

Computer Vision

Lecture 9 – Coordinate-Based Networks

Kumar Bipin

BE, MS, PhD (MMMTU, IISc, IIIT-Hyderabad)

Robotics, Computer Vision, Deep Learning, Machine Learning, System Software



Agenda

9.1 Implicit Neural Representations

9.2 Differentiable Volumetric Rendering

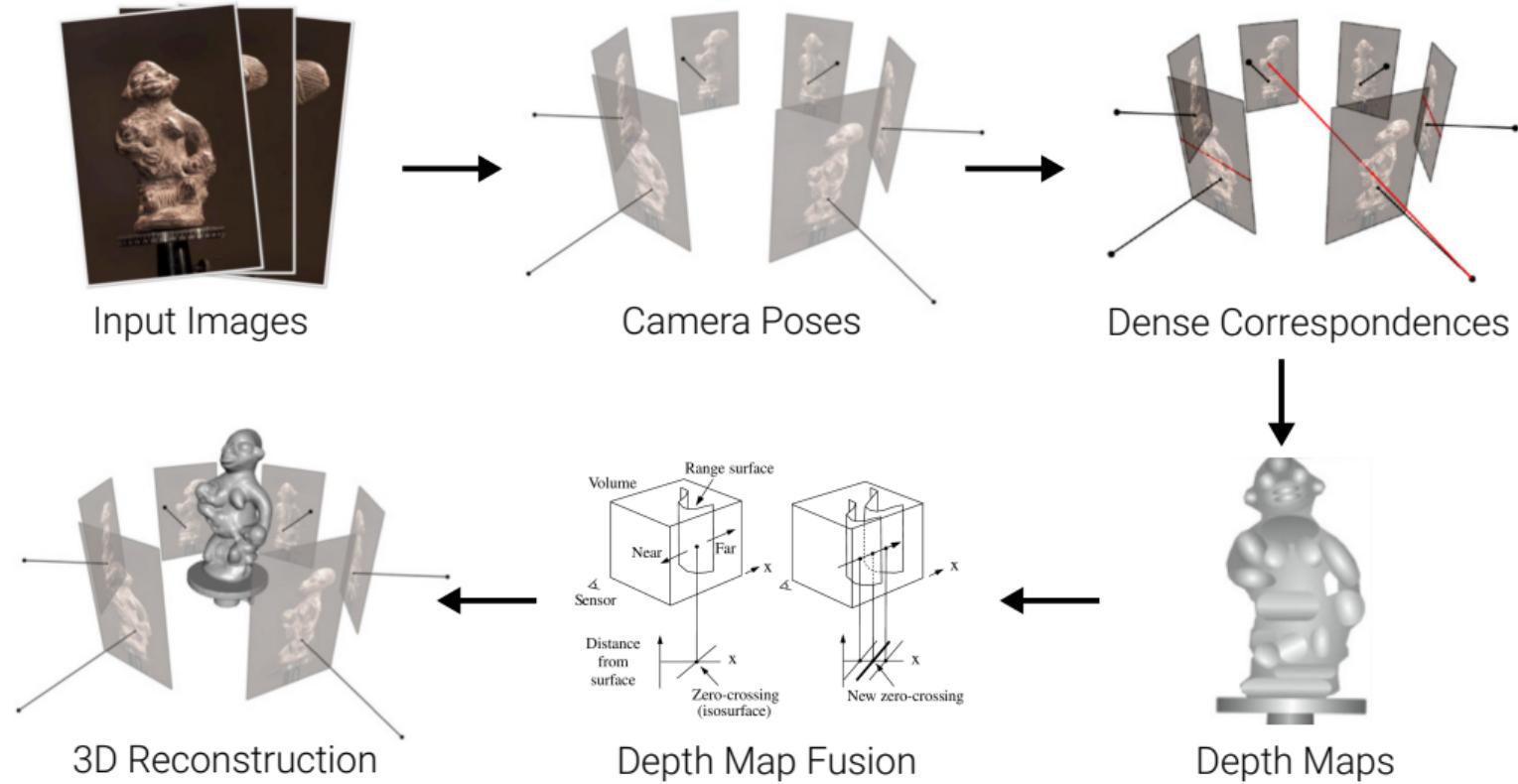
9.3 Neural Radiance Fields

9.4 Generative Radiance Fields

9.1

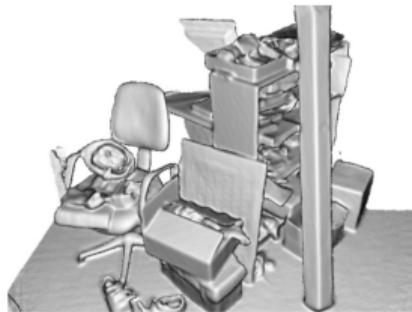
Implicit Neural Representations

Traditional 3D Reconstruction Pipeline



Can we **learn** 3D reconstruction **from data?**

3D Datasets and Repositories



[Newcombe et al., 2011]



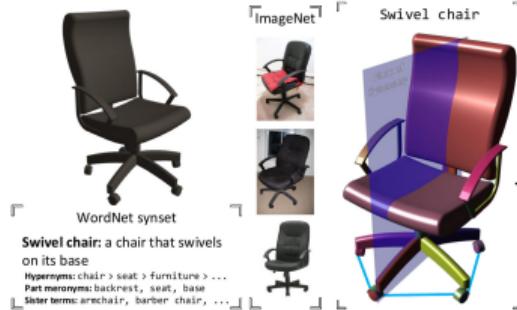
[Choi et al., 2011]



[Dai et al., 2017]



[Wu et al., 2015]



[Chang et al., 2015]



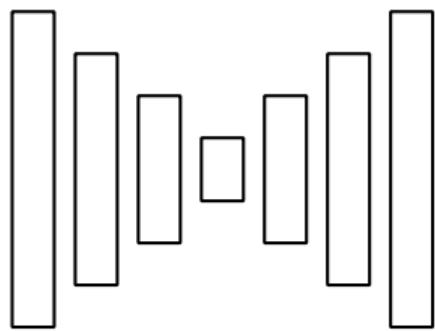
[Chang et al., 2017]

What is a good **output** representation?

3D Reconstruction from a 2D Image



Input Images



Neural Network



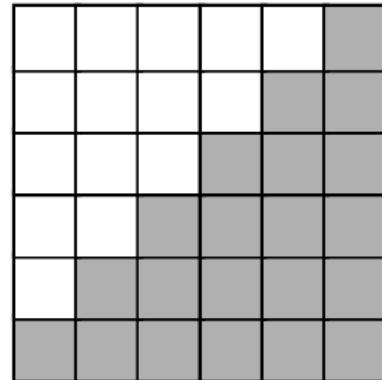
3D Reconstruction

Learning-based 3D Reconstruction

Voxels:

- ▶ **Discretization** of 3D space into grid
- ▶ Easy to process with neural networks
- ▶ Cubic memory $O(n^3)$ ⇒ limited resolution
- ▶ Manhattan world bias

[Maturana et al., IROS 2015]

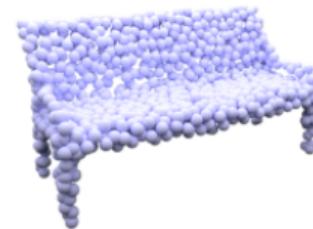
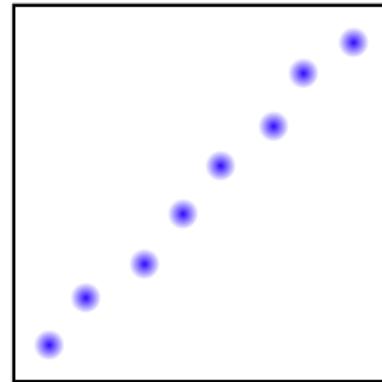


Learning-based 3D Reconstruction

Points:

- ▶ **Discretization** of surface into 3D points
- ▶ Does not model connectivity / topology
- ▶ Limited number of points
- ▶ Global shape description

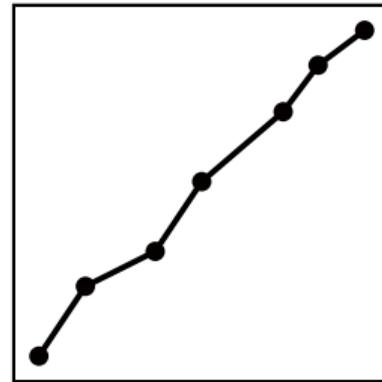
[Fan et al., CVPR 2017]



Learning-based 3D Reconstruction

Meshes:

- ▶ **Discretization** into vertices and faces
- ▶ Limited number of vertices / granularity
- ▶ Requires class-specific template – or –
- ▶ Leads to self-intersections



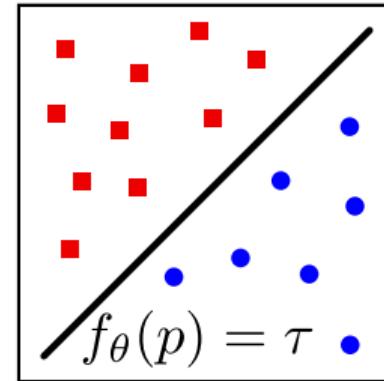
[Groueix et al., CVPR 2018]



Learning-based 3D Reconstruction

This work:

- ▶ Implicit representation \Rightarrow **No discretization**
- ▶ Arbitrary topology & resolution
- ▶ Low memory footprint
- ▶ Not restricted to specific class



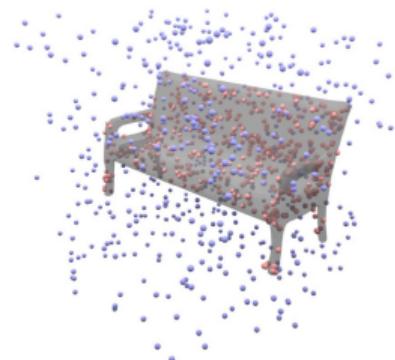
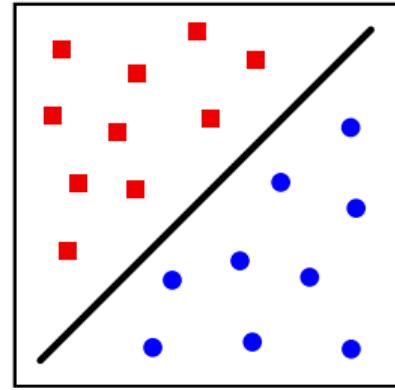
Occupancy Networks

Key Idea:

- ▶ Do not represent 3D shape explicitly
- ▶ Instead, consider surface **implicitly** as **decision boundary** of a non-linear classifier:

$$f_{\theta} : \mathbb{R}^3 \times \mathcal{X} \rightarrow [0, 1]$$

↑ ↑ ↑
3D Location Condition (eg, Image) Occupancy Probability



Remarks:

- ▶ The function f_{θ} models an **occupancy field**

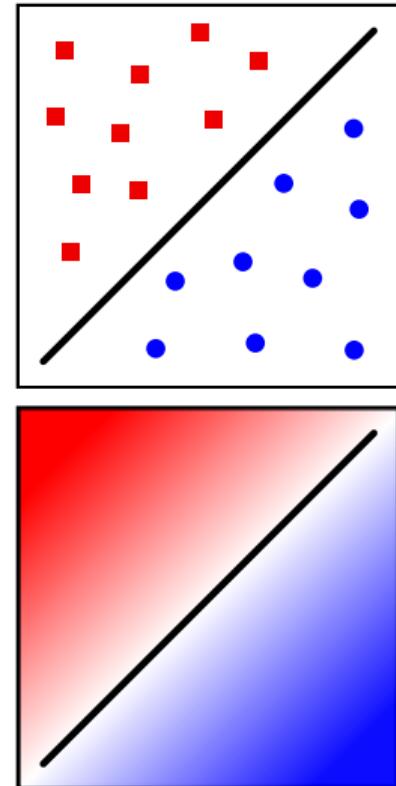
Occupancy Networks

Key Idea:

- ▶ Do not represent 3D shape explicitly
- ▶ Instead, consider surface **implicitly** as **decision boundary** of a non-linear classifier:

$$f_{\theta} : \mathbb{R}^3 \times \mathcal{X} \rightarrow [0, 1]$$

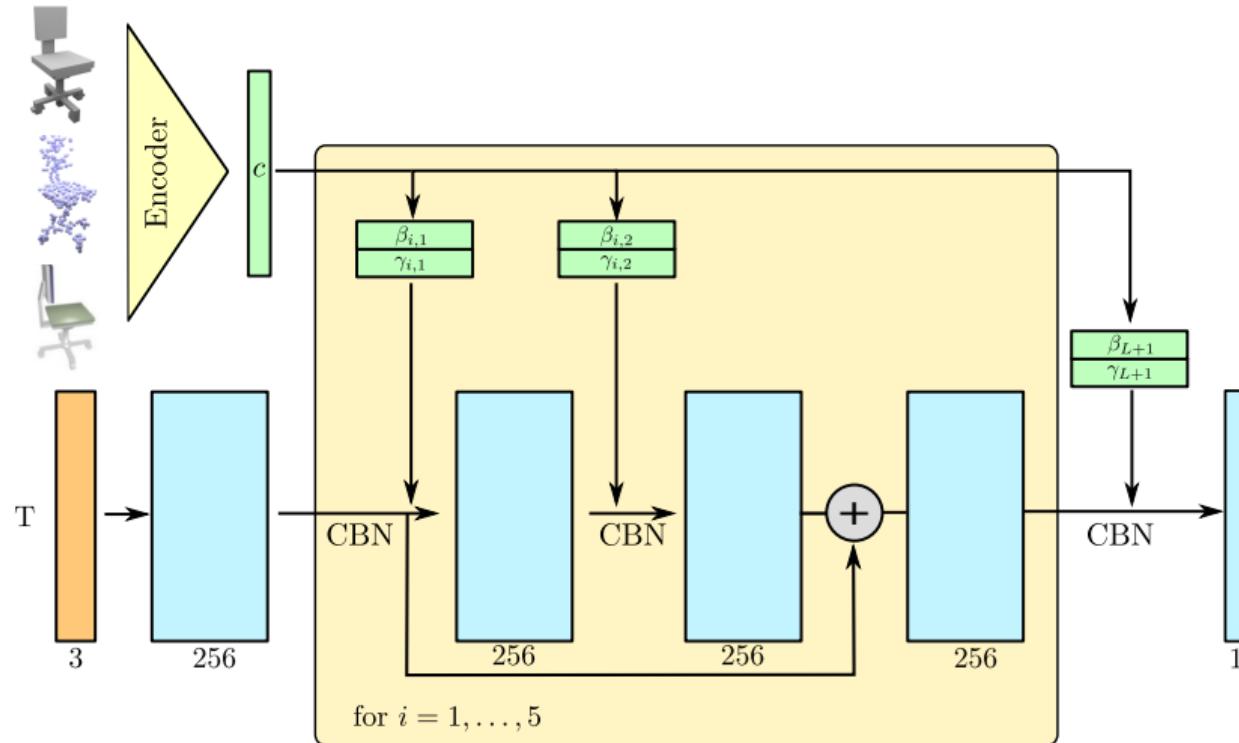
↑ ↑ ↑
3D Location Condition (eg, Image) Occupancy Probability



Remarks:

- ▶ The function f_{θ} models an **occupancy field**
- ▶ Also possible: **signed distance field** [Park et al., 2019]

Network Architecture



Training Objective

Occupancy Network:

$$\mathcal{L}(\theta, \psi) = \sum_{j=1}^K \text{BCE}(f_\theta(p_{ij}, z_i), o_{ij}) + KL [q_\psi(z | (p_{ij}, o_{ij})_{j=1:K}) \| p_0(z)]$$

- ▶ K : Randomly sampled 3D points ($K = 2048$)
- ▶ BCE : Cross-entropy loss

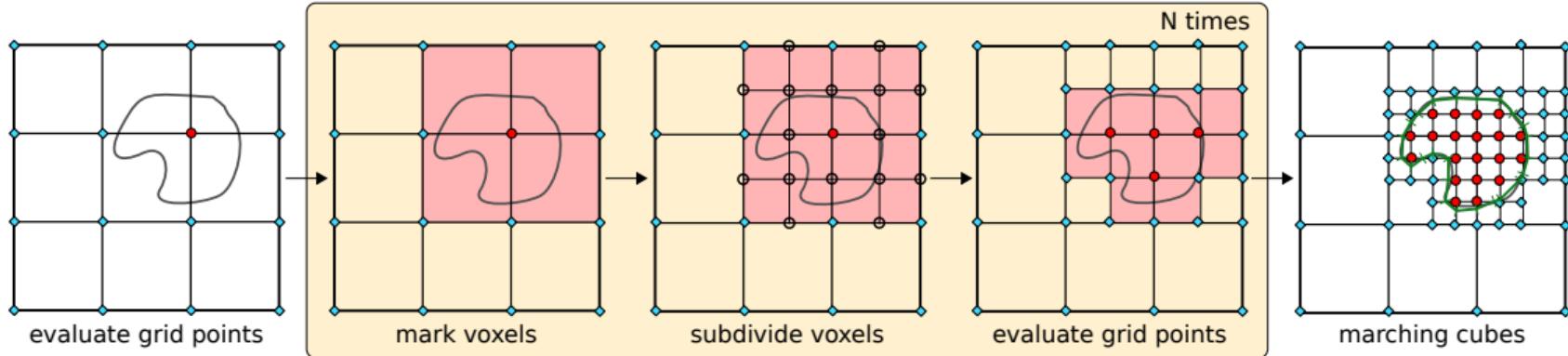
Training Objective

Variational Occupancy Encoder:

$$\mathcal{L}(\theta, \psi) = \sum_{j=1}^K \text{BCE}(f_\theta(p_{ij}, z_i), o_{ij}) + KL [q_\psi(z|(p_{ij}, o_{ij})_{j=1:K}) \| p_0(z)]$$

