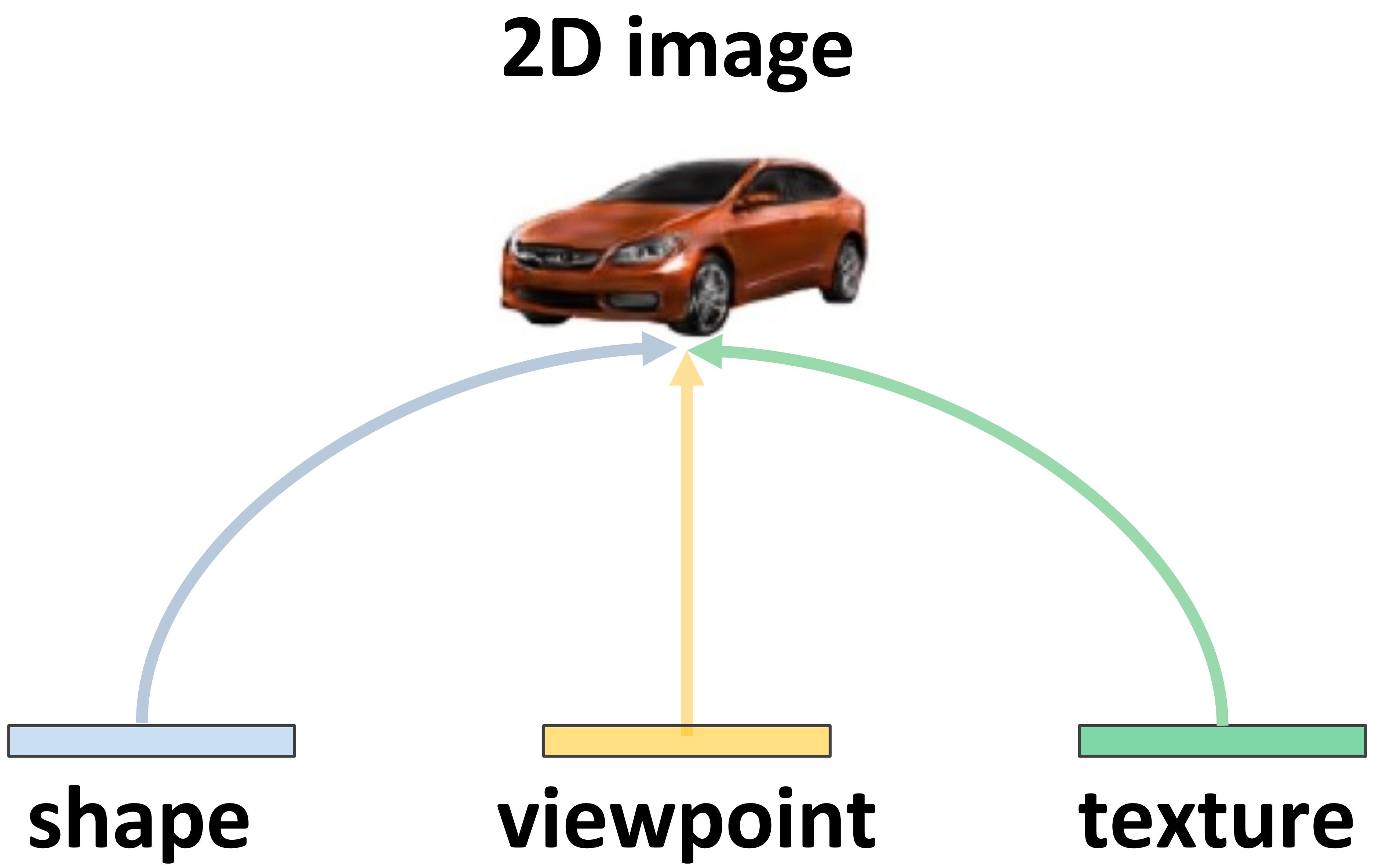
3D Generative Adversarial Networks

Input image | Reconstructed 3D shape | Input image | Reconstructed 3D shape

Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. [Wu*, Zhang*, et al., NeurIPS 2016]

How to add Color and Texture?

Learning 3D Disentanglement



Learning 3D Disentanglement

Visual Object Networks: Image Generation with Disentangled 3D Representation [Zhu et al., NeurIPS 2018]

Learning 3D Disentanglement

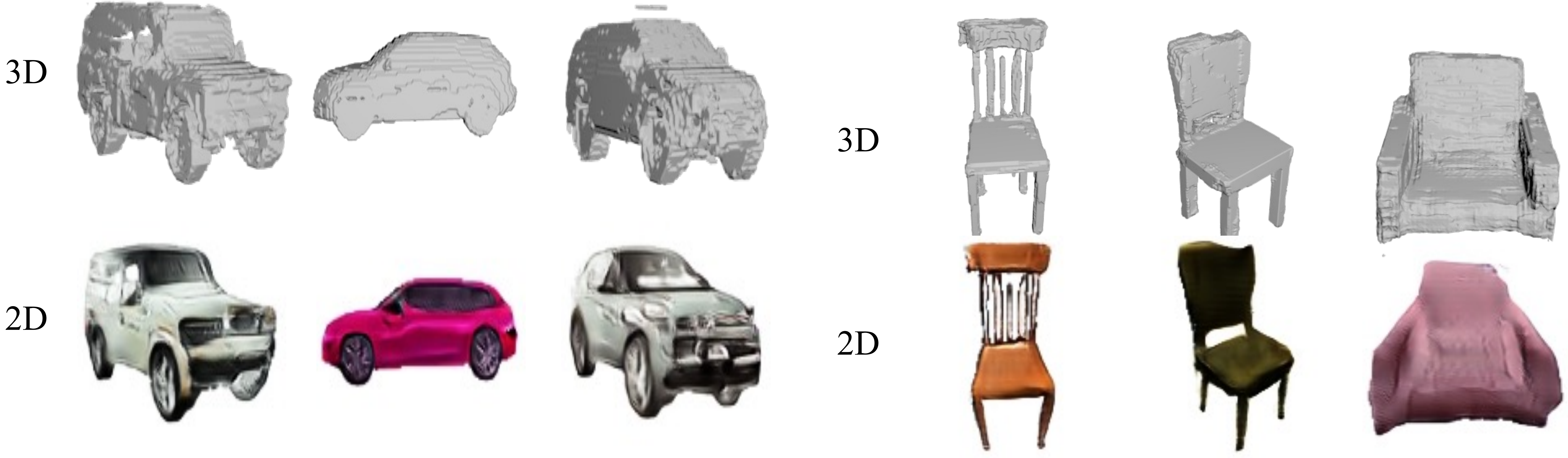
samples from 2D GANs

our 3D, 2.5D, and 2D output

3D disentanglement

Visual Object Networks: Image Generation with Disentangled 3D Representation [Zhu et al., NeurIPS 2018]

Learning 3D Disentanglement



Visual Object Networks: Image Generation with Disentangled 3D Representation [Zhu et al., NeurIPS 2018]