- ▶ K : Randomly sampled 3D points ($K = 2048$)
- ▶ BCE : Cross-entropy loss
- ▶ q_ψ : Encoder

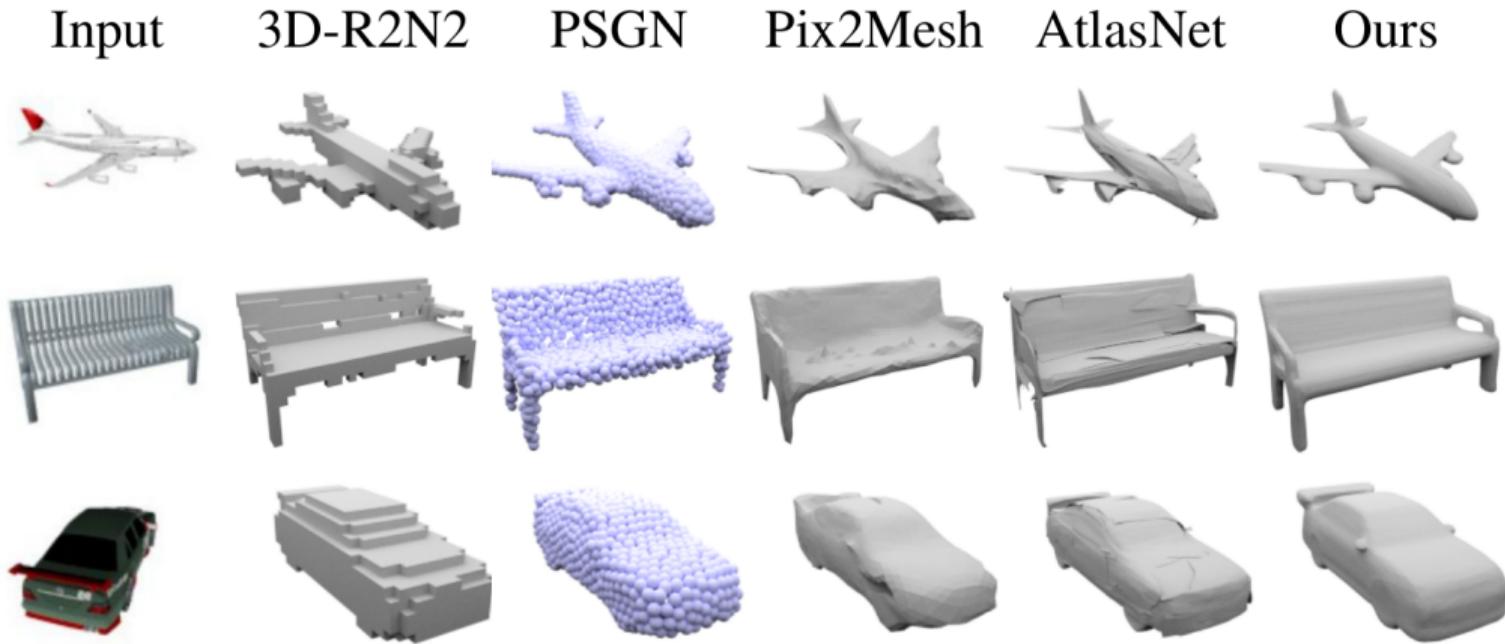
Occupancy Networks



Multiresolution IsoSurface Extraction (MISE):

- ▶ Build octree by incrementally querying the occupancy network
- ▶ Extract triangular mesh using marching cubes algorithm (1-3 seconds in total)

Results



Representing **Materials** and **Lighting**

Problem Definition



Input Image



3D Geometry

Neural
Network

Manipulating the Illumination



Changing the Viewpoint



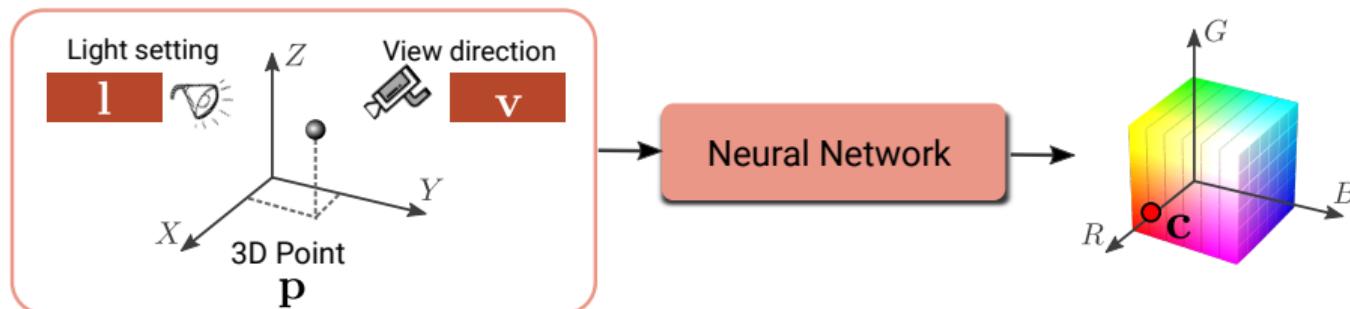
Conditional Surface Light Field

Rendering equation:

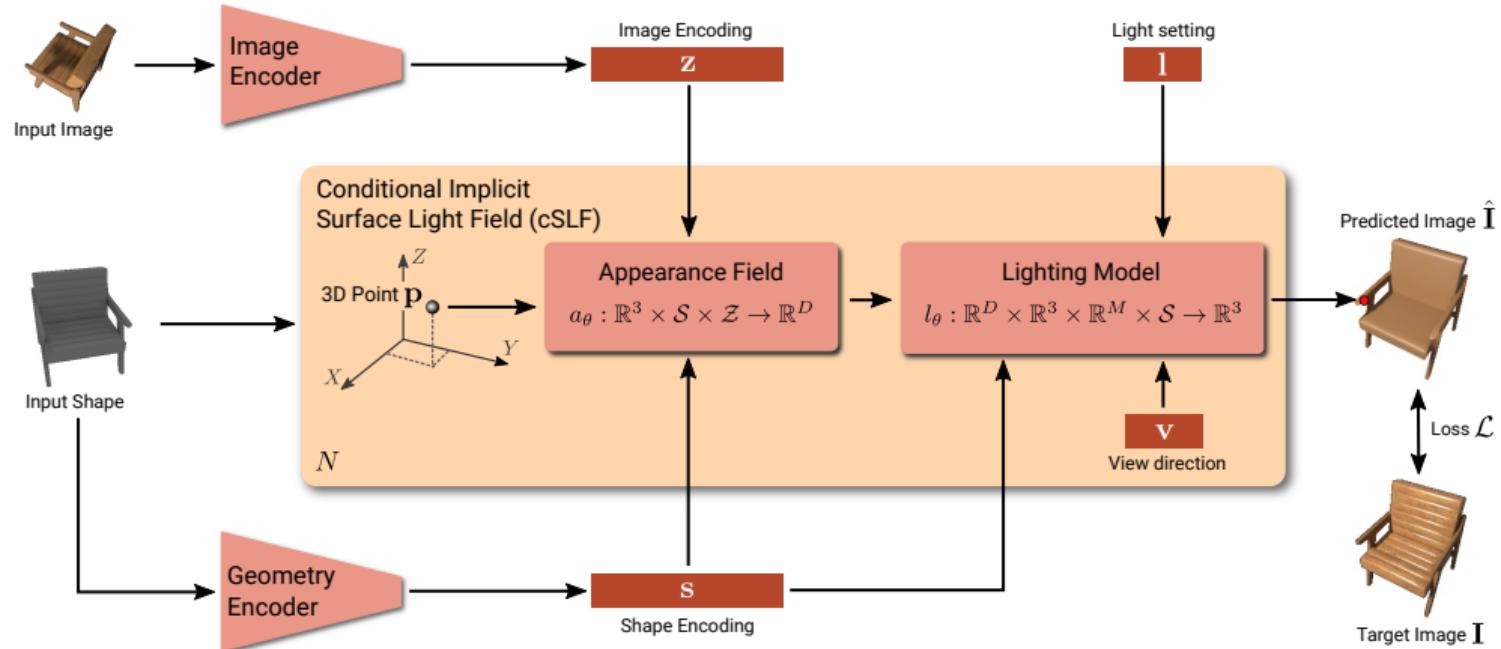
$$L(\mathbf{p}, \mathbf{v}, \mathbf{l}, \mathbf{n}) = \int_{\Omega} \text{BRDF}(\mathbf{p}, \mathbf{s}, \mathbf{v}) \cdot \mathbf{l}(\mathbf{s}) \cdot (\mathbf{n}^T \mathbf{s}) d\mathbf{s}$$

Conditional surface light field:

$$L_{\text{cSLF}}(\mathbf{p}, \mathbf{v}, \mathbf{l}) : \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^M \rightarrow \mathbb{R}^3$$

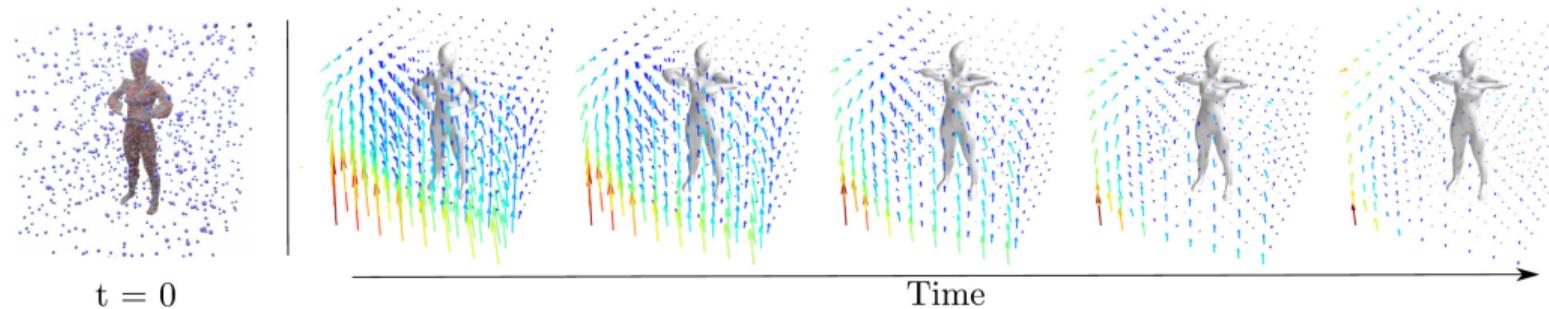


Single-Image Appearance Prediction



Representing **Motion**

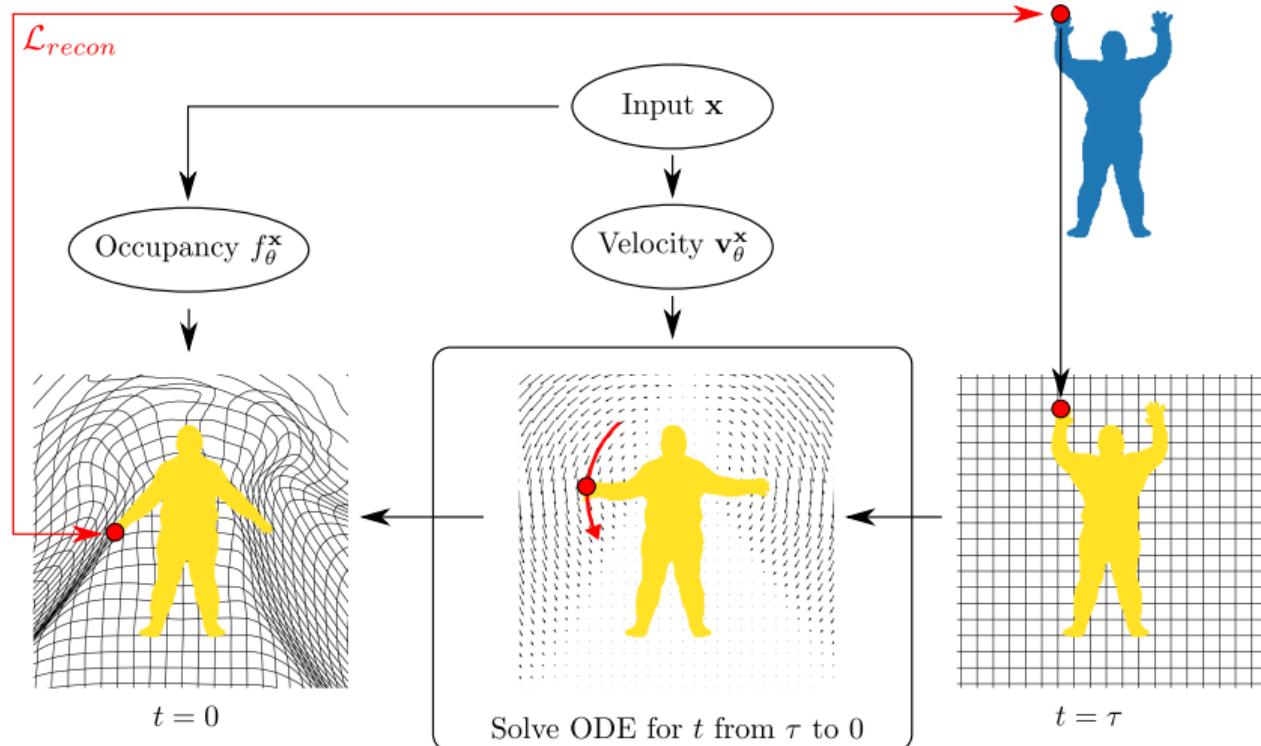
Occupancy Flow



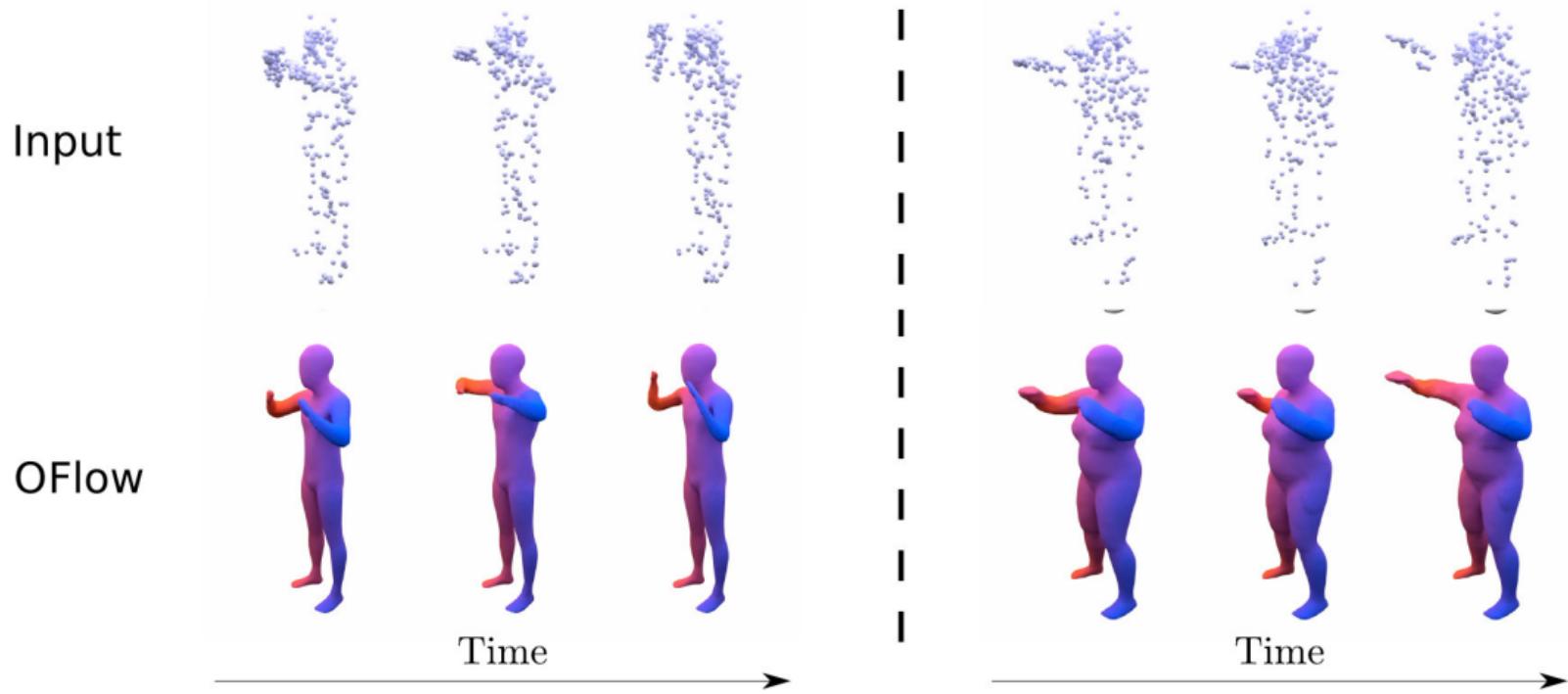
- ▶ Extending Occupancy Networks to 4D is hard (curse of dimensionality)
- ▶ Represent **shape** at $t = 0$ using a 3D Occupancy Network
- ▶ Represent **motion** by temporally and spatially continuous vector field
- ▶ Relationship between 3D trajectory \mathbf{s} and velocity \mathbf{v} given by (differentiable) ODE:

$$\frac{\partial \mathbf{s}(t)}{\partial t} = \mathbf{v}(\mathbf{s}(t), t)$$

Occupancy Flow



Results

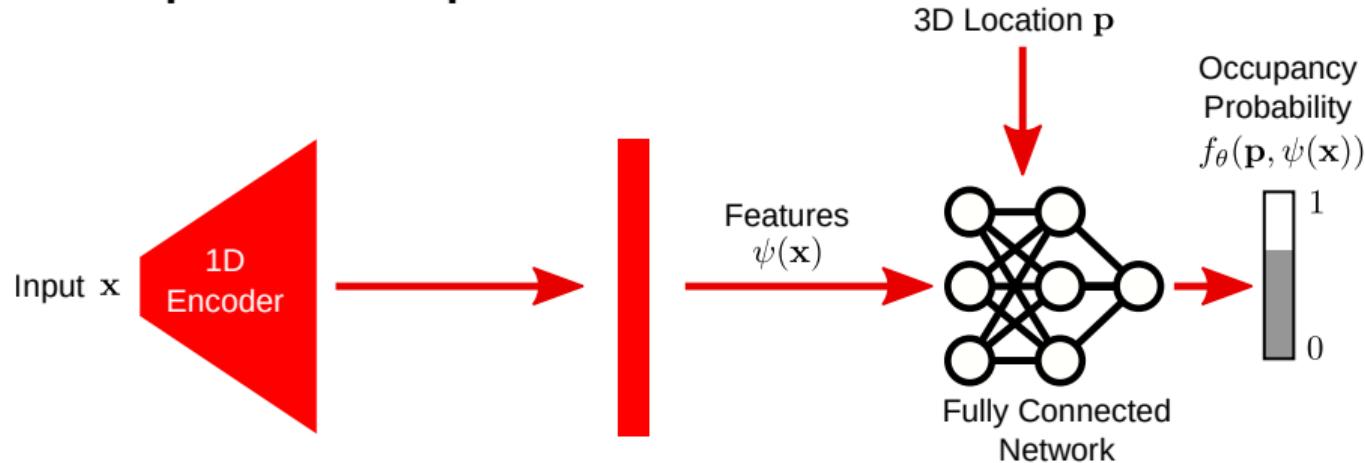


- No **correspondences** needed \Rightarrow **implicitly established** by our model!

Representing **Scenes**

Limitations

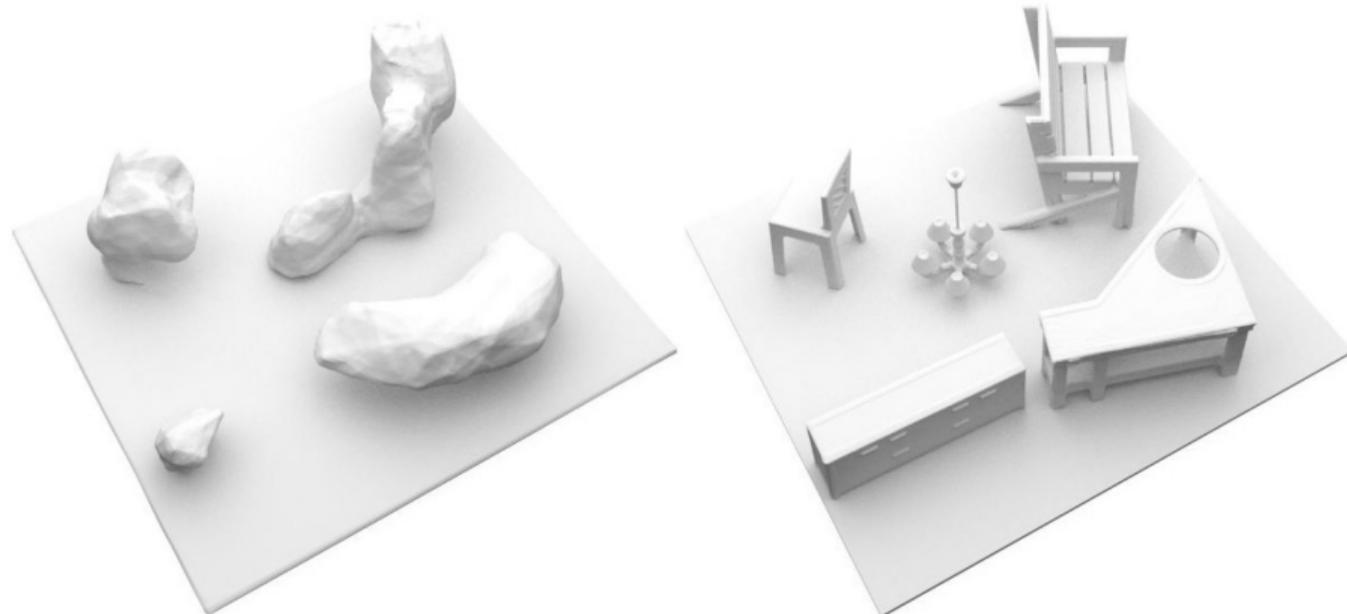
Structure of implicit neural representations:



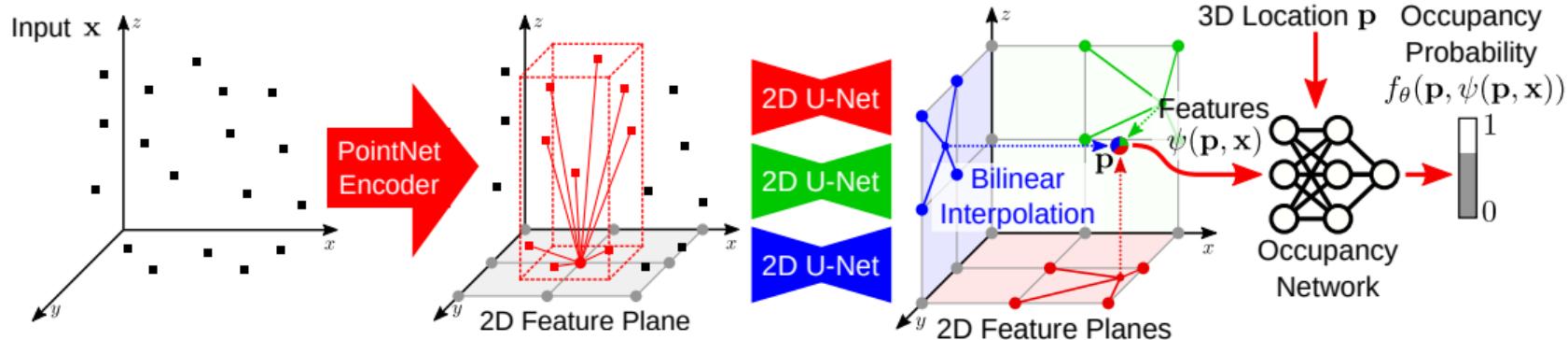
- **Global latent code** \Rightarrow no local information, overly smooth geometry
- **Fully connected architecture** \Rightarrow does not exploit translation equivariance

Limitations

Implicit models work well for simple objects but **poorly on complex scenes:**

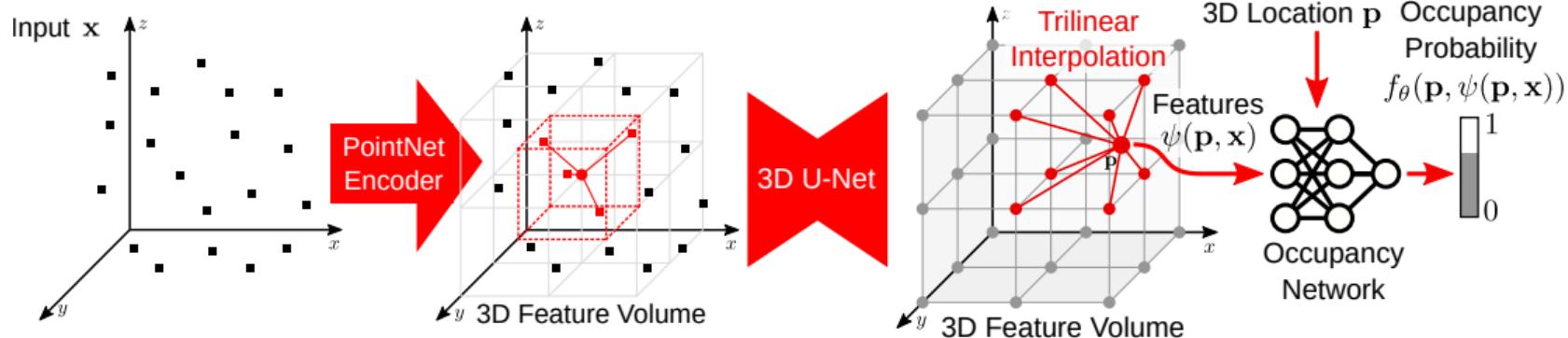


Convolutional Occupancy Networks



- ▶ **2D Plane Encoder:** Local PointNet processes input, project onto canonical plane
- ▶ **2D Plane Decoder:** Processed by U-Net, query features via bilinear interpolation
- ▶ **Occupancy Readout:** Shallow occupancy network $f_\theta(\cdot)$

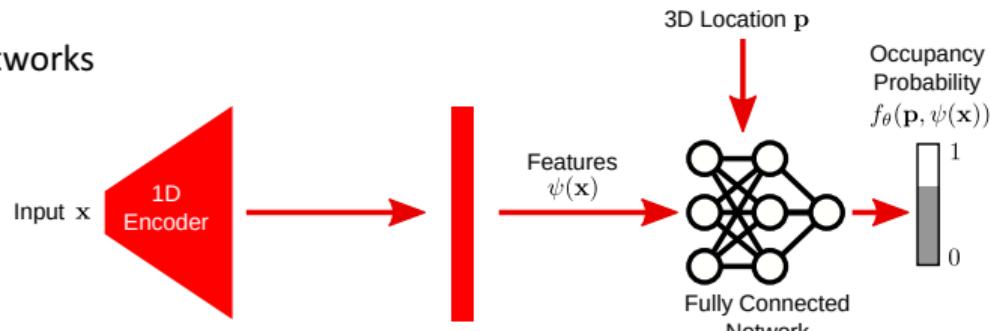
Convolutional Occupancy Networks



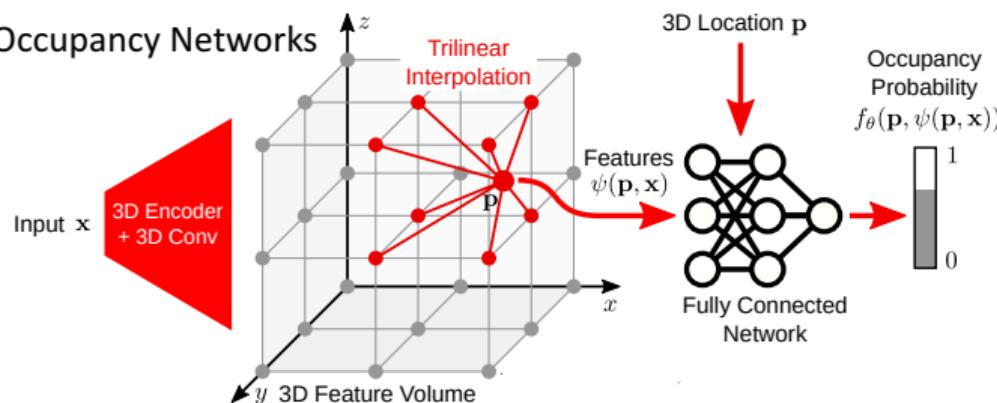
- ▶ **3D Volume Encoder:** Local PointNet processes input, volumetric feature encoding
- ▶ **3D Volume Decoder:** Processed by 3D U-Net, query features via trilinear interp.
- ▶ **Occupancy Readout:** Shallow occupancy network $f_\theta(\cdot)$

Comparison

Occupancy Networks



Convolutional Occupancy Networks

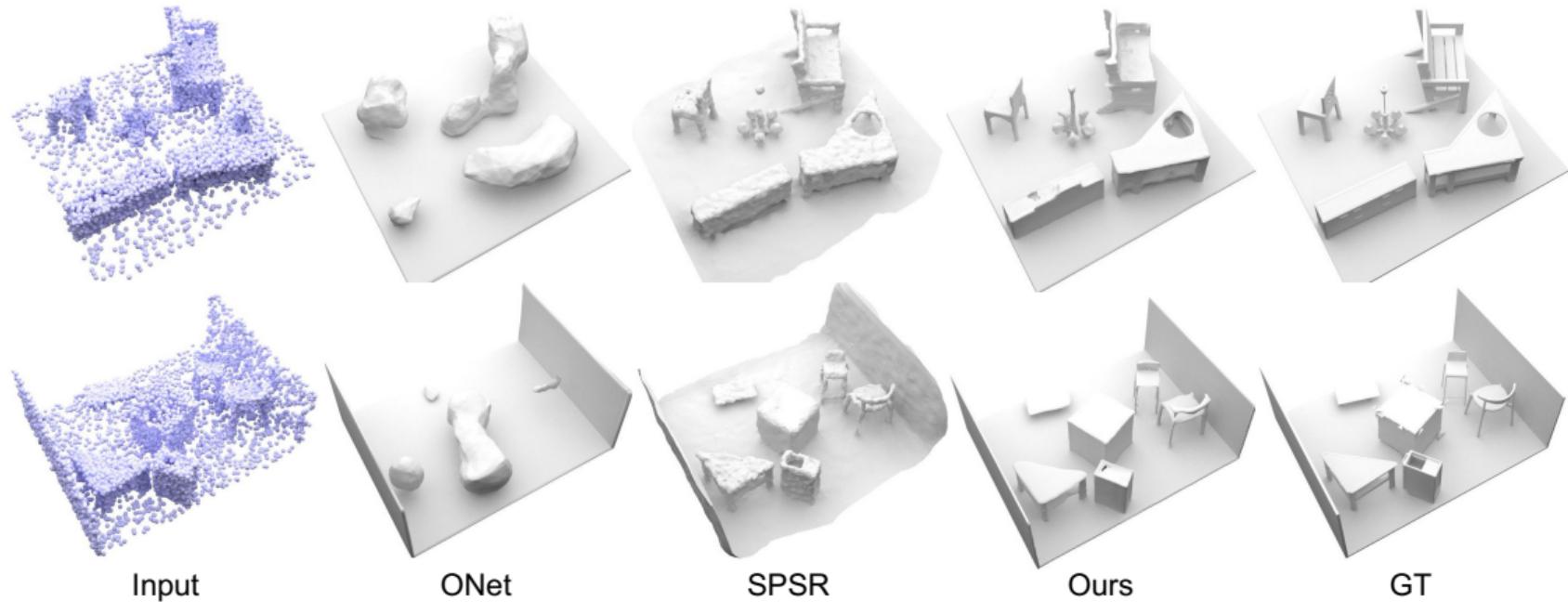


Object-Level Reconstruction



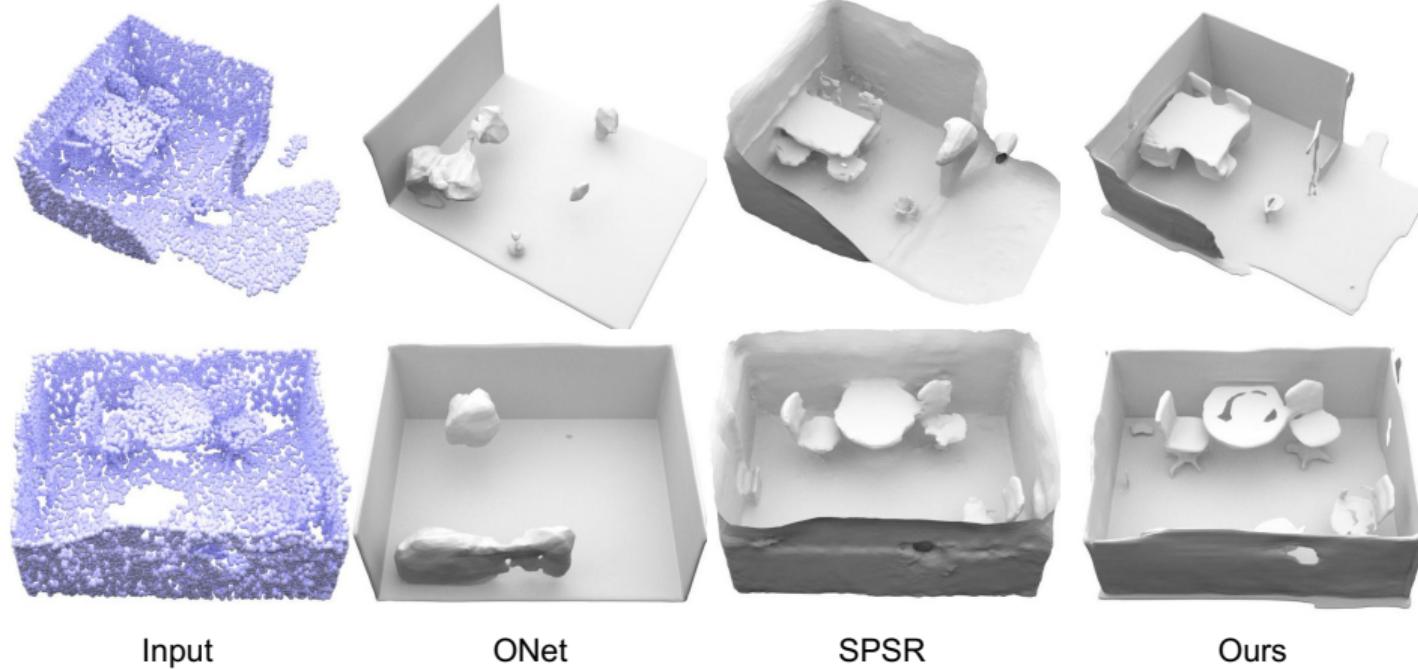
- ▶ Comparison to occupancy networks: **Higher accuracy** and **faster convergence**