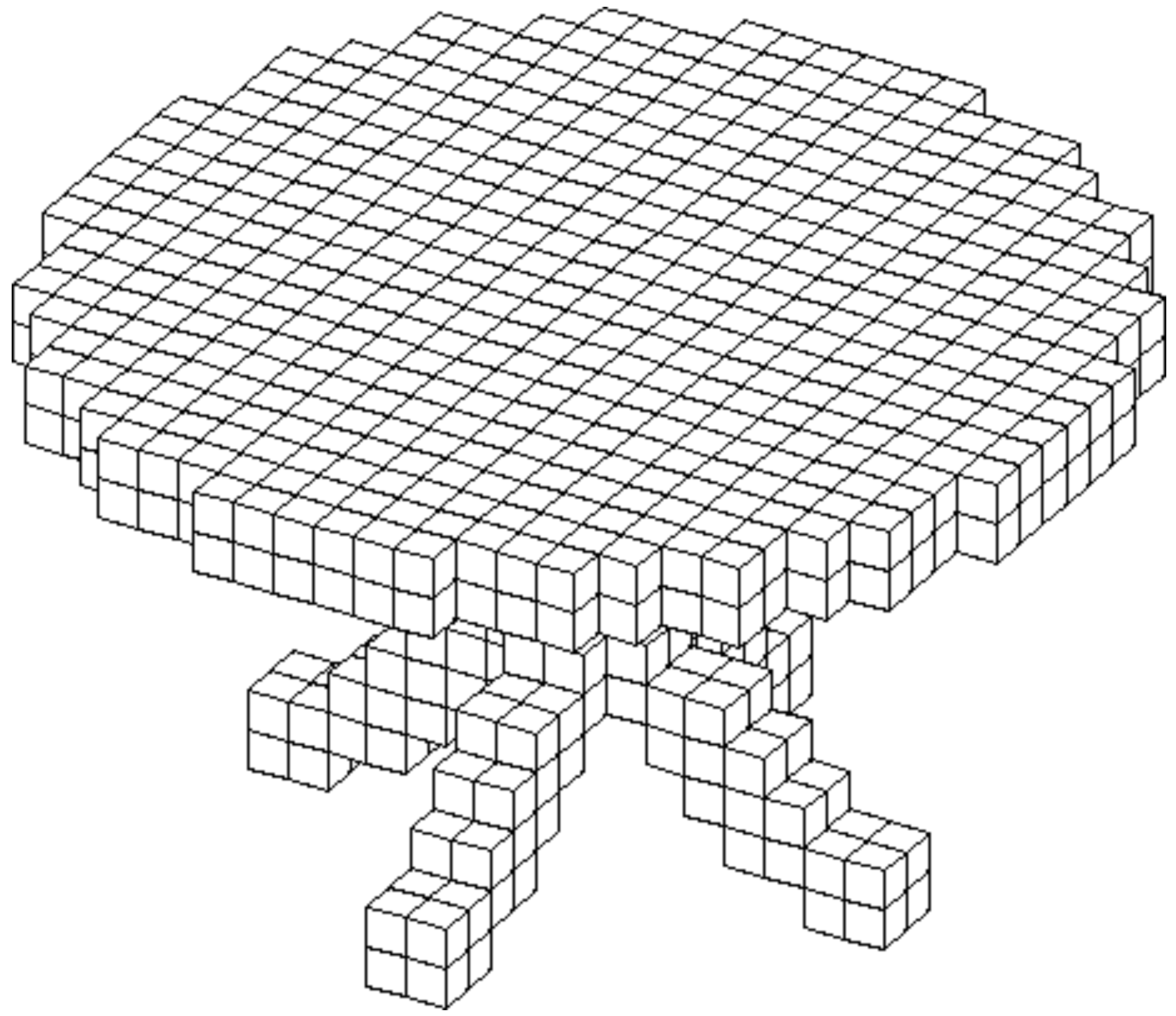
Learning 3D Disentanglement

Visual Object Networks: Image Generation with Disentangled 3D Representation [Zhu et al., NeurIPS 2018]

Limitations:

1. Voxel representation is expensive.
2. Requires ground truth 3D data.

Volumetric 3D



Each grid cell stores information (e.g., occupancy, color)

Very general but memory-intensive

256x256x256 -> 1024x1024x1024

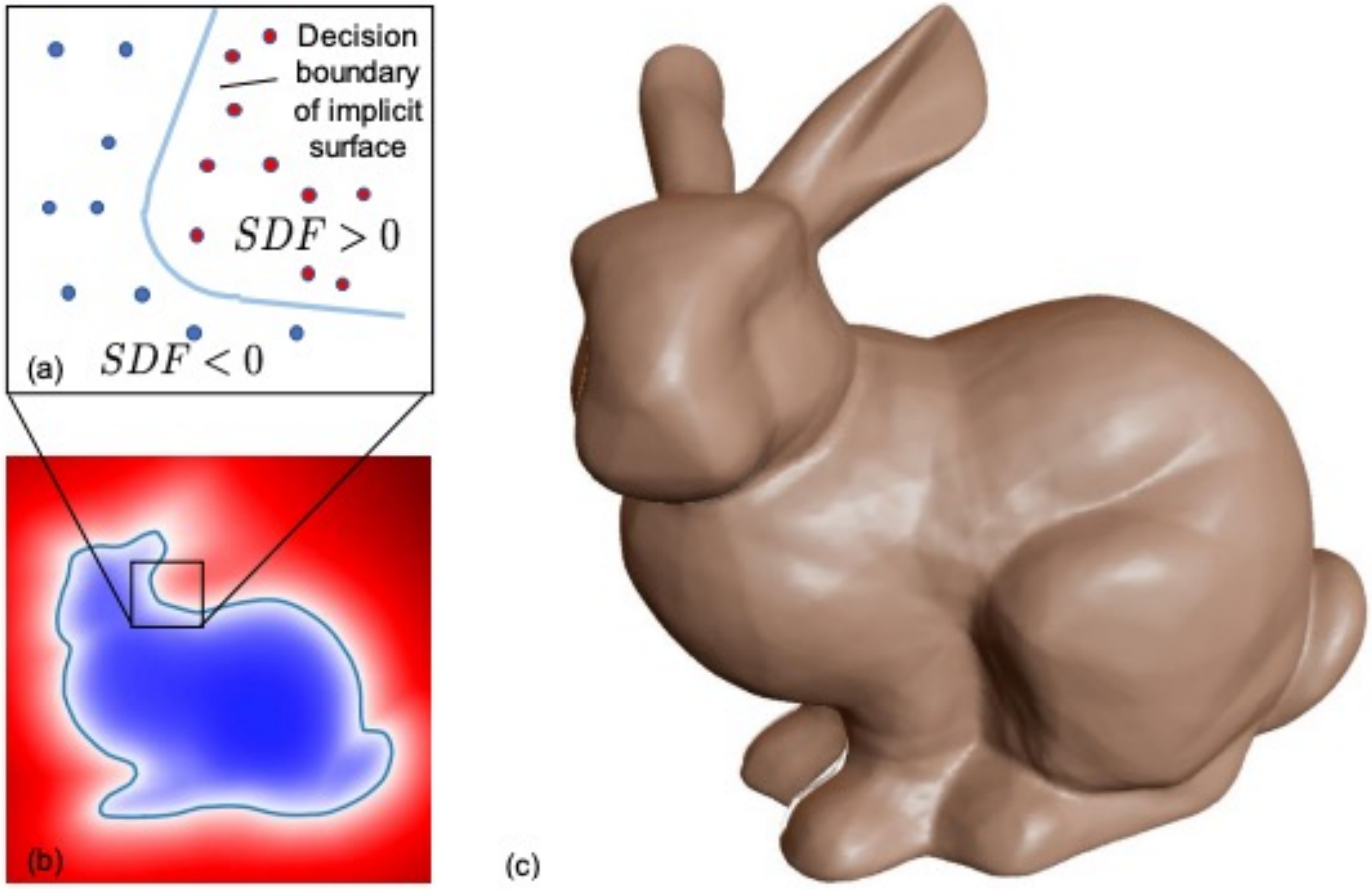
Cannot even fit a single training data to GPU

Slide credit: Shubham Tulsiani

Improvements:

1. Using implicit representation (network-based)

Signed Distance Function (SDF)



Explicit function:

$y = 2x.\,(\mathrm{y} = \mathrm{f}(\mathrm{x}))$

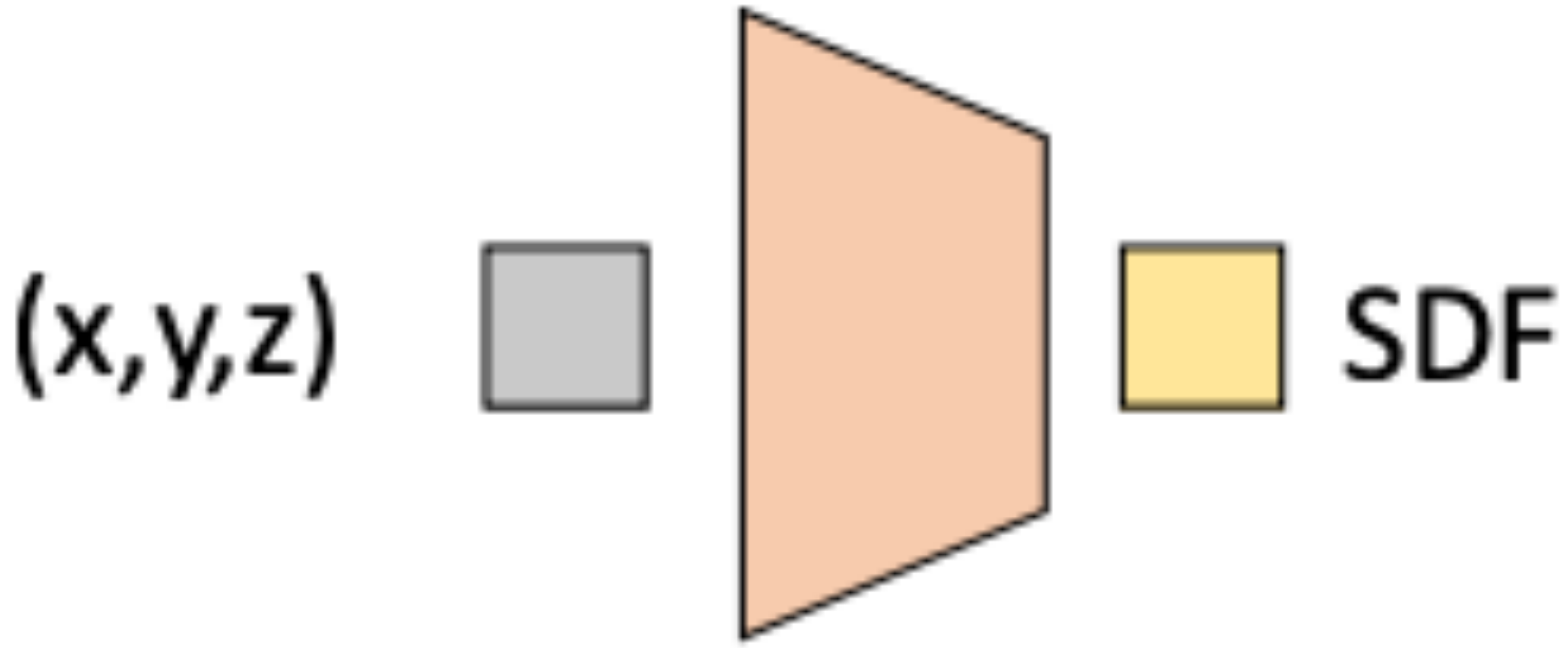
Implicit function:

$2y - 4x = 0, F(x, y) = 0$

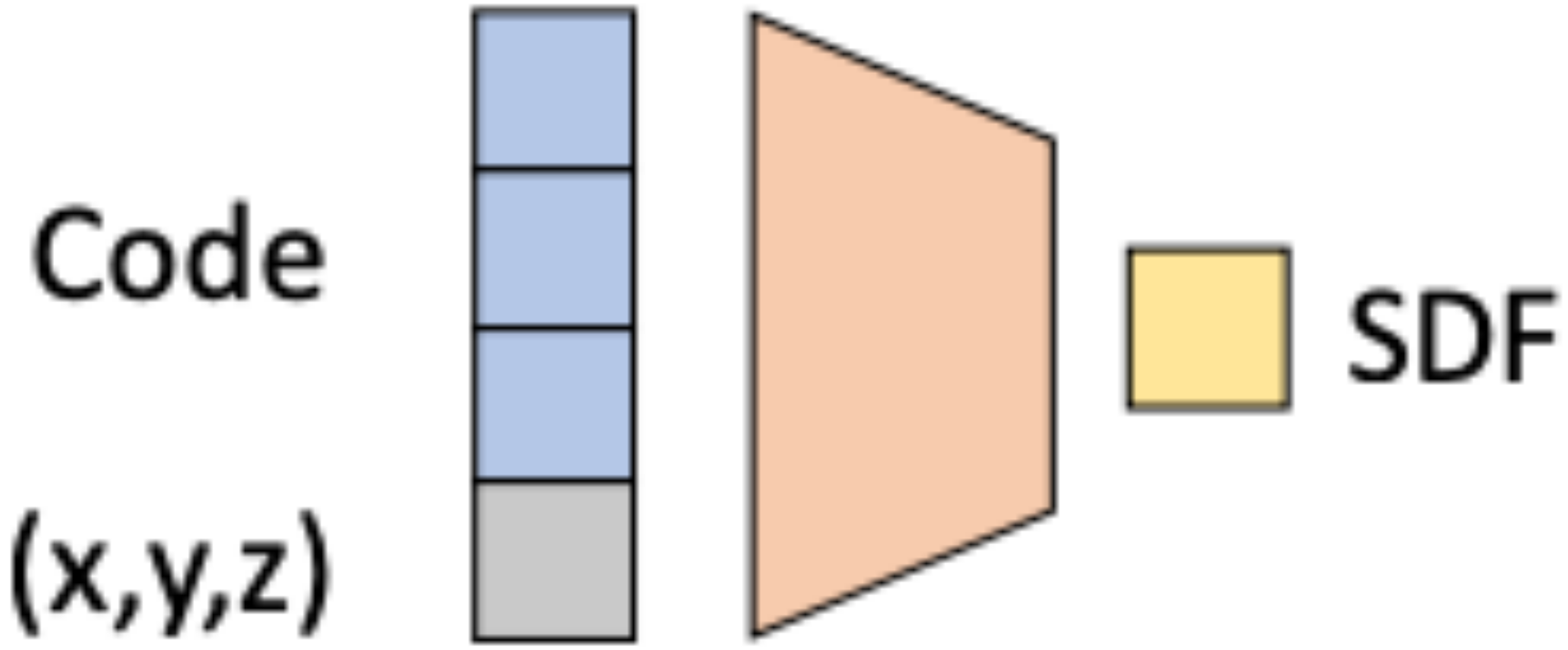
A set of zeros of a function of two variables.

DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. [Park et al., CVPR 2019]

Deep SDF



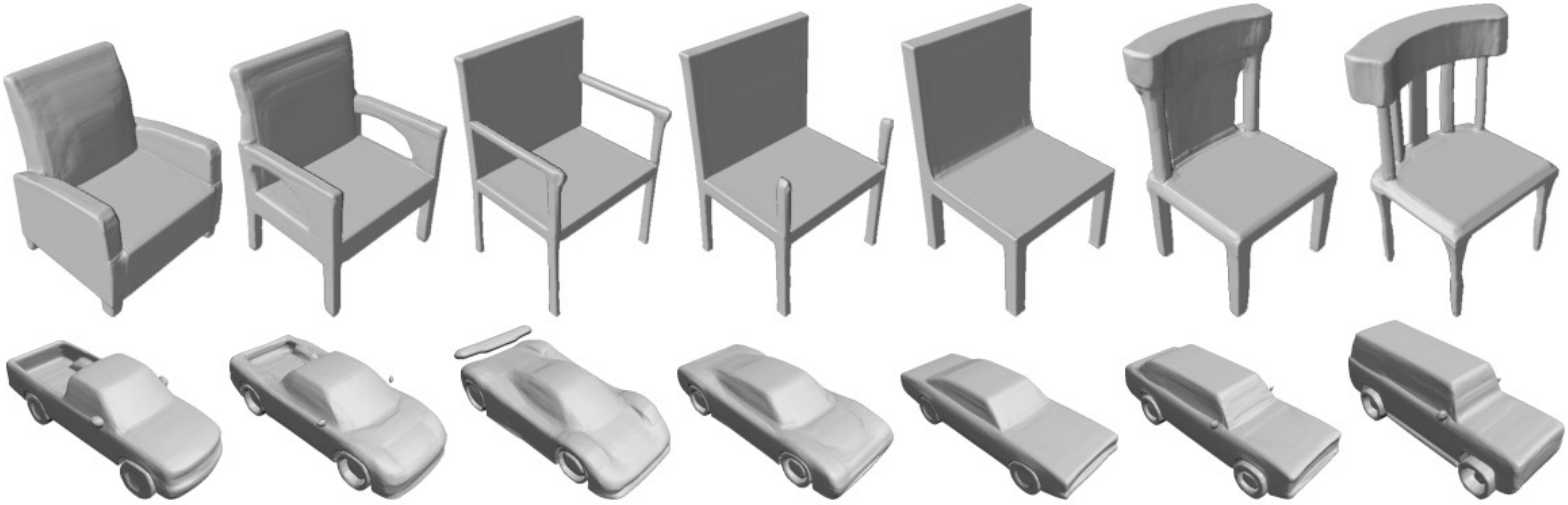
(a) Single Shape DeepSDF



(b) Coded Shape DeepSDF

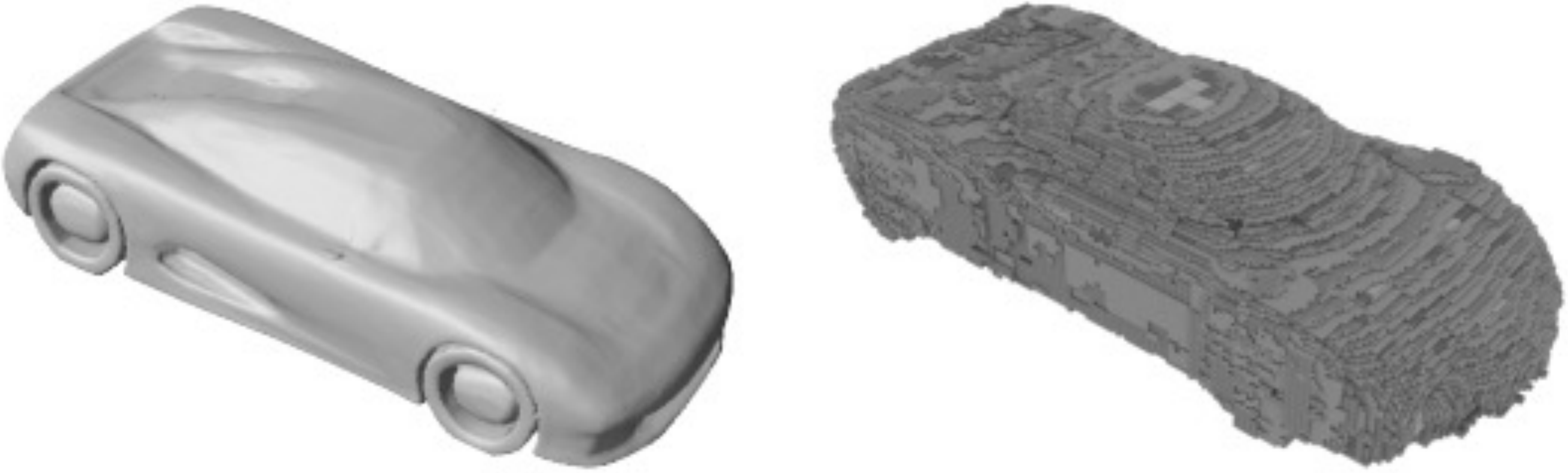
DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. [Park et al., CVPR 2019]

Deep SDF



DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. [Park et al., CVPR 2019]

Deep SDF



DeepSDF preserve details and render visually pleasing results compared to voxel-based methods.