Scene-Level Reconstruction



- Trained and evaluated on **synthetic rooms**

Scene-Level Reconstruction

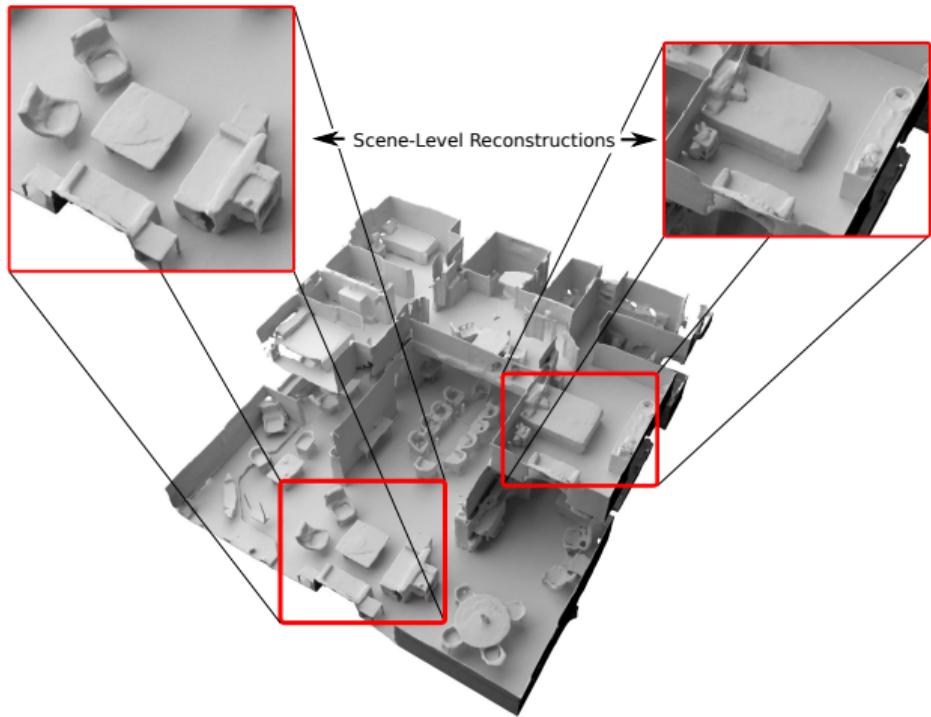


- Trained on **synthetic rooms** and evaluated on **ScanNet**

Large-Scale Reconstruction

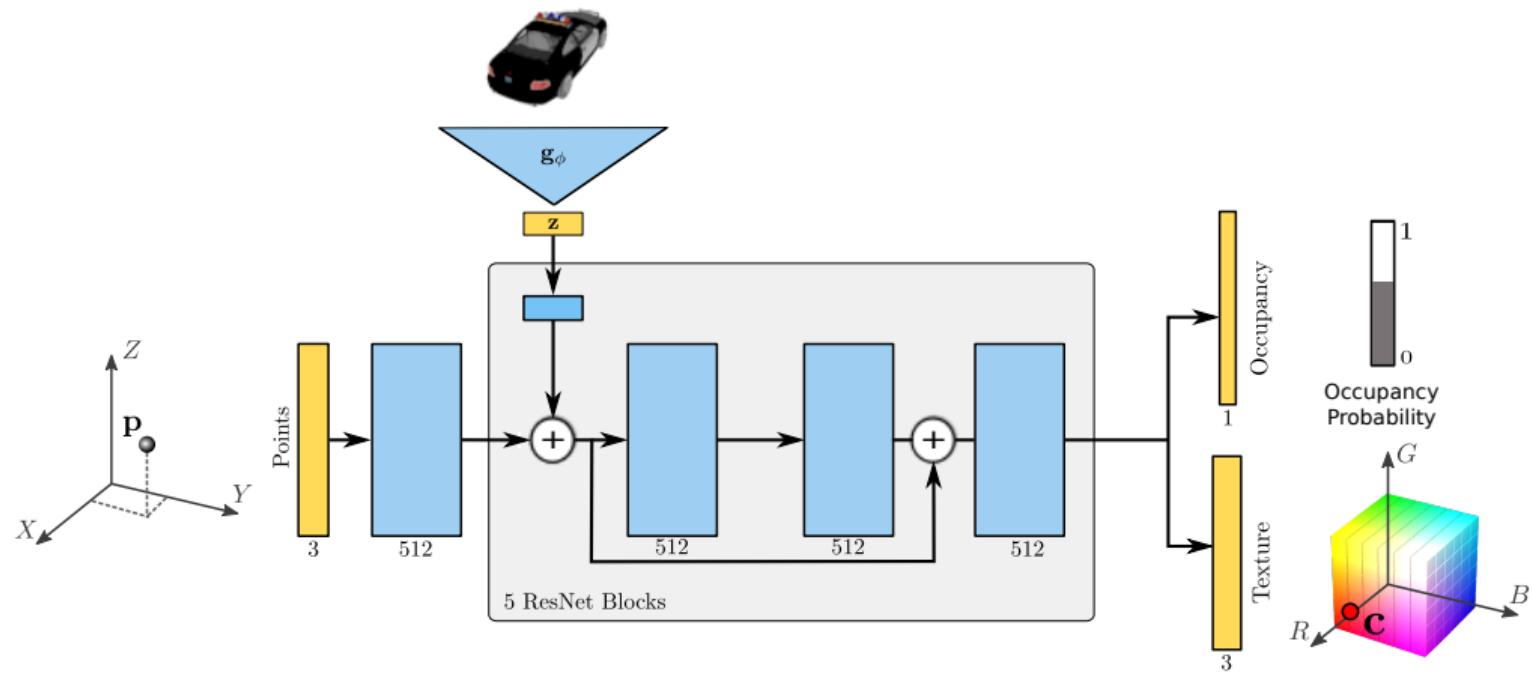
Results on Matterport3D

- ▶ Fully convolutional model
- ▶ Trained on synthetic crops
- ▶ **Sliding window** inference
- ▶ Scales to any scene size



Learning from **Images**

DVR: Differentiable Volumetric Rendering

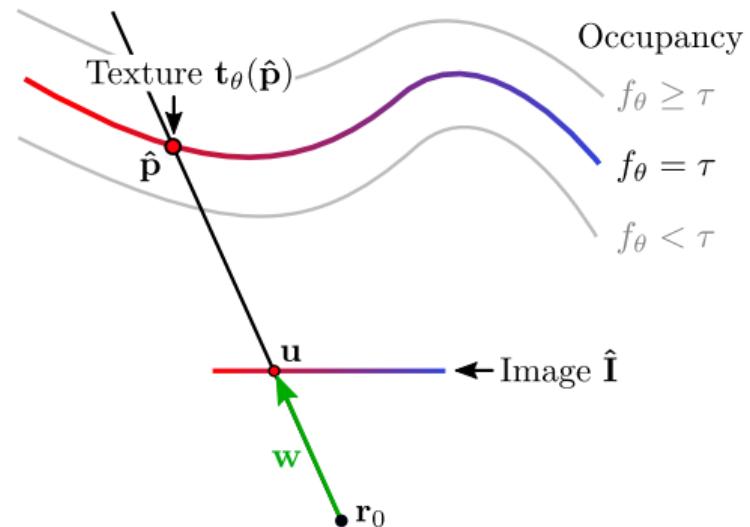


Forward Pass (Rendering)

DVR: Differentiable Volumetric Rendering

Forward Pass:

- ▶ For all pixels \mathbf{u}
- ▶ Find surface point $\hat{\mathbf{p}}$ along ray \mathbf{w} via ray marching and root finding (secant method)
- ▶ Evaluate texture field $\mathbf{t}_\theta(\hat{\mathbf{p}})$ at $\hat{\mathbf{p}}$
- ▶ Insert color $\mathbf{t}_\theta(\hat{\mathbf{p}})$ at pixel \mathbf{u}



Secant Method

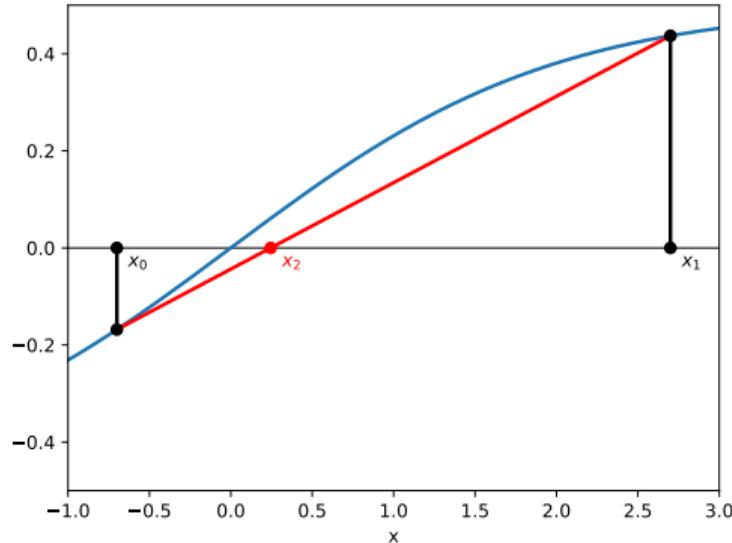
Consider the **line** $(x_0, f(x_0)) \rightarrow (x_1, f(x_1))$:

$$y_2 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} (x_2 - x_1) + f(x_1)$$

The **root** of this function at $y_2 = 0$ is:

$$x_2 = x_1 - f(x_1) \frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

Iteratively applying this rule yields the **secant method** which is a finite-difference approximation of Newton's method.



Secant Method

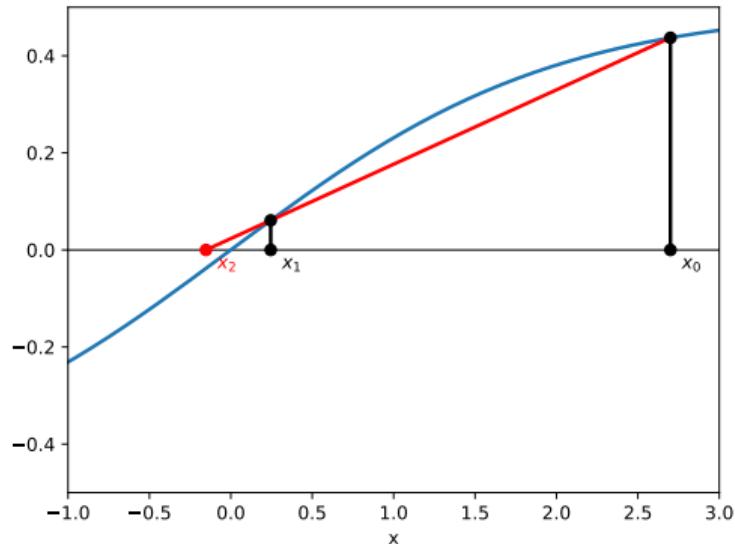
Consider the **line** $(x_0, f(x_0)) \rightarrow (x_1, f(x_1))$:

$$y_2 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} (x_2 - x_1) + f(x_1)$$

The **root** of this function at $y_2 = 0$ is:

$$x_2 = x_1 - f(x_1) \frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

Iteratively applying this rule yields the **secant method** which is a finite-difference approximation of Newton's method.



Secant Method

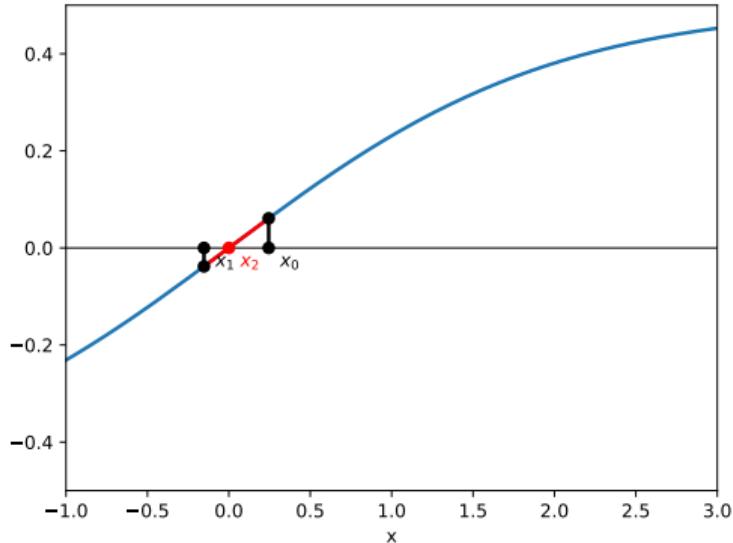
Consider the **line** $(x_0, f(x_0)) \rightarrow (x_1, f(x_1))$:

$$y_2 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} (x_2 - x_1) + f(x_1)$$

The **root** of this function at $y_2 = 0$ is:

$$x_2 = x_1 - f(x_1) \frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

Iteratively applying this rule yields the **secant method** which is a finite-difference approximation of Newton's method.



Backward Pass (Differentiation)

Total Derivative

Let $f(x, y)$ be a function where $y = y(x)$ depends on x .

The **partial derivative** is defined as:

$$\frac{\partial f(x, y)}{\partial x} = \frac{\partial f}{\partial x} \frac{\partial x}{\partial x}$$

The **total derivative** is defined as:

$$\frac{df(x, y)}{dx} = \frac{\partial f}{\partial x} \frac{\partial x}{\partial x} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial x}$$

Example:

$$f(x, y) = xy$$
$$\frac{\partial f(x, y)}{\partial x} = y$$

Example:

$$f(x, y) = xy \wedge y = x$$
$$\frac{df(x, y)}{dx} = y + x = 2x$$

Implicit Functions and Implicit Differentiation

An **implicit equation** is a relation

$$f(x, y) = 0$$

where $y(x)$ is defined only implicitly.

Implicit differentiation computes the total derivative of both sides wrt. x

$$\frac{\partial f}{\partial x} \frac{\partial x}{\partial x} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial x} = 0$$

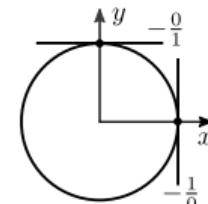
and solves for $\frac{\partial y}{\partial x}$.

Example: Let's assume

$$x^2 + y^2 = 1$$

Implicit differentiation yields

$$2x + 2y \frac{\partial y}{\partial x} = 0$$
$$\frac{\partial y}{\partial x} = -\frac{x}{y}$$



Note the presence of y in this term, i.e., the implicit “function” is a curve.