Improvements:

1. Using implicit representation (network-based)
2. Learning from image collections

HoloGAN

<u>Representation</u>: 3D feature representation

<u>Training</u>: Adversarial loss + latent code reconstruction

<u>Modulation</u>: AdaIN

HoloGAN: Unsupervised Learning of 3D Representations From Natural Images. [Nguyen-Phuoc et al., ICCV 2019]

HoloGAN

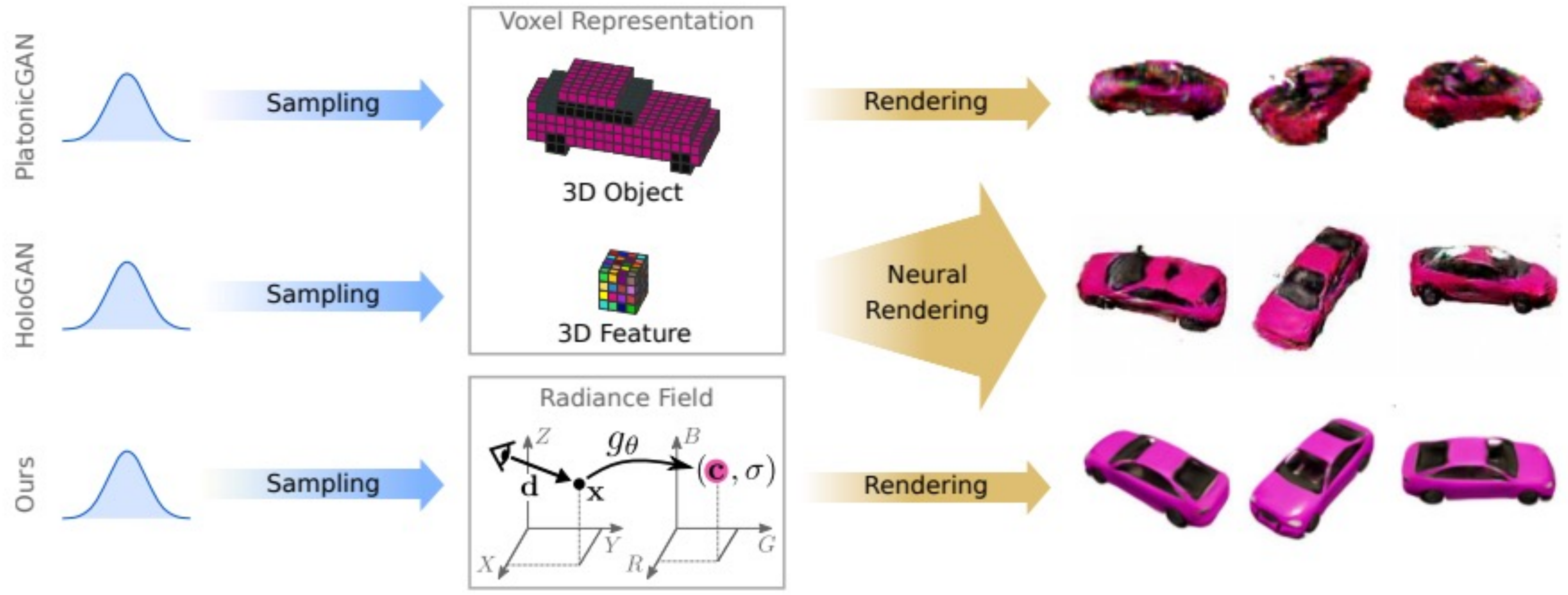
Limitations:
- Do not synthesize geometric outputs (e.g., voxels, SDF).
- No explicit viewpoint consistency. (same issue with Visual Object Networks)

HoloGAN: Unsupervised Learning of 3D Representations From Natural Images. [Nguyen-Phuoc et al., ICCV 2019]

NeRF + GANs

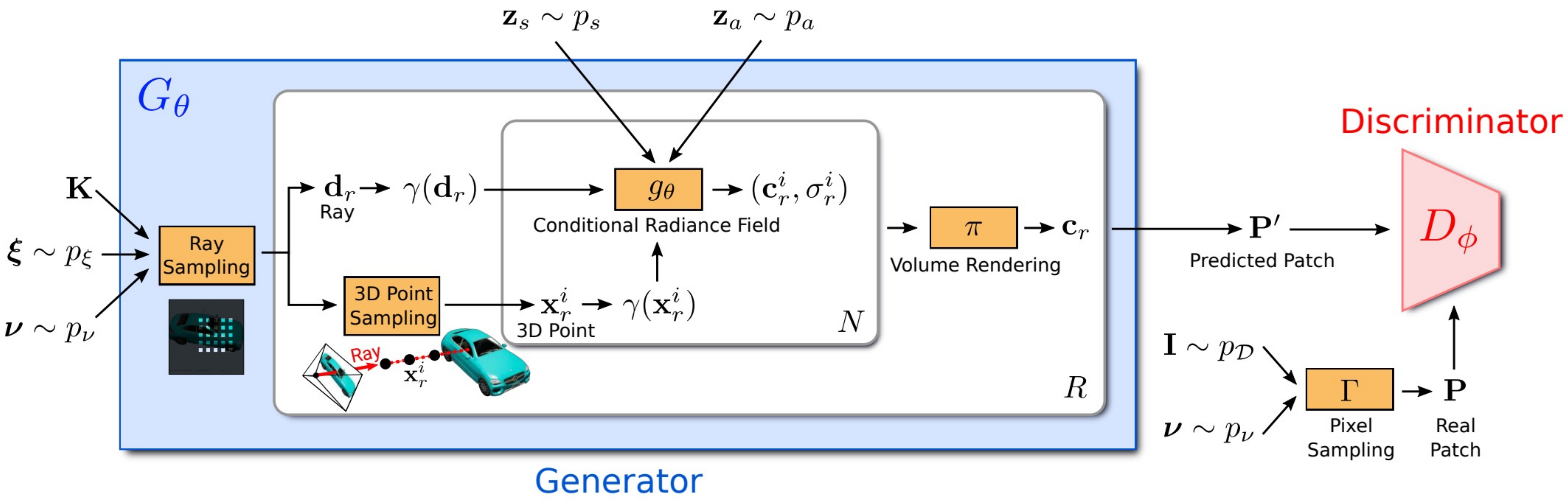
(Neural rendering + Generative Models)

GRAF: Generative Radiance Fields



GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis. [Schwarz*, Liao*, et al., ICCV 2019]

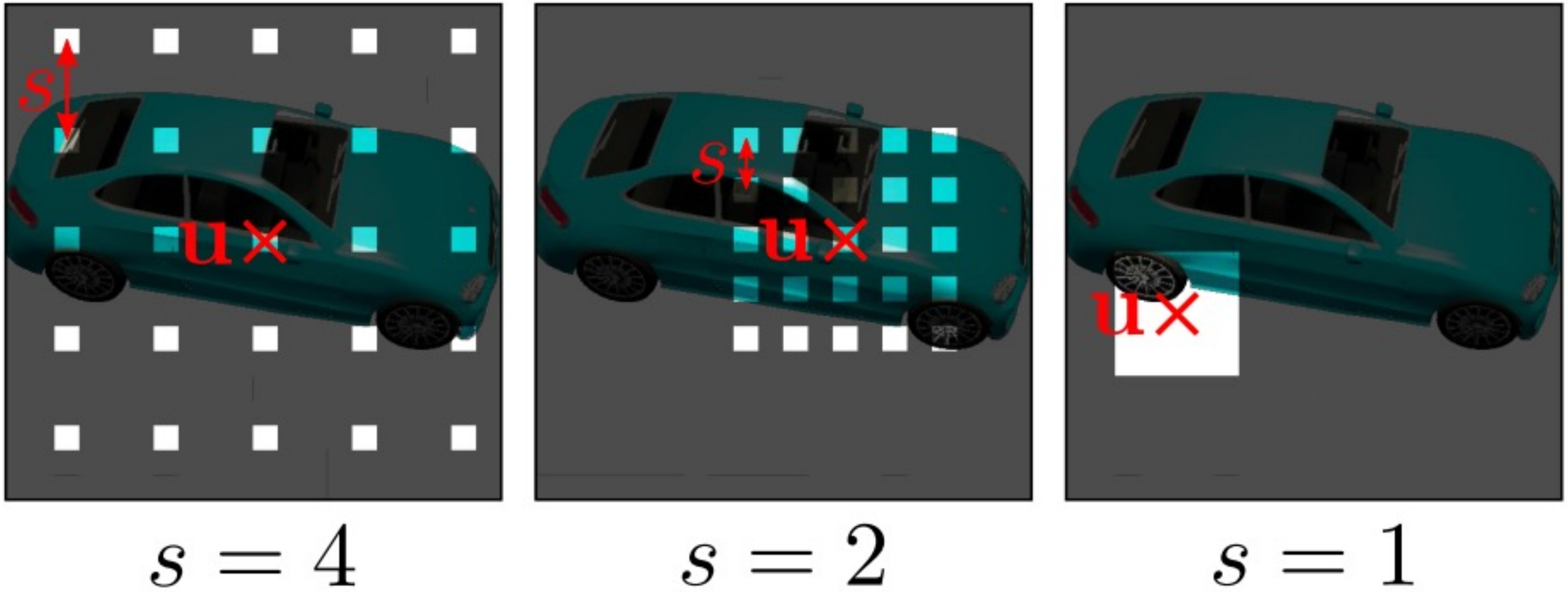
GRAF: Generative Radiance Fields



- **NeRF Generator** is conditioned on both shape and appearance code.
- **Patch-based Discriminator** (full-image discriminator is too slow)

GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis. [Schwarz*, Liao*, et al., ICCV 2019]

GRAF: Generative Radiance Fields



Multi-scale ray sampling

GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis. [Schwarz*, Liao*, et al., ICCV 2019]