DVR: Differentiable Volumetric Rendering

Backward Pass:

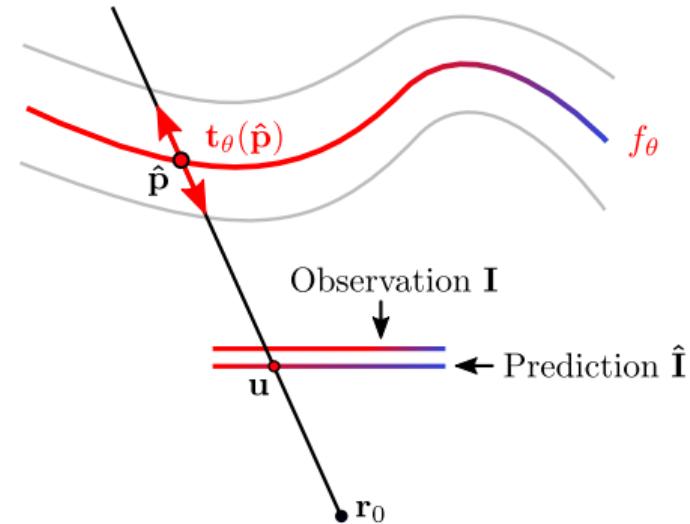
- ▶ Image Observation \mathbf{I}
- ▶ Loss $\mathcal{L}(\hat{\mathbf{I}}, \mathbf{I}) = \sum_{\mathbf{u}} \|\hat{\mathbf{I}}_{\mathbf{u}} - \mathbf{I}_{\mathbf{u}}\|$
- ▶ Gradient of loss function:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{\mathbf{u}} \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{I}}_{\mathbf{u}}} \cdot \frac{\partial \hat{\mathbf{I}}_{\mathbf{u}}}{\partial \theta}$$

$$\frac{\partial \hat{\mathbf{I}}_{\mathbf{u}}}{\partial \theta} = \frac{\partial \mathbf{t}_{\theta}(\hat{\mathbf{p}})}{\partial \theta} + \frac{\partial \mathbf{t}_{\theta}(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \cdot \frac{\partial \hat{\mathbf{p}}}{\partial \theta}$$

- ▶ Differentiation of $f_{\theta}(\hat{\mathbf{p}}) = \tau$ yields:

$$\frac{\partial \hat{\mathbf{p}}}{\partial \theta} = -\mathbf{w} \left(\frac{\partial f_{\theta}(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \cdot \mathbf{w} \right)^{-1} \frac{\partial f_{\theta}(\hat{\mathbf{p}})}{\partial \theta}$$



⇒ **Analytic solution** and **no need** for storing **intermediate results**

DVR: Differentiable Volumetric Rendering

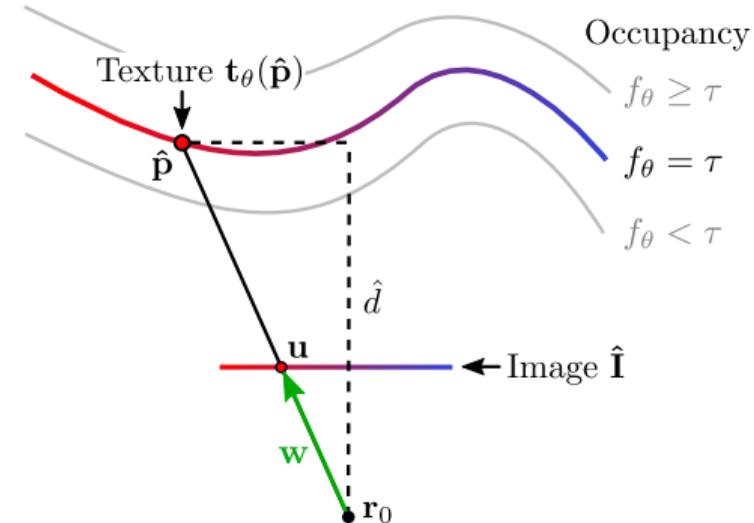
Consider the ray: $\hat{\mathbf{p}} = \mathbf{r}_0 + \hat{d}\mathbf{w}$

- By implicit differentiation of $f_\theta(\hat{\mathbf{p}}) = \tau$ (i.e., on both sides) wrt. θ , we obtain:

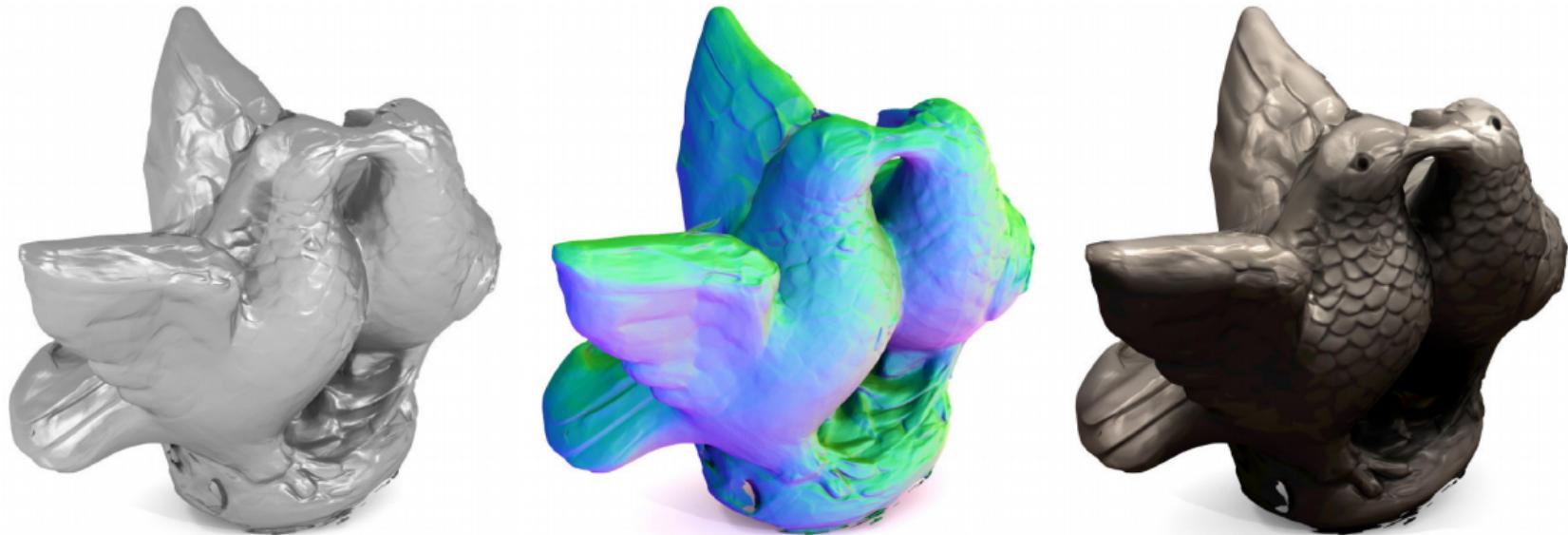
$$\begin{aligned} \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \theta} + \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \cdot \frac{\partial \hat{\mathbf{p}}}{\partial \theta} &= 0 \\ \Leftrightarrow \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \theta} + \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \cdot \mathbf{w} \frac{\partial \hat{d}}{\partial \theta} &= 0 \end{aligned}$$

- Solving for $\frac{\partial \hat{d}}{\partial \theta}$, we obtain:

$$\frac{\partial \hat{\mathbf{p}}}{\partial \theta} = \mathbf{w} \frac{\partial \hat{d}}{\partial \theta} = -\mathbf{w} \left(\frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \cdot \mathbf{w} \right)^{-1} \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \theta}$$



Results



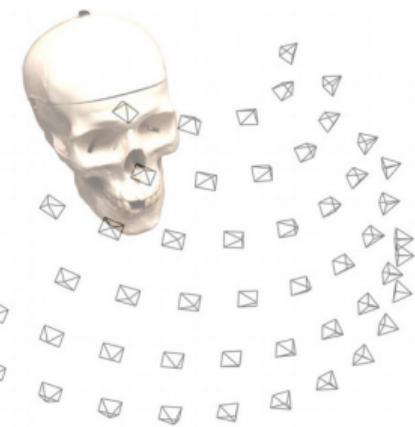
Implicit Differentiable Renderer



3D Surface



Light & Reflectance



Cameras

Related work by Lipman et al.:

- ▶ Condition on surface normal and view direction for **view-dependent appearance**
- ▶ Optimize geometry, appearance and **camera poses**

9.3

Neural Radiance Fields

Novel View Synthesis

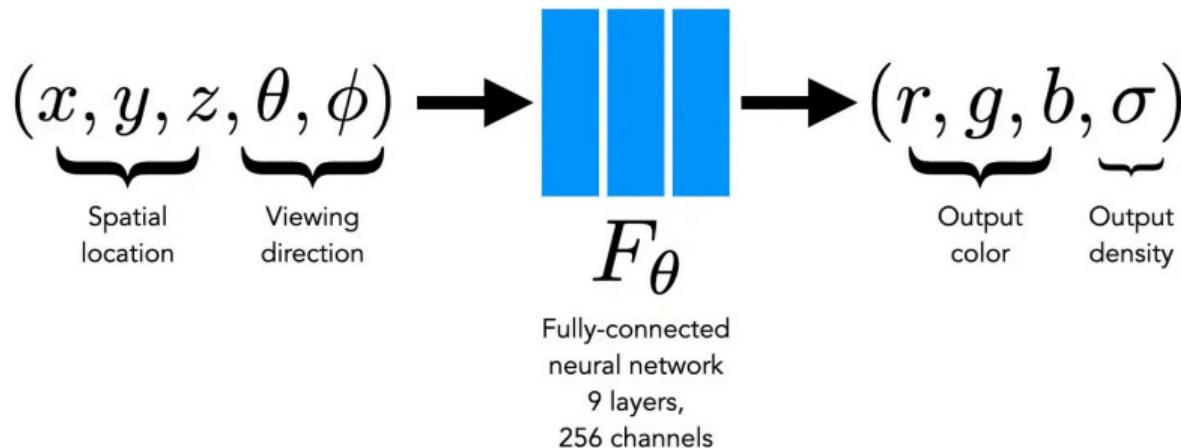


Inputs: sparsely sampled images of scene

Outputs: *new views of same scene*

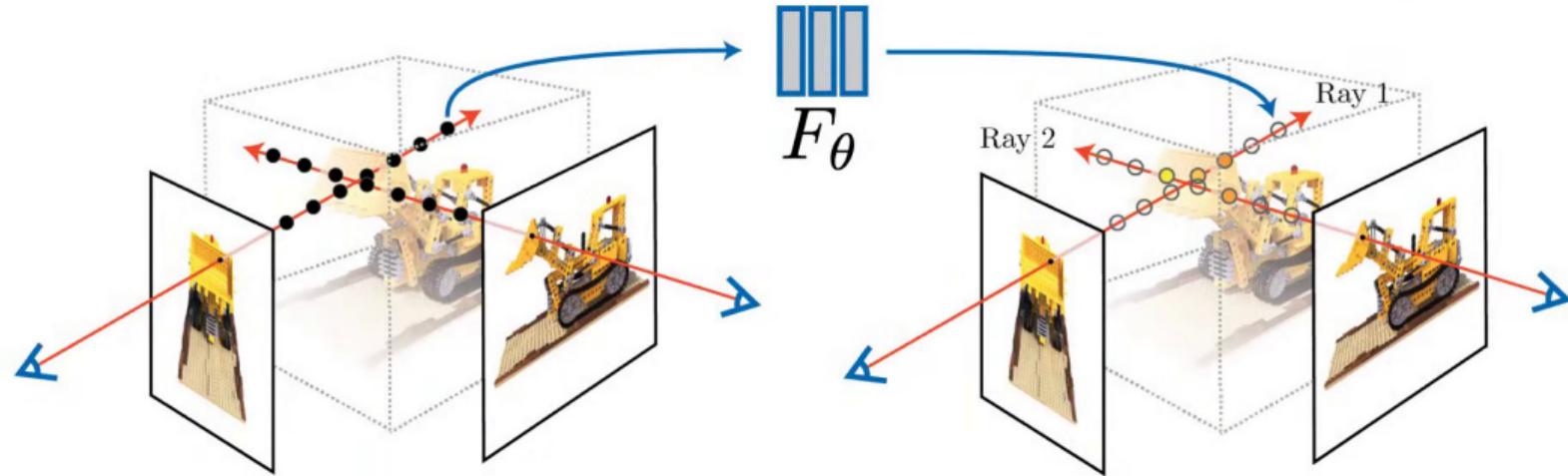
- ▶ **Task:** Given a set of images of a scene, render image from novel viewpoint
- ▶ Slide credits: Ben Mildenhall and Jon Barron

NeRF: Representing Scenes as Neural Radiance Fields



- ▶ Vanilla **ReLU MLP** that maps from **location/view direction to color/density**
- ▶ **Density** σ describes how solid/transparent a 3D point is (can model, e.g., fog)
- ▶ Conditioning on view direction allows for modeling **view-dependent effects**
- ▶ In practice, the view direction is input as a normalized 3D vector \mathbf{d} , not (θ, ϕ)

Volume Rendering



- ▶ **Volume rendering** works very similar to traditional **ray tracing** in graphics
- ▶ Shoot ray through the scene, sample points along ray, query radiance field to obtain color/density, apply alpha composition to obtain pixel color

Volume Rendering

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

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

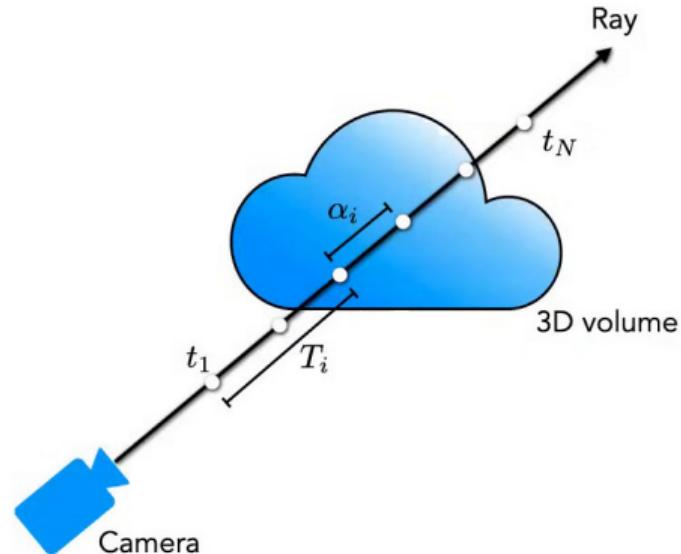
weights colors

How much light is blocked earlier along ray:

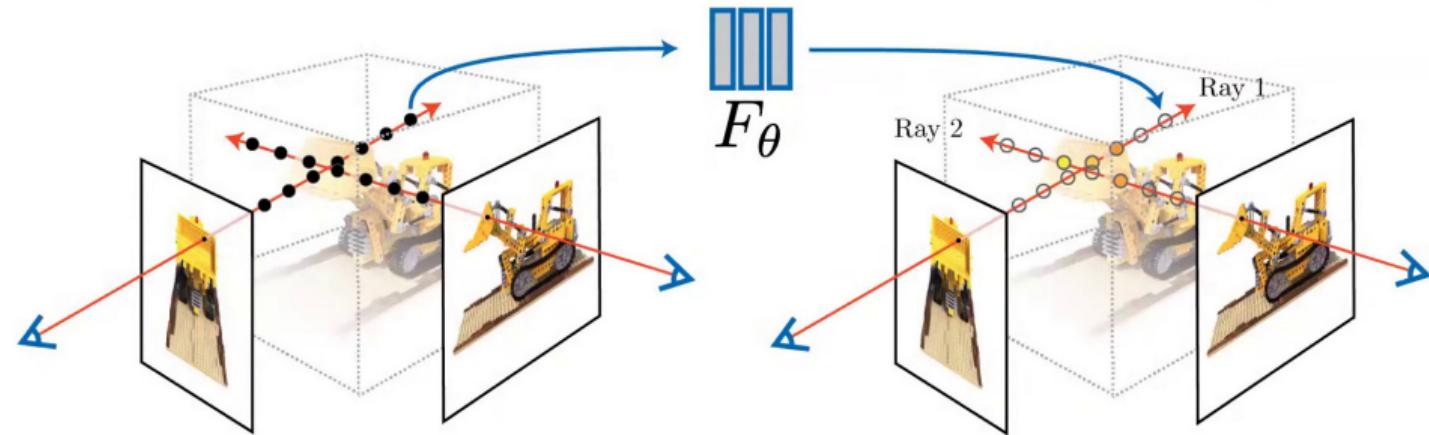
$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$

How much light is contributed by ray segment i :

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



NeRF Training



$$\min_{\theta} \sum_i \| \text{render}_i(F_\theta) - I_i \|^2$$

- Shoot ray, render ray to pixel, minimize **reconstruction error** via backpropagation

NeRF Training

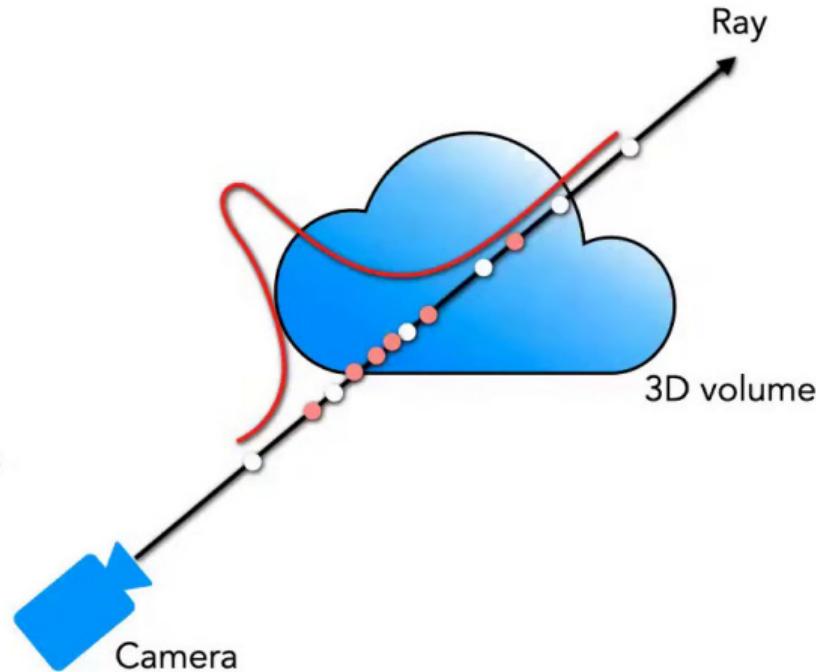


- NeRF's parameters are optimized on **many different views** of a single scene

Two Pass Rendering

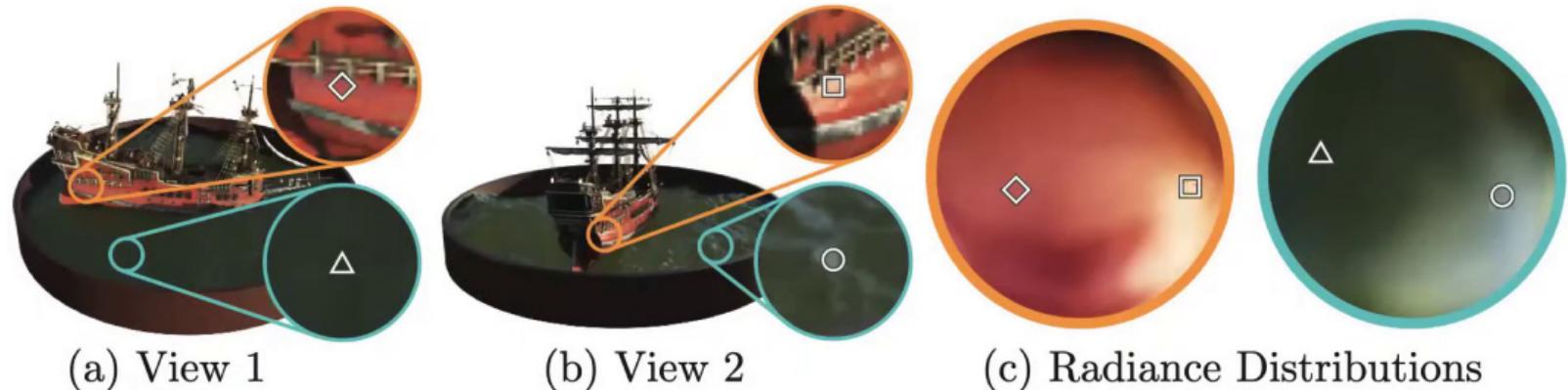
$$C \approx \sum_{i=1}^N T_i \alpha_i c_i$$

treat weights as probability distribution for new samples



- To allocate samples more efficiently, **sampling** is performed in **two passes**