Training a 3D-Aware GAN

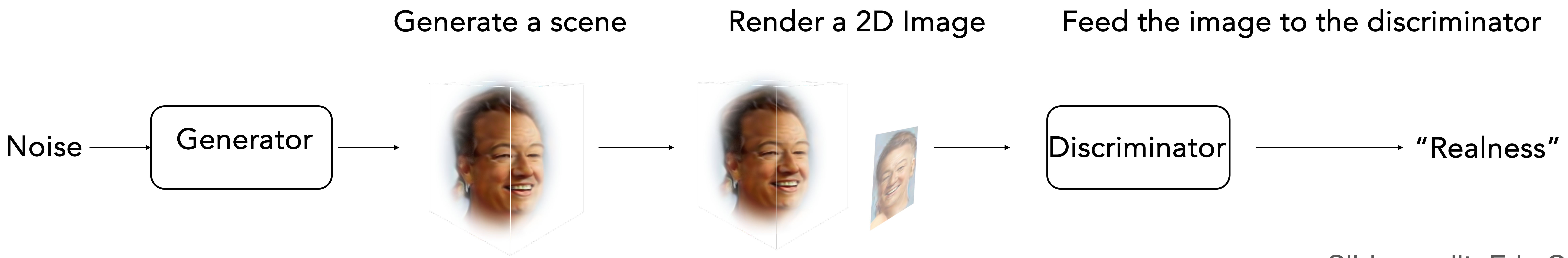
3D-Aware GAN Training Steps

1. Generate a representation of a *scene*

2. Render the scene from a random camera pose

3. Feed the image to a 2D discriminator

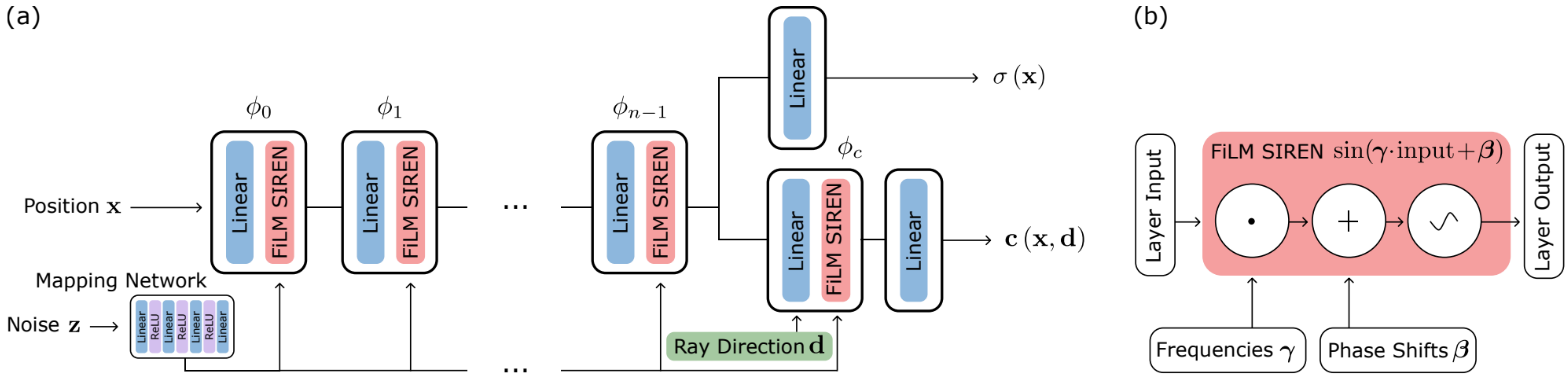
4. Backpropagate through the discriminator and differentiable rendering



Slide credit: Eric Chan

pi-GAN: Periodic Implicit Generative Adversarial Networks for 3D-Aware Image Synthesis [Chan et al., 2021]

π-GAN



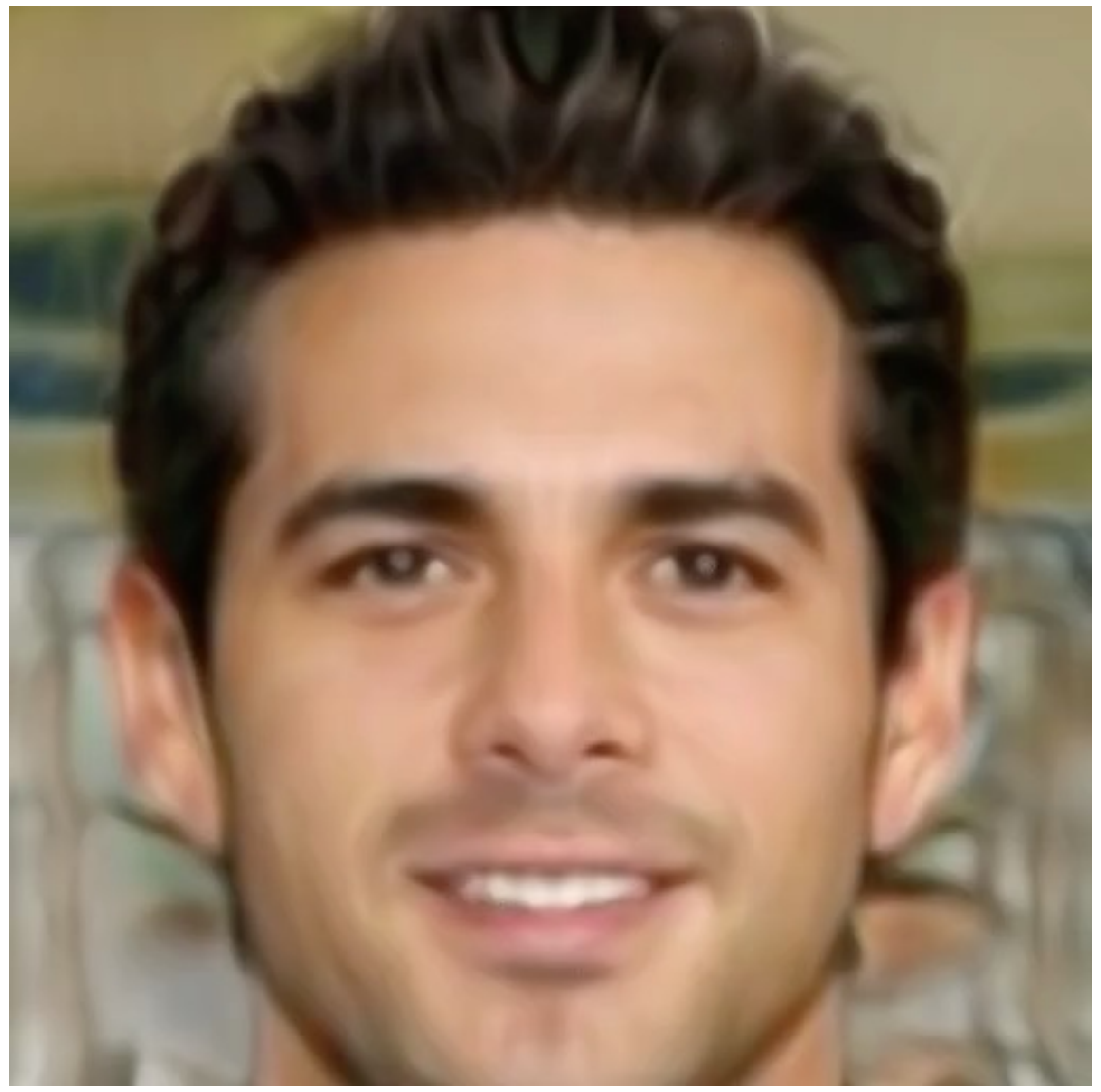
Mapping network + AdaIN (FILM) + learnable positional encoding

$$\phi_i(\mathbf{x}_i) = \sin(\boldsymbol{\gamma}_i \cdot (\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i) + \boldsymbol{\beta}_i)$$

pi-GAN: Periodic Implicit Generative Adversarial Networks for 3D-Aware Image Synthesis [Chan et al., 2021]

π-GAN

Focal Length



Camera Position

Latent Interpolation

Slide credit: Eric Chan

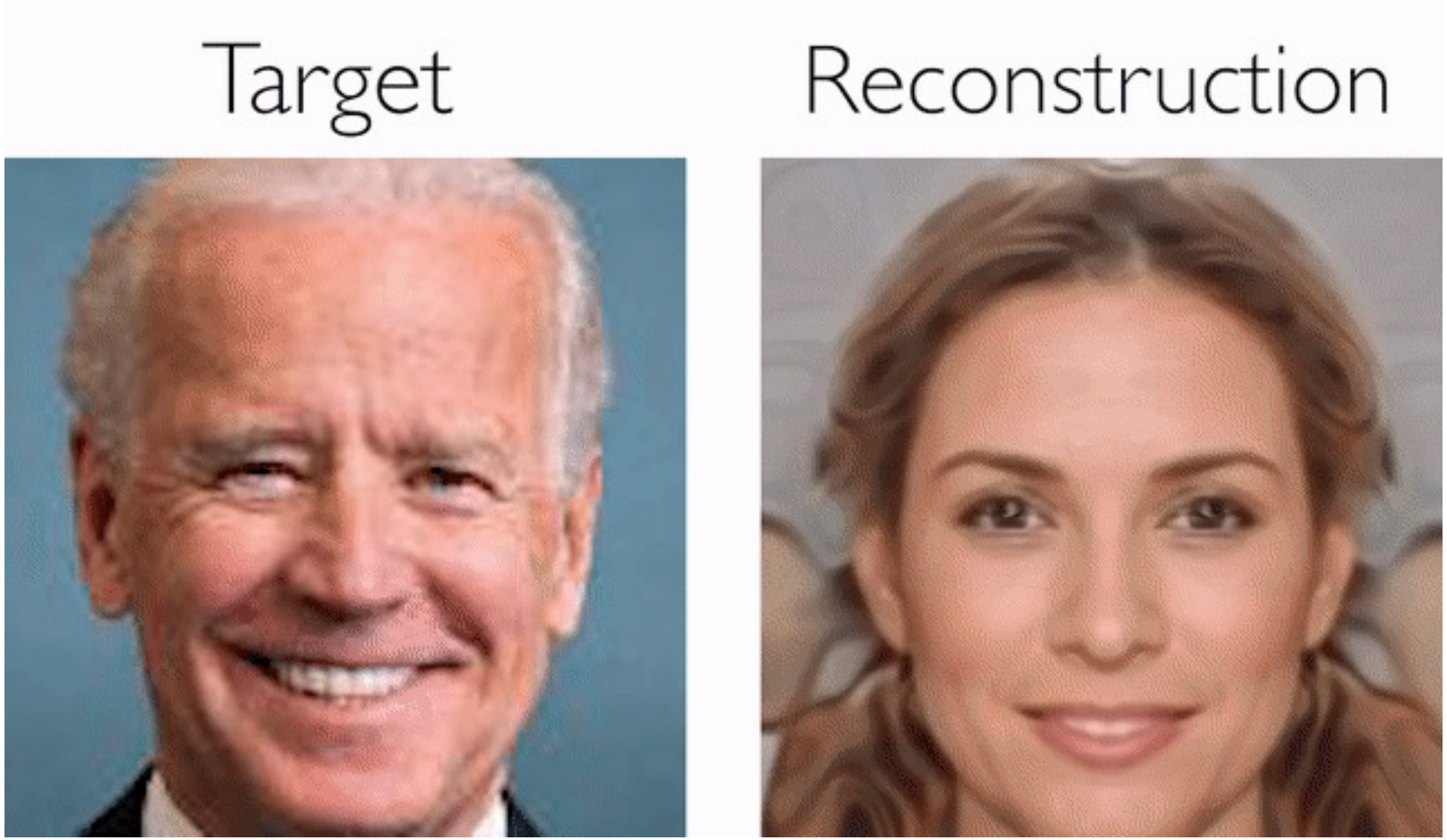
pi-GAN: Periodic Implicit Generative Adversarial Networks for 3D-Aware Image Synthesis [Chan et al., 2021]

π-GAN

Slide credit: Eric Chan

pi-GAN: Periodic Implicit Generative Adversarial Networks for 3D-Aware Image Synthesis [Chan et al., 2021]

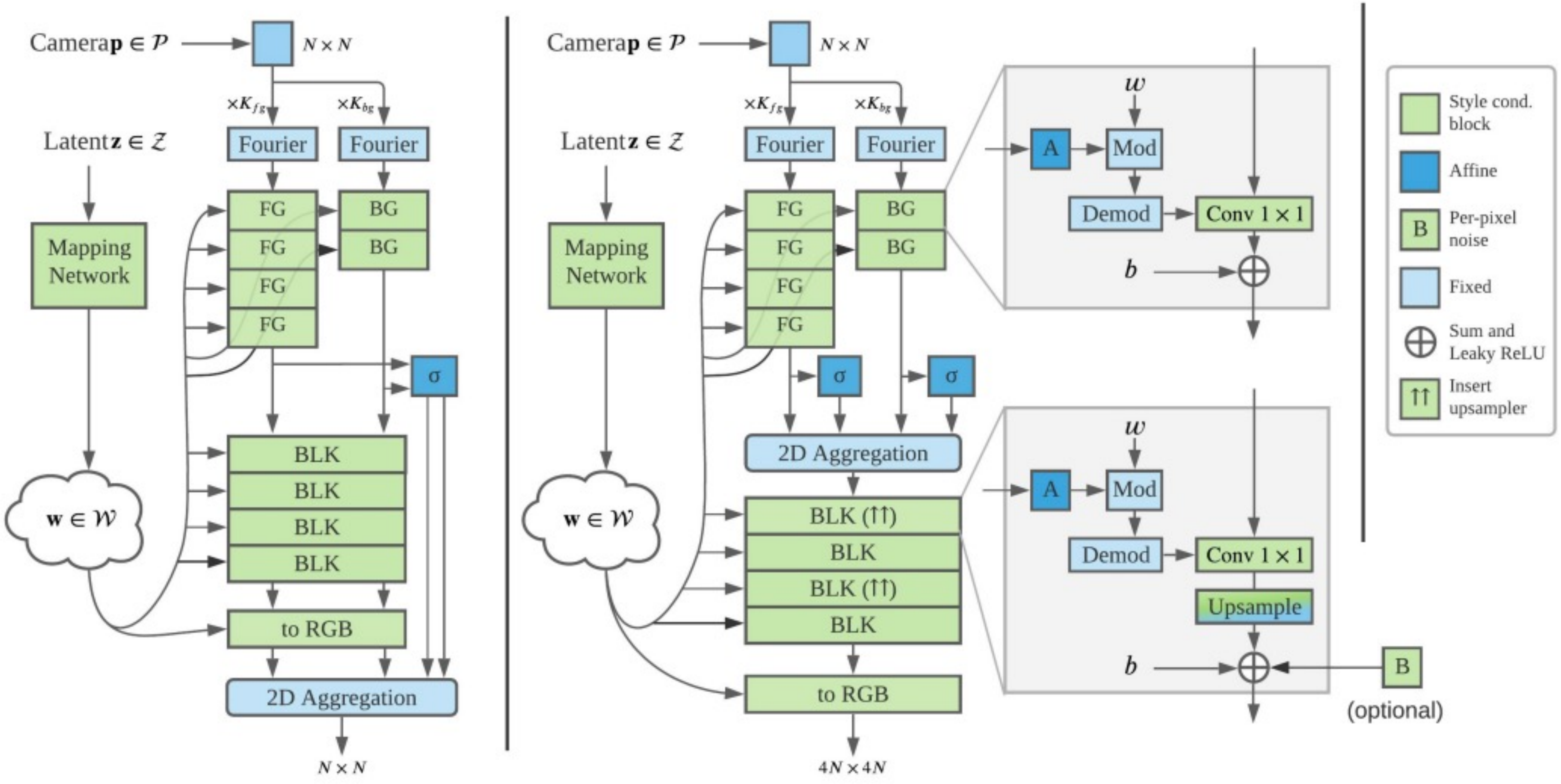
π-GAN



Slide credit: Eric Chan

pi-GAN: Periodic Implicit Generative Adversarial Networks for 3D-Aware Image Synthesis [Chan et al., 2021]