Dependency on View Direction



- ▶ NeRF models **view-dependent effects** (e.g., non-Lambertian materials)
- ▶ Viewpoint condition happens later in MLP to not entangle with the geometry
- ▶ View dependency can be visualized by querying color for different view directions

Fourier Features



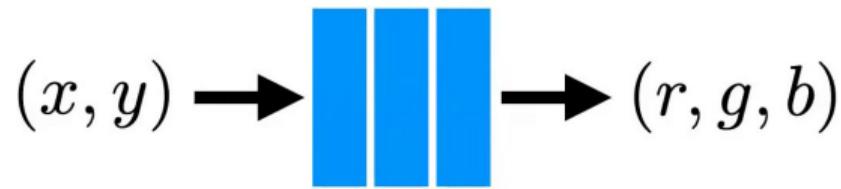
NeRF (Naive)



NeRF (with positional encoding)

- Essential trick: Compute **positional encoding** for input point \mathbf{x} and direction \mathbf{d}

Toy Example



Memorizing a single RGB Image:

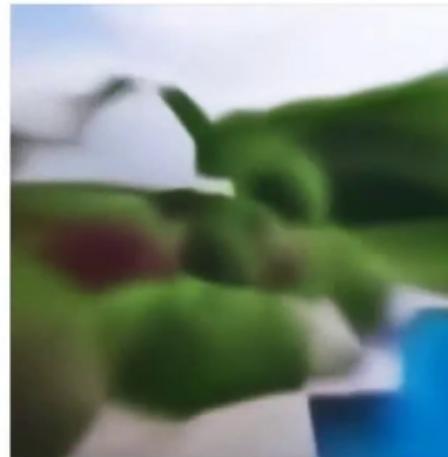
- ▶ Simple MLP which maps from **pixel location** (x, y) to **RGB color** (r, g, b)

Toy Example: Memorizing a single RGB Image

Ground truth image

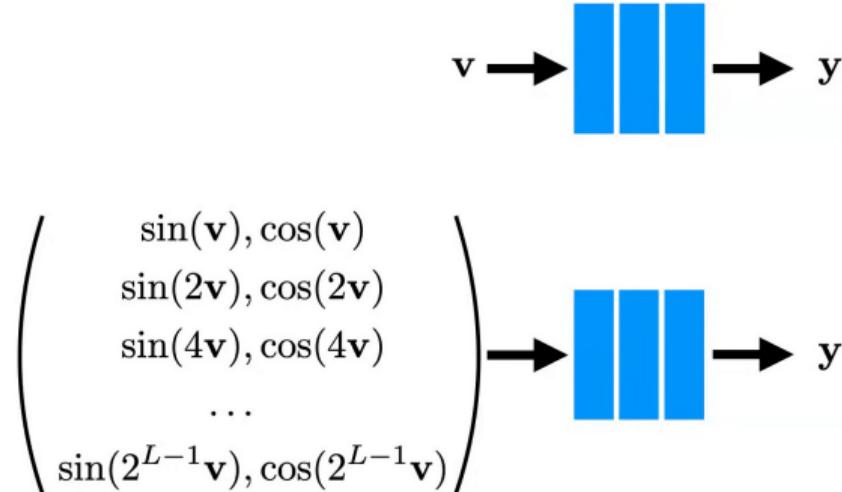


Standard fully-connected net



- ▶ Surprisingly, this does not work! Results are oversmooth with ReLU artifacts

Positional Encoding / Fourier Features



- ▶ Solution: pass low-dimensional coordinates through fixed **positional encoding** or **random Fourier features** of varying frequencies controlled by L
- ▶ These features let networks **learn high-freq. functions** in low-dim. domains

Positional Encoding / Fourier Features

Ground truth image



Standard fully-connected net

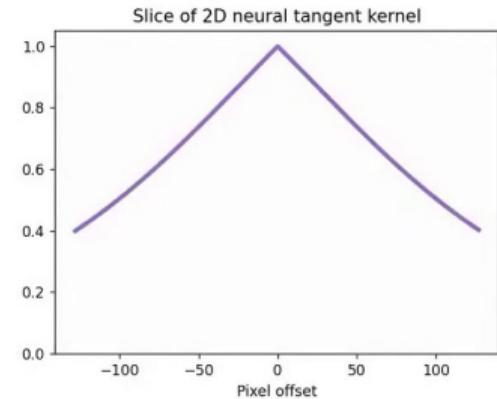
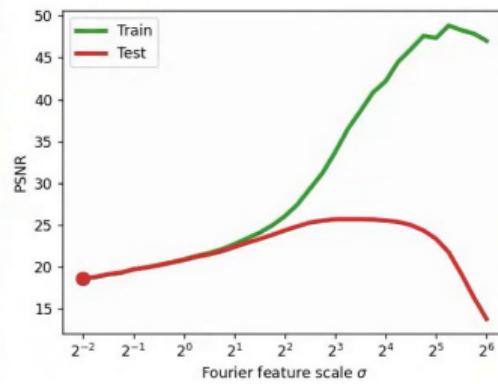
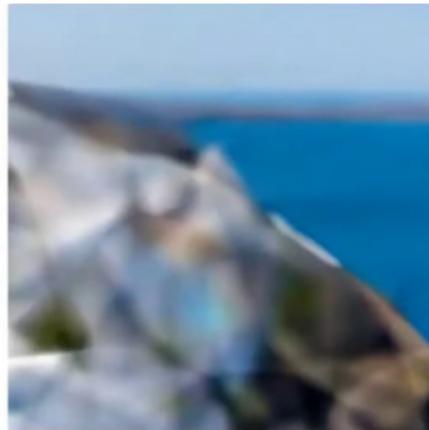


With Positional Encoding



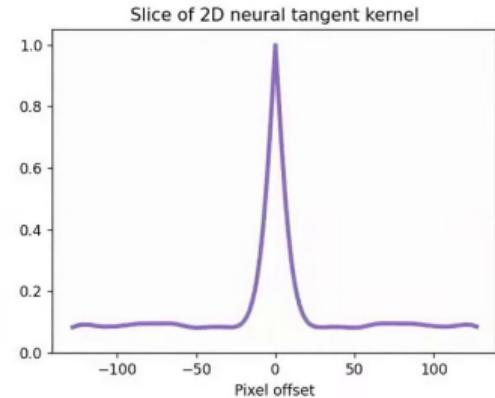
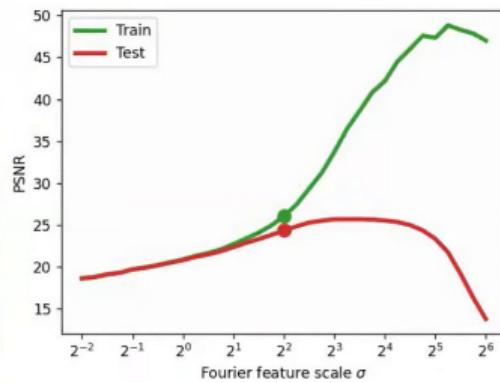
- Results **sharper and more detailed**, no ReLU artifacts, faster convergence

Mapping Bandwidth controls Underfitting/Overfitting



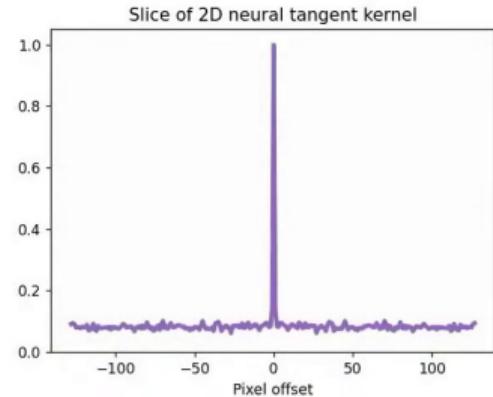
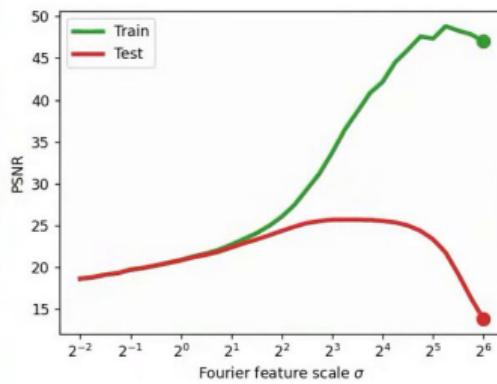
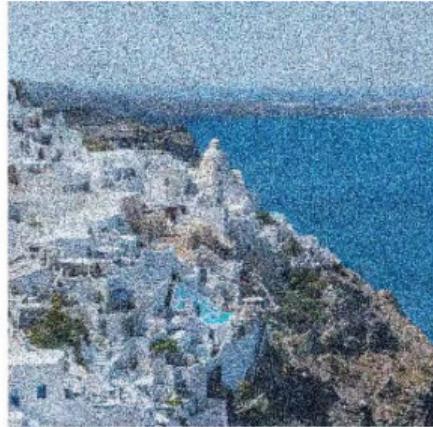
$$\gamma(\mathbf{v}) = [\cos(\mathbf{B}\mathbf{v}), \sin(\mathbf{B}\mathbf{v})] \quad \mathbf{B} \sim \mathcal{N}(0, \sigma^2)$$

Mapping Bandwidth controls Underfitting/Overfitting



$$\gamma(\mathbf{v}) = [\cos(\mathbf{B}\mathbf{v}), \sin(\mathbf{B}\mathbf{v})] \quad \mathbf{B} \sim \mathcal{N}(0, \sigma^2)$$

Mapping Bandwidth controls Underfitting/Overfitting



$$\gamma(\mathbf{v}) = [\cos(\mathbf{B}\mathbf{v}), \sin(\mathbf{B}\mathbf{v})] \quad \mathbf{B} \sim \mathcal{N}(0, \sigma^2)$$

NeRF Results



9.4

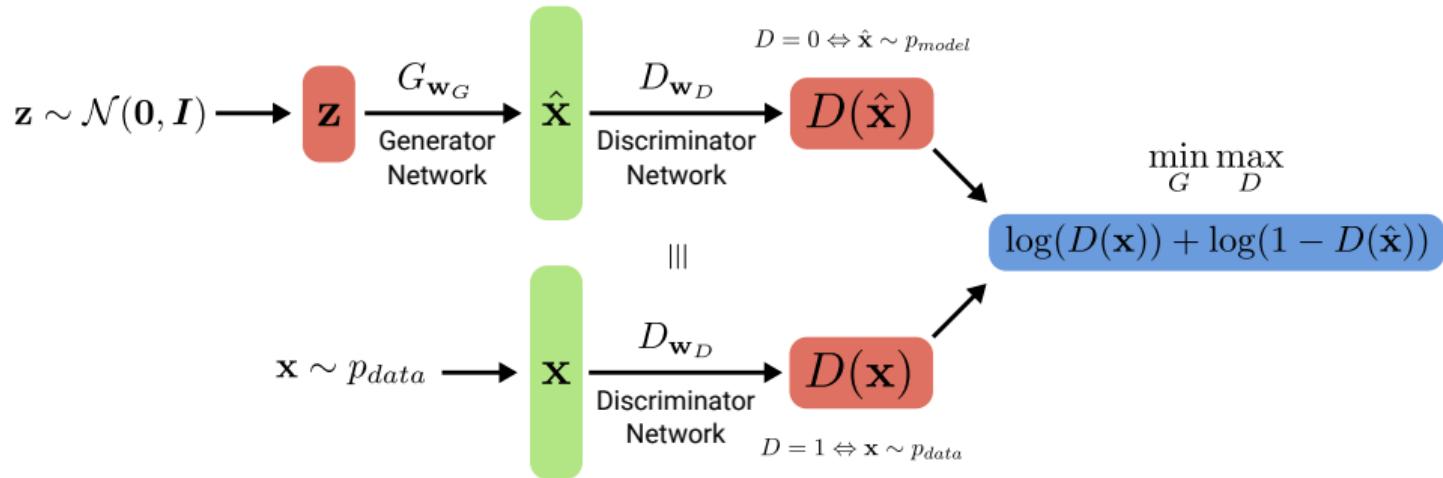
Generative Radiance Fields

Recap: Generative Adversarial Networks

Generative Models

- ▶ The term **generative model** refers to any model that takes a dataset drawn from p_{data} and learns a probability distribution p_{model} to represent p_{data}
- ▶ In some cases, the model estimates p_{model} **explicitly** and therefore allow for evaluating the (approximate) likelihood $p_{model}(\mathbf{x})$ of a sample \mathbf{x}
- ▶ In other cases, the model is only able to **generate samples** from p_{model}
- ▶ GANs are prominent examples of this family of **implicit models**
- ▶ They provide a framework for training models without explicit likelihood

Generative Adversarial Networks



D and G play the following **two-player minimax game** with value function $V(G, D)$:

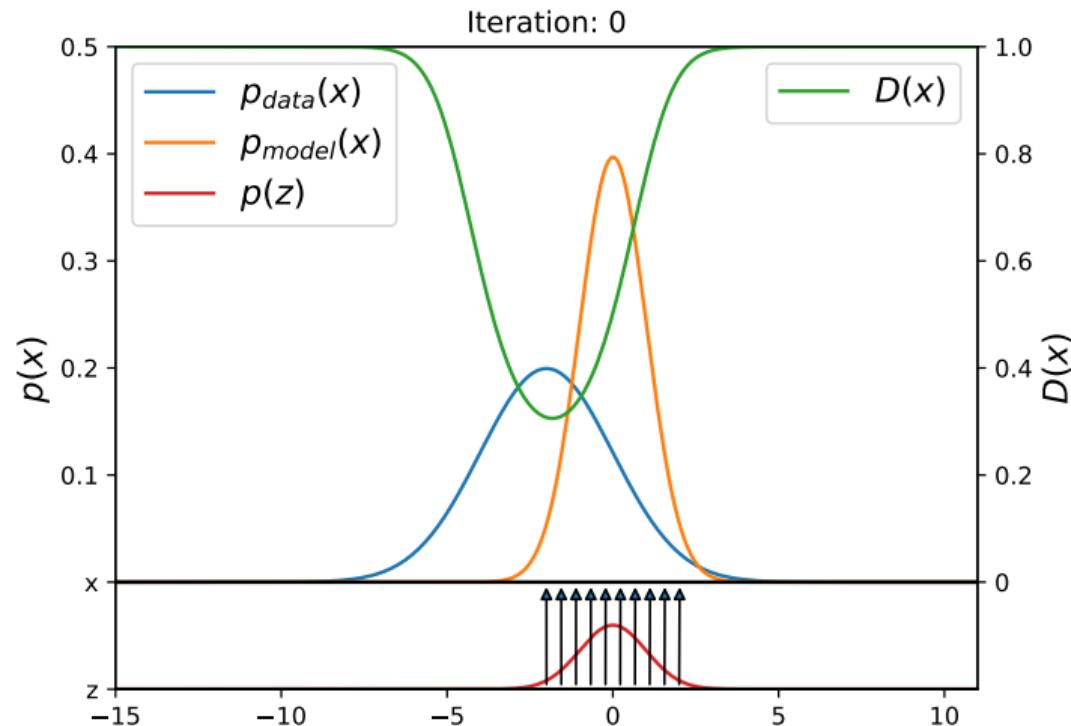
$$G^*, D^* = \operatorname{argmin}_G \operatorname{argmax}_D V(D, G)$$

$$V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

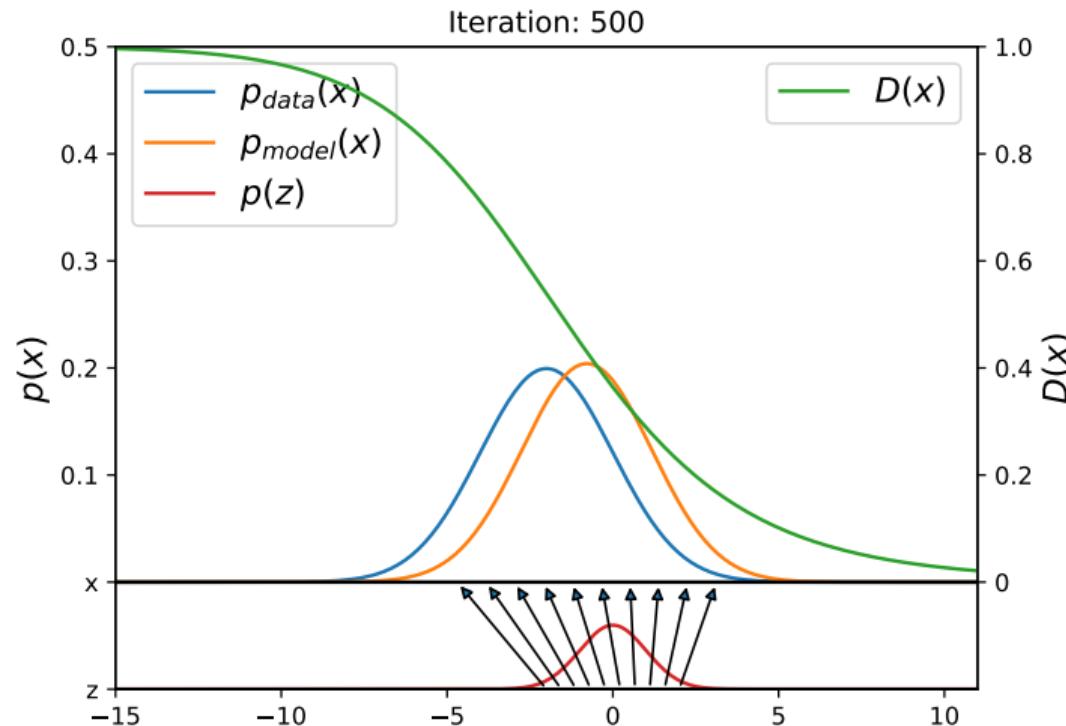
Generative Adversarial Networks

- ▶ **Theoretical analysis** shows that this minimax game recovers $p_{model} = p_{data}$ if G and D are given enough capacity and assuming that D^* can be reached
- ▶ In practice, however, we must use iterative numerical optimization and optimizing D in the inner loop to completion is computationally prohibitive and would lead to overfitting on finite datasets
- ▶ Therefore, we resort to **alternating optimization**:
 - ▶ k steps of optimizing D (typically $k \in \{1, \dots, 5\}$)
 - ▶ 1 step of optimizing G (using a small enough learning rate)
- ▶ This way, we maintain D near its optimal solution as long as G changes slowly