Advanced Architectures: StyleNeRF

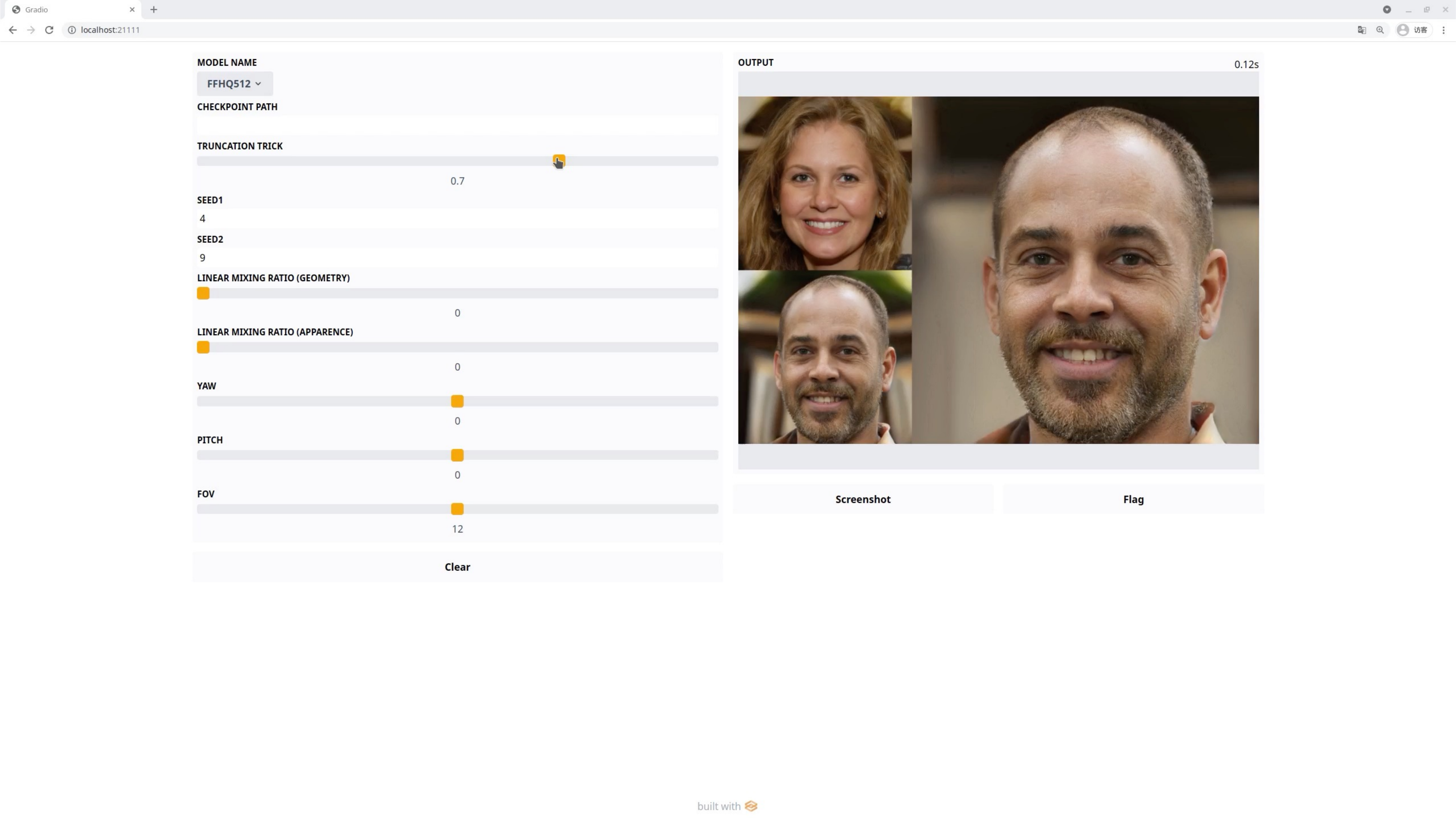


Baseline architecture

Proposed architecture
rendering features via volumetric rendering + GANs-based upsampler

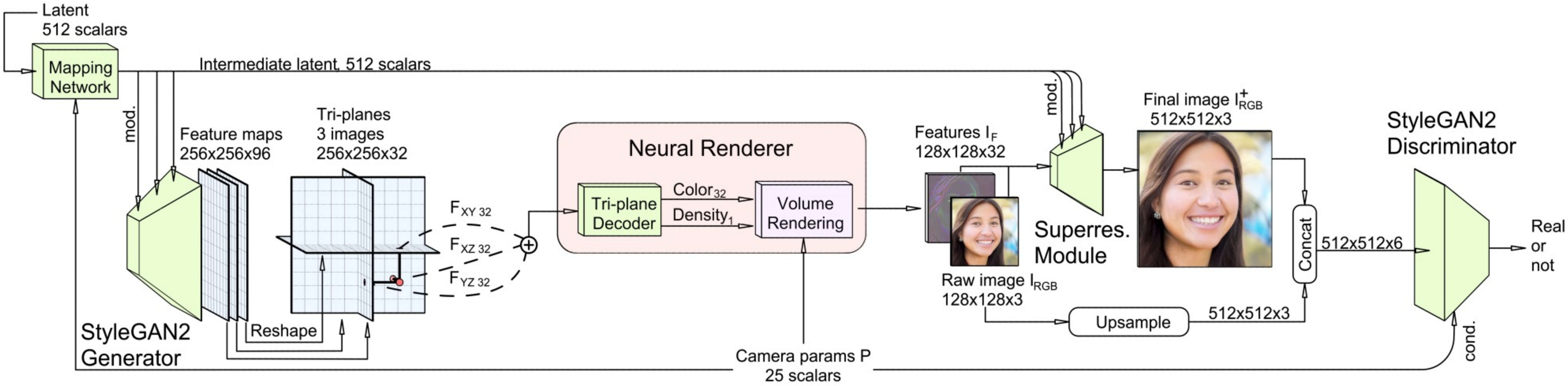
Also see recent work: e.g., StyleNeRF [Gu et al.], EG3D [Chan et al.], StyleSDF [Or-El et al.], ShadeGAN [Pan et al.], …

StyleNeRF: A Style-based 3D-Aware Generator for High-resolution Image Synthesis [Gu et al., 2021]



StyleNeRF: A Style-based 3D-Aware Generator for High-resolution Image Synthesis [Gu et al., 2021]

Advanced Architectures: EG3D (StyleNeRF+Triplane)



Tri-plane representation for speed-up

F(x, y, x) -> F(x, y) + F(x, z) + F(y, z)

Rendering features via volumetric rendering

features are useful for upsampling

Generate final output via image encoder

If the model is too slow, use GAN-based upsampler

Also see recent work: e.g., StyleNeRF [Gu et al.], EG3D [Chan et al.], StyleSDF [Or-El et al.], ShadeGAN [Pan et al.], …

EG3D: Efficient Geometry-aware 3D Generative Adversarial Networks [Chan et al., 2021]

EG3D: Efficient Geometry-aware 3D Generative Adversarial Networks [Chan et al., 2021]

Text-based Editing with Generative NeRFs

Text-based Editing

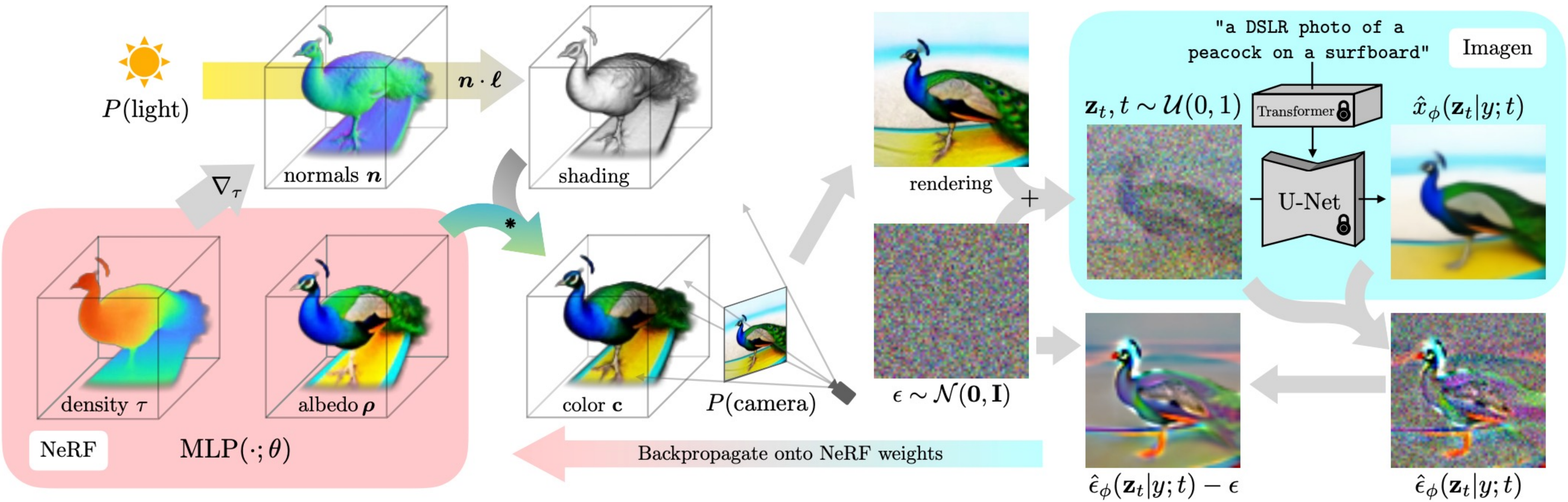
a DSLR photo of a squirrel

wearing a purple hoodie

reading a book

DreamFusion: Text-to-3D using 2D Diffusion [Poole et al., 2022]