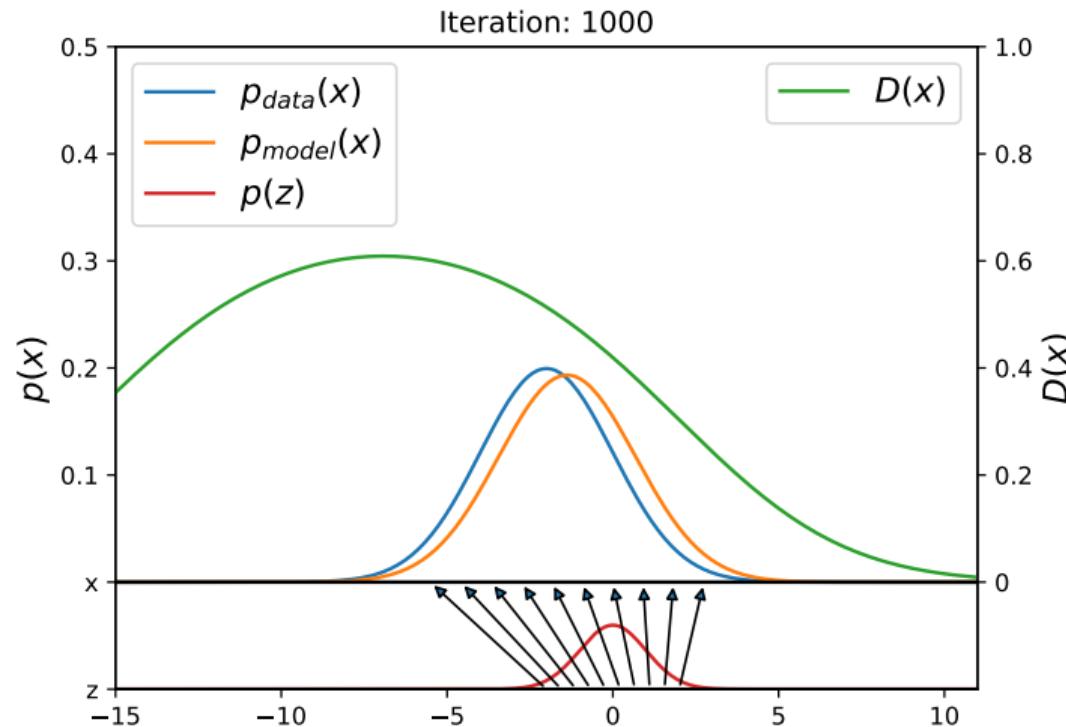
1D Example



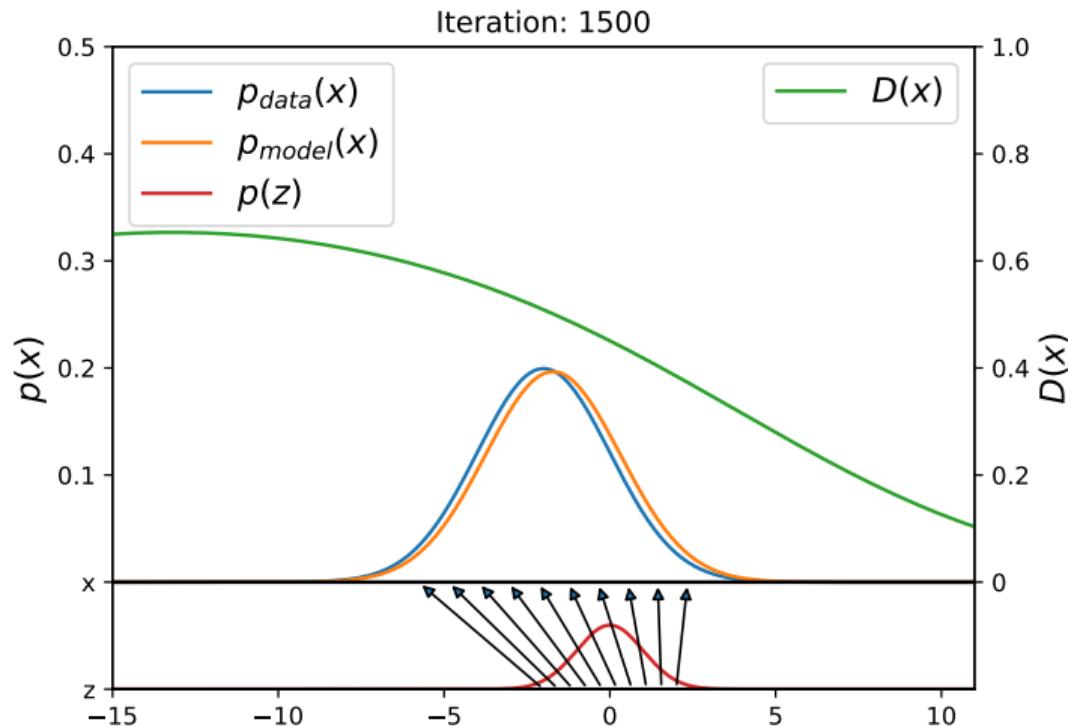
1D Example



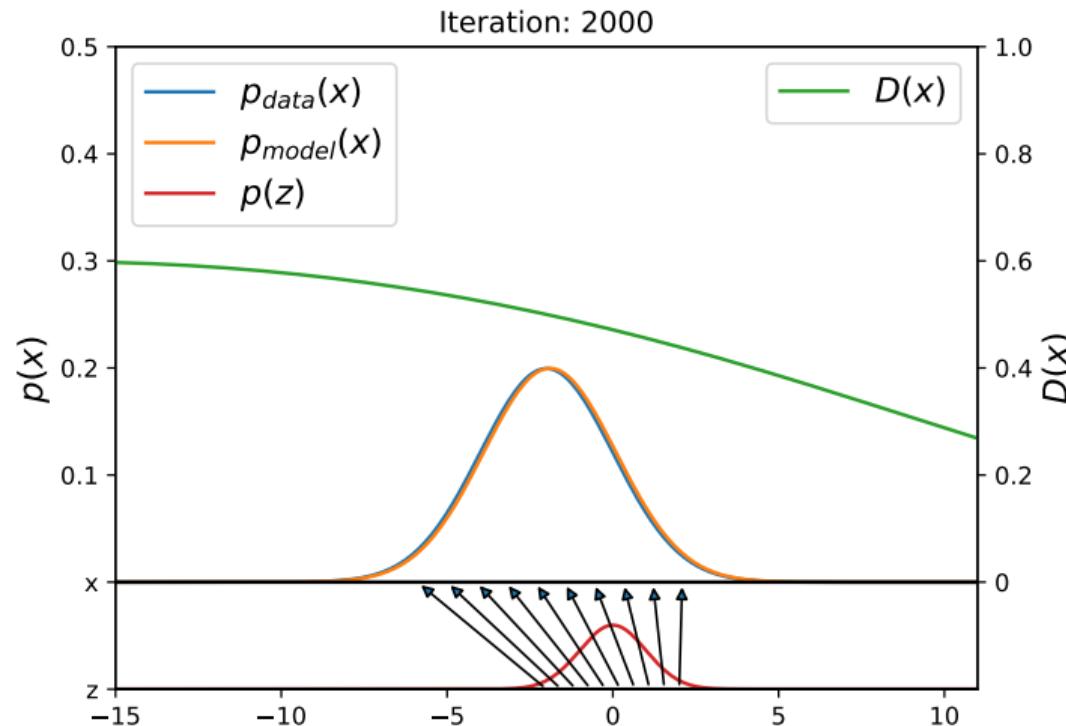
1D Example



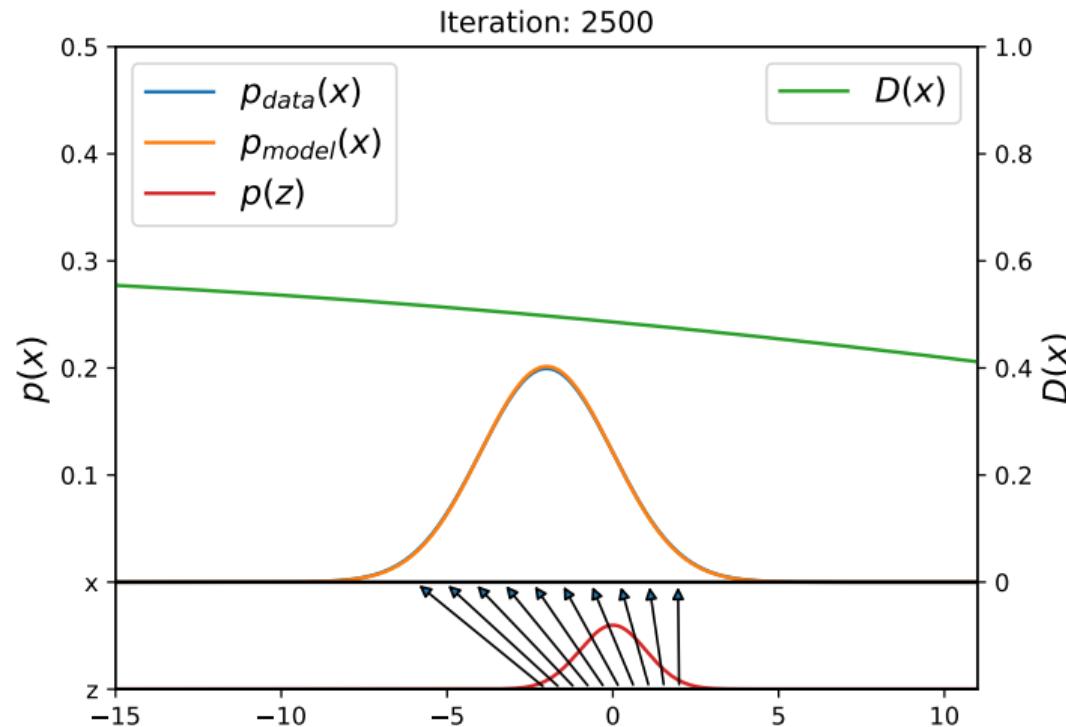
1D Example



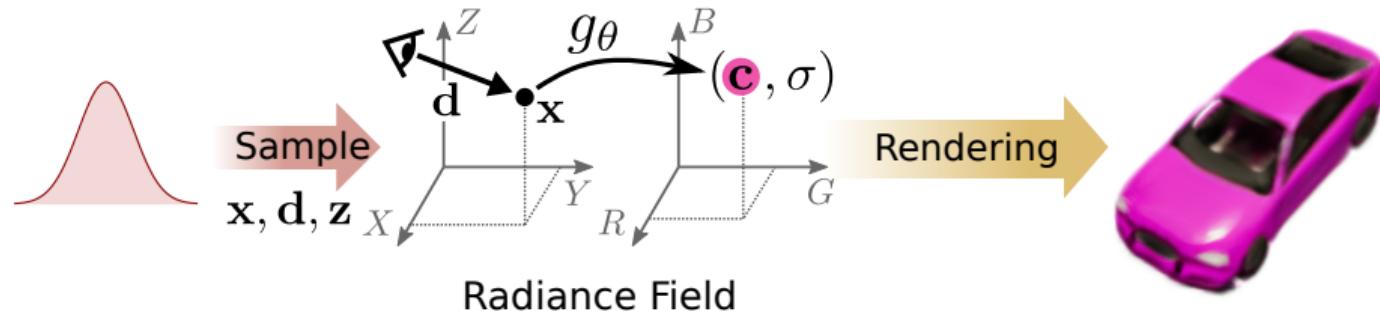
1D Example



1D Example

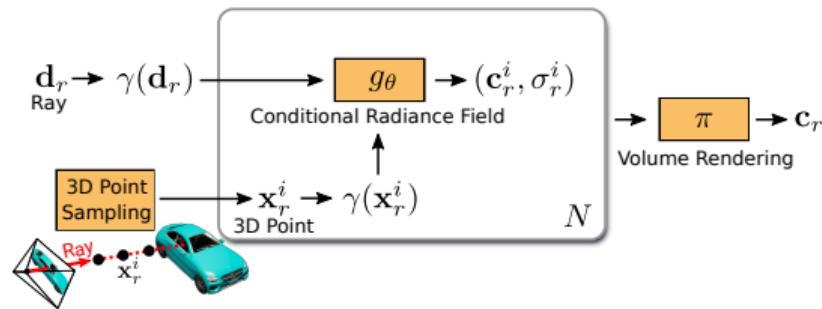


GRAF: Generative Radiance Fields



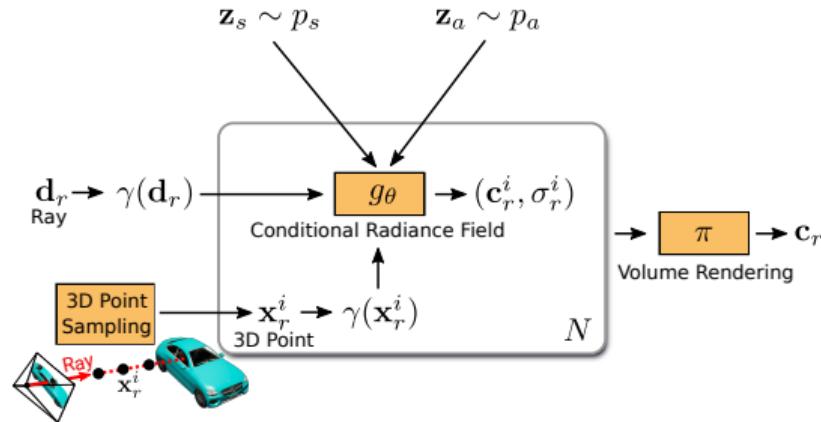
- ▶ **Generative model for radiance fields**
- ▶ Train from **unstructured and unposed 2D image collections**
- ▶ Challenges: Can't discriminate in 3D, but volumetric rendering is slow

GRAF: Generative Radiance Fields



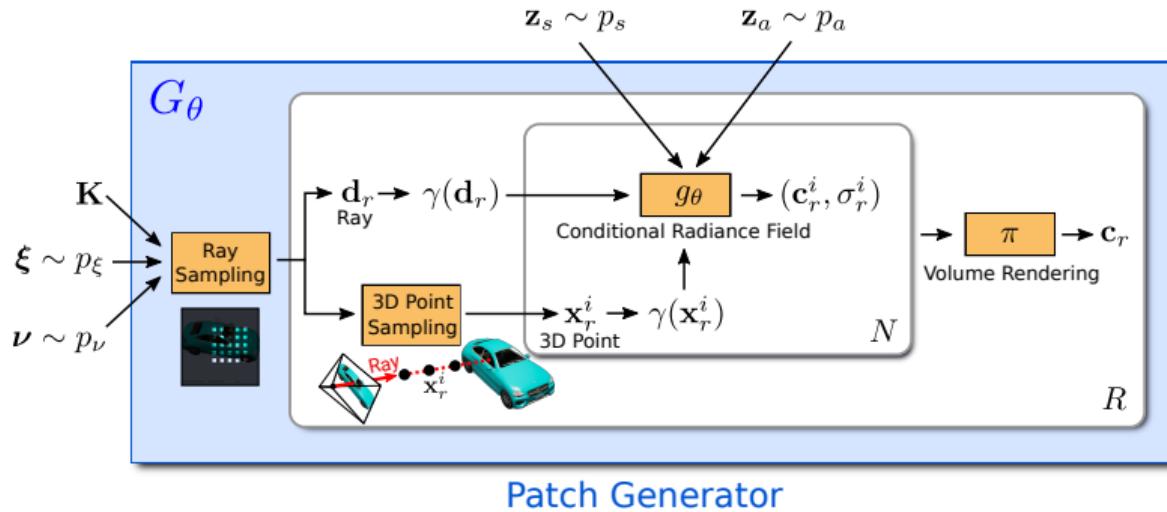
- ▶ A **radiance field** g_θ maps a 3D point \mathbf{x}_r^i and viewing direction \mathbf{d}_r to color/density
- ▶ By sampling N points along the ray, we can **render** a pixel's color \mathbf{c}_r

GRAF: Generative Radiance Fields



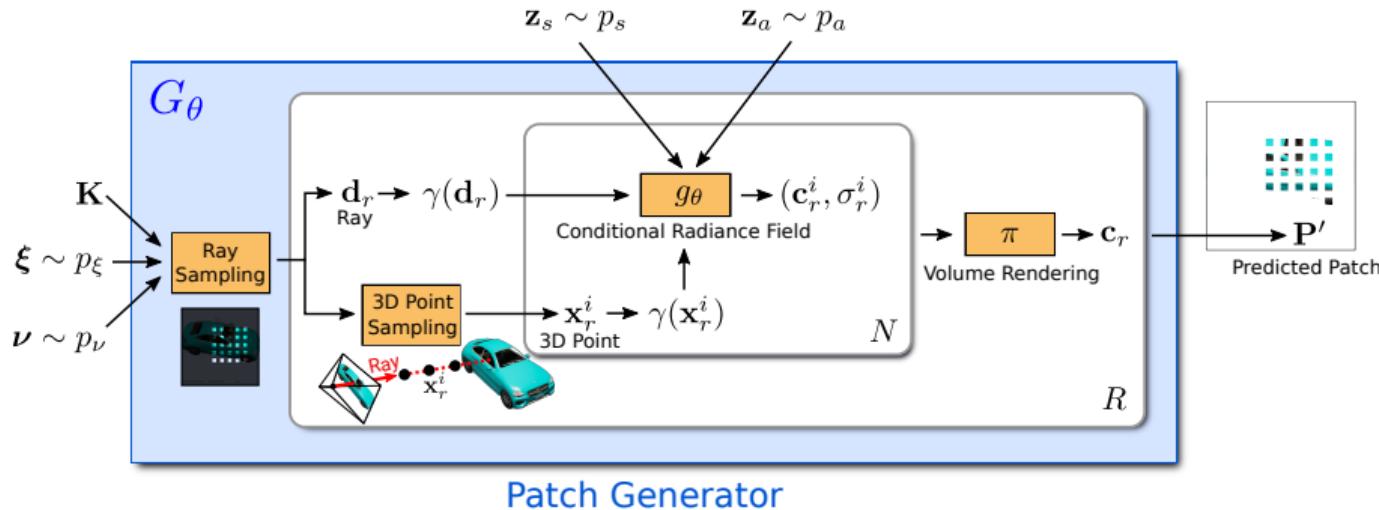
- GRAF **conditions** the radiance field g_θ on additional latent codes \mathbf{z}
- Here, \mathbf{z}_s is a **shape latent code** and \mathbf{z}_a is an **appearance latent code**

GRAF: Generative Radiance Fields



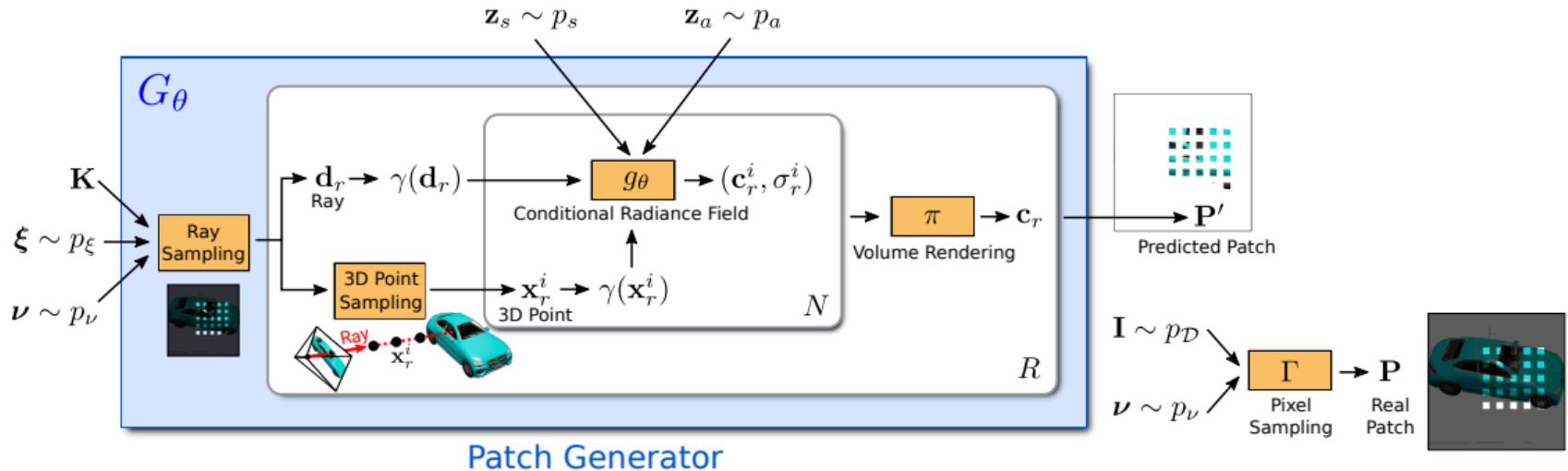
- The generator repeats this process for R rays, sparsely sampled on a 2D grid
- The camera **intrinsics** \mathbf{K} , **extrinsics** ξ and **2D grid** ν are drawn randomly

GRAF: Generative Radiance Fields



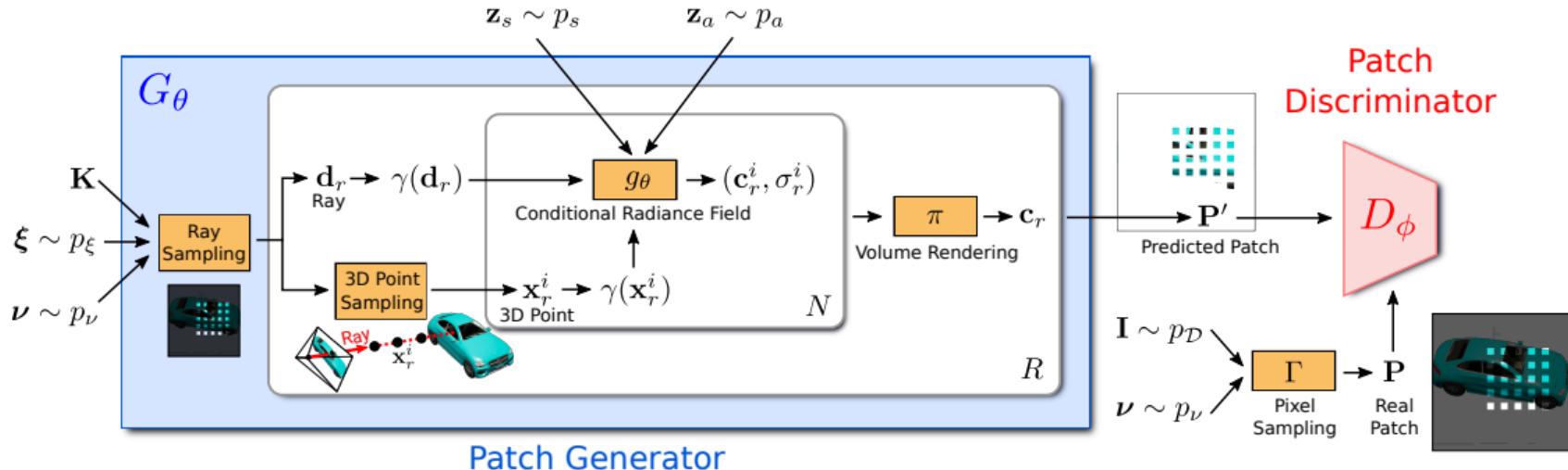
- ▶ This generates **image patches** of size 32×32 pixels
- ▶ The sampling pattern changes the **location** and **stride** (scale)

GRAF: Generative Radiance Fields



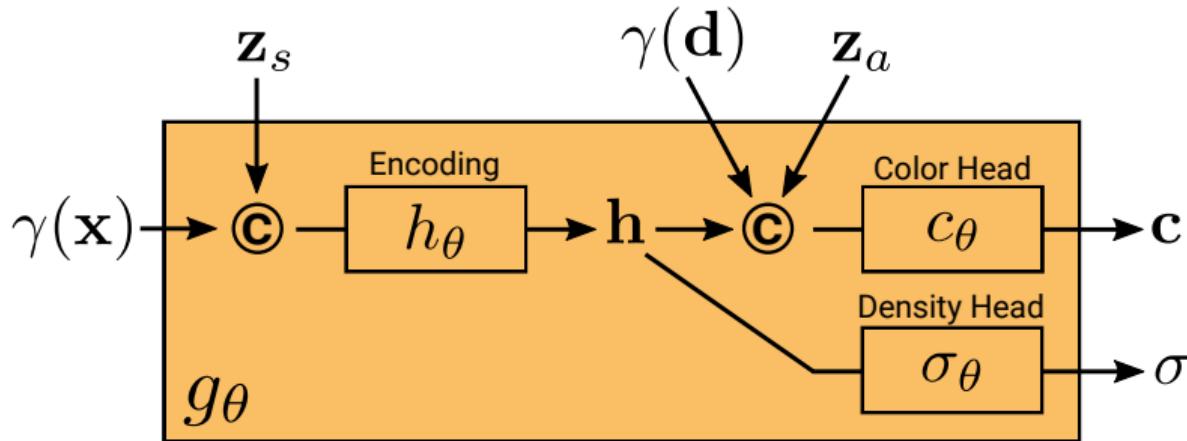
- ▶ Similarly to the generator, we can extract **real patches** of the same size
- ▶ Here \mathbf{I} denotes a real image sampled from the data distribution $p_{\mathcal{D}}$

GRAF: Generative Radiance Fields



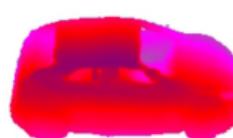
- We can now compare both patches using a **2D discriminator**
- GRAF implements the 2D discriminator as a simple **4 layer ConvNet**

Conditional Radiance Field Network Architecture

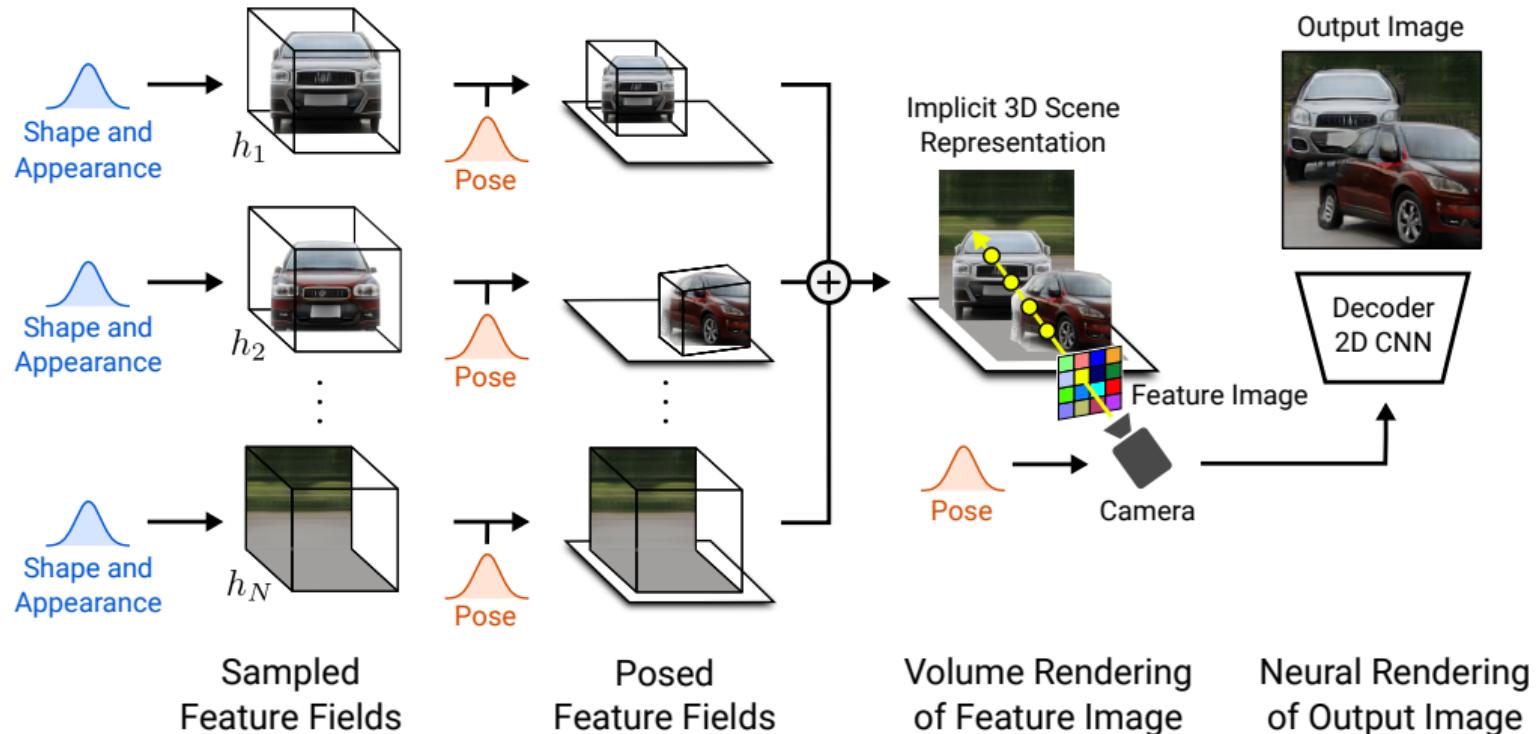


- ▶ The **volume density** σ depends solely on the 3D point \mathbf{x} and the shape code \mathbf{z}_s
- ▶ The **predicted color** \mathbf{c} additionally depends on the viewing direction \mathbf{d} and \mathbf{z}_a
- ▶ This allows to model **view-dependent appearance** (e.g., specularities)

GRAF: Generative Radiance Fields



GIRAFFE: Compositional Generative Neural Feature Fields



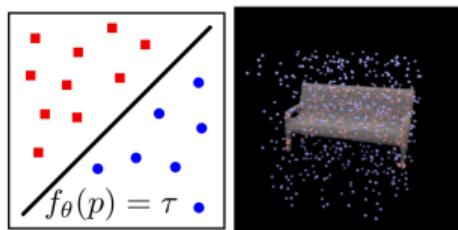
Summary

Neural Networks as Continuous Shape Representations

Occupancy Networks

[Mescheder et al. 2019]

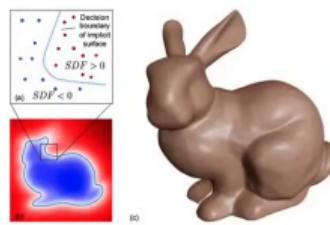
$(x, y, z) \rightarrow$ occupancy



DeepSDF

[Park et al. 2019]

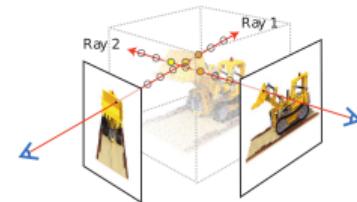
$(x, y, z) \rightarrow$ distance



Neural Radiance Fields

[Mildenhall et al. 2020]

$(x, y, z, \theta, \phi) \rightarrow$ color, density



Scene Representation Networks

[Sitzmann et al. 2019]

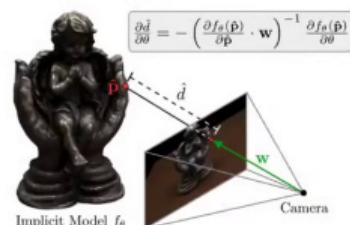
$(x, y, z) \rightarrow$ latent vec. (color, dist)



Differentiable Volumetric Rendering

[Niemeyer et al. 2020]

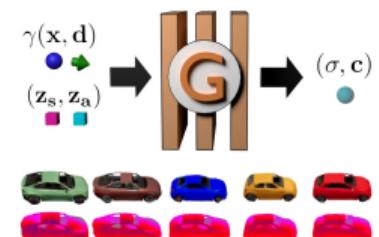
$(x, y, z) \rightarrow$ color, occ.



Generative Radiance Fields

[Schwarz et al. 2020]

$(x, y, z, \theta, \phi, \mathbf{z}) \rightarrow$ color, density



Summary

Coordinate-Based Networks:

- ▶ Effective output representation for shape, appearance, material, motion
- ▶ No discretization, model arbitrary topology
- ▶ Can be learned from images via differentiable rendering
- ▶ Many applications: reconstruction, motion, view synthesis, robotics

However:

- ▶ Geometry must be extracted in post-processing step (1 sec for ONet)
- ▶ Extension to 4D not straightforward (curse of dimensionality)
- ▶ Fully connected architecture and global condition lead to oversmooth results
- ▶ Promising: Local features (ConvONet), Better input encoding (NeRF)