Text-based Editing



FOR loop

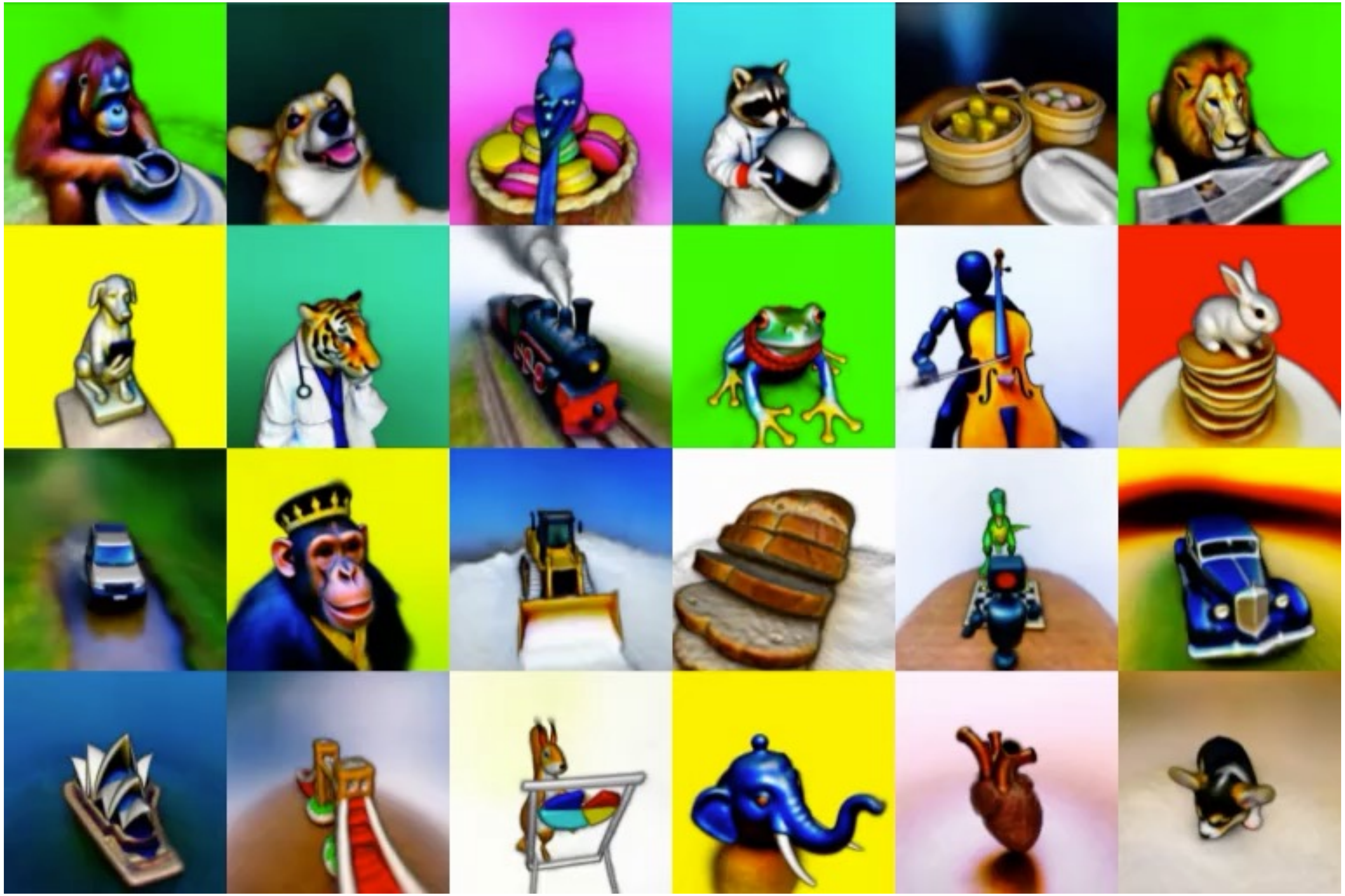
Step 1. Render a view using existing NeRF

Step 2. Add noise and denoise using a pre-trained Stable Diffusion model

Step 3. Update NeRF parameters with the gradient (difference between added and predicted noises)

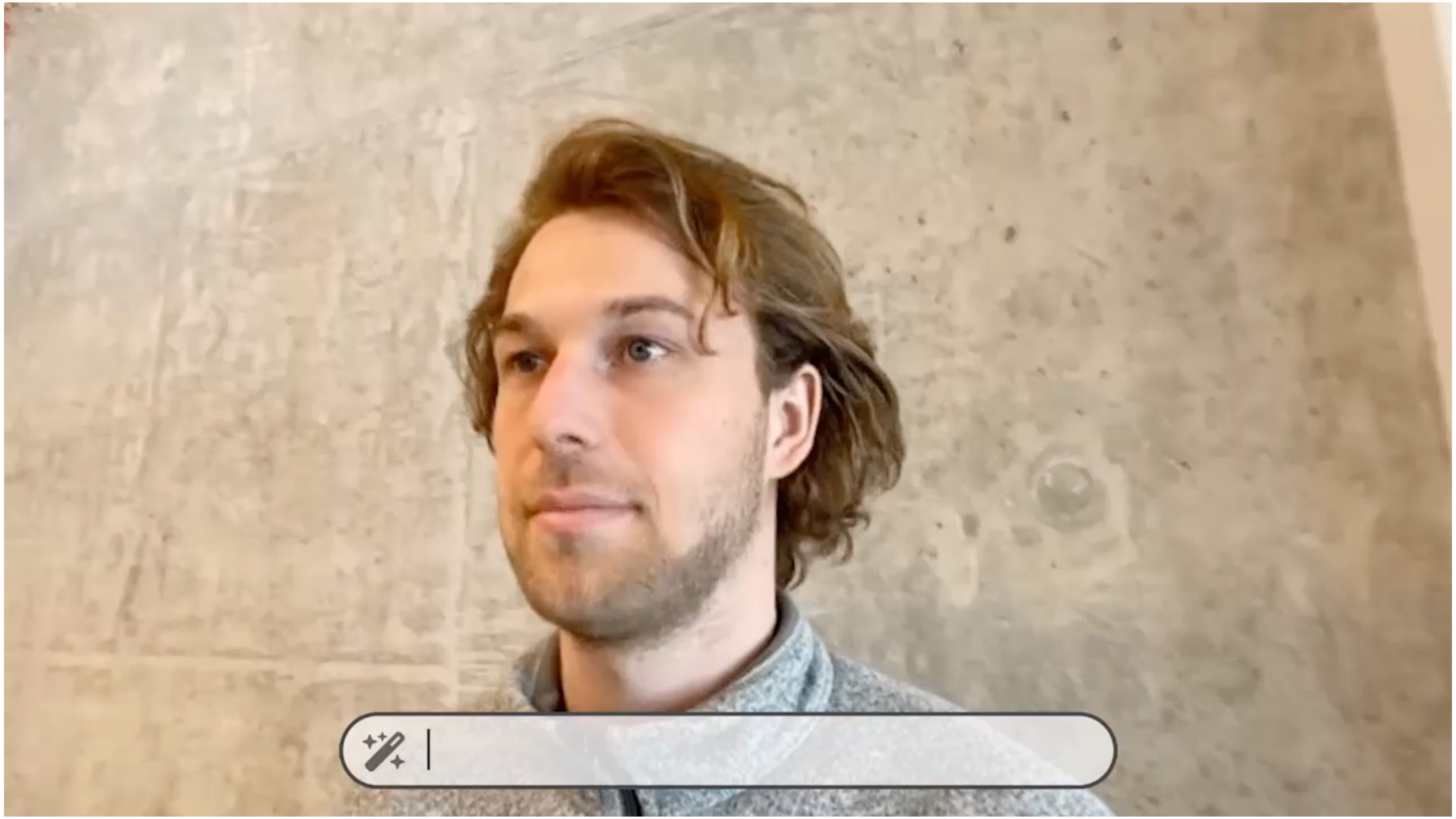
DreamFusion: Text-to-3D using 2D Diffusion [Poole et al., 2022]

Text-based Editing



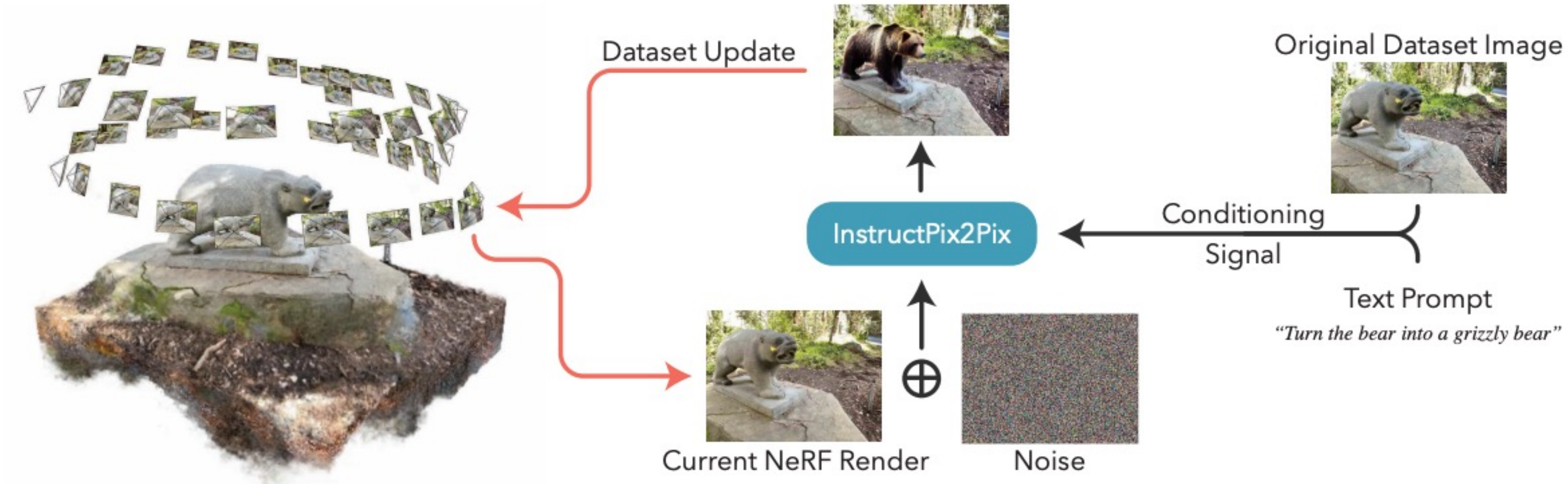
DreamFusion: Text-to-3D using 2D Diffusion [Poole et al., 2022]

Instruct NeRF2NeRF



Instruct-NeRF2NeRF: Editing 3D Scenes with Instructions [Haque et al., 2023]

Instruct NeRF2NeRF



FOR loop

Step 1. Render a view using existing NeRF

Step 2. Use InstructPix2Pix to produce output images

Step 3. Update NeRF parameters with the generated result from Step 2

InstructPix2pix: image-conditional diffusion model (https://www.timothybrooks.com/instruct-pix2pix/)

Instruct-NeRF2NeRF: Editing 3D Scenes with Instructions [Haque et al., 2023]

Thank You!

https://learning-image-synthesis.github.io/