Neural Scene Representation for Reconstruction and Rendering

CS580

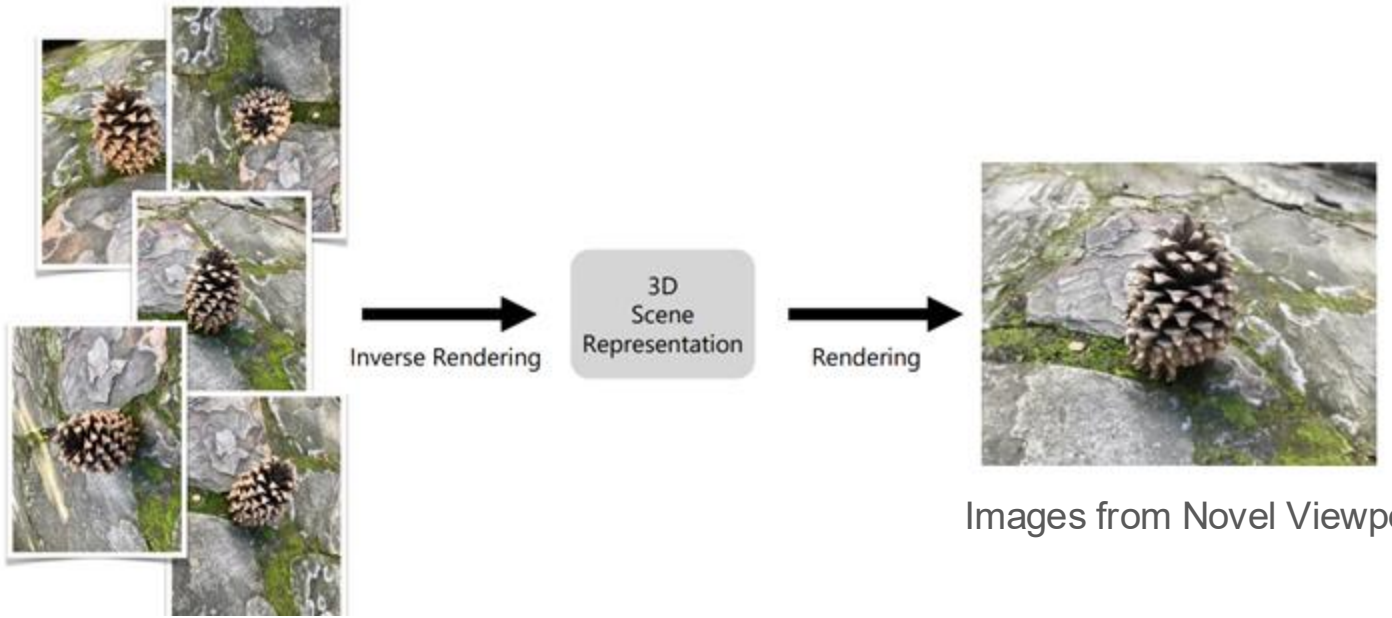
Youngju Na

KAIST SGVR Lab

yjna2907@kaist.ac.kr

Table of Contents

Motivation: Reconstructing the 3D World from images

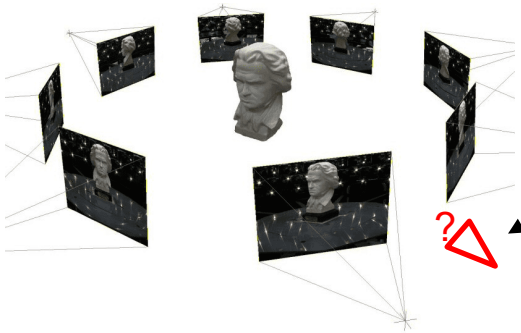


Images from multiple camera viewpoints

Images from Novel Viewpoints

Neural Radiance Fields ECCV 2020 Oral - Best Paper Honorable Mention

Input: images from various camera viewpoints



Output: images from novel camera viewpoints

Source: https://theaisummer.com/nerf/

Examples (synthesized from novel views)

Videos: https://www.youtube.com/watch?v=JuH79E8rdKc&t=191s

1. Implicit Representation

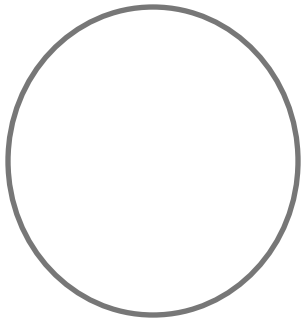
(NeRF)

Implicit Representation

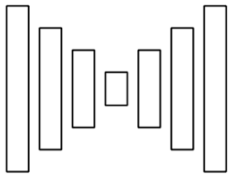
$f(\cdot)$ is a parameterized 2D/3D scalar field

x: coordinate

$f(x) = \|x^2\| - 1$

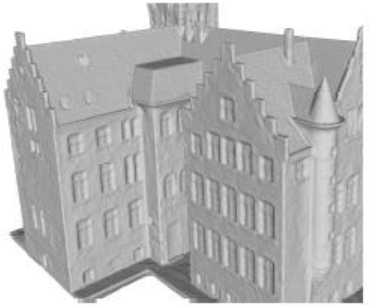


x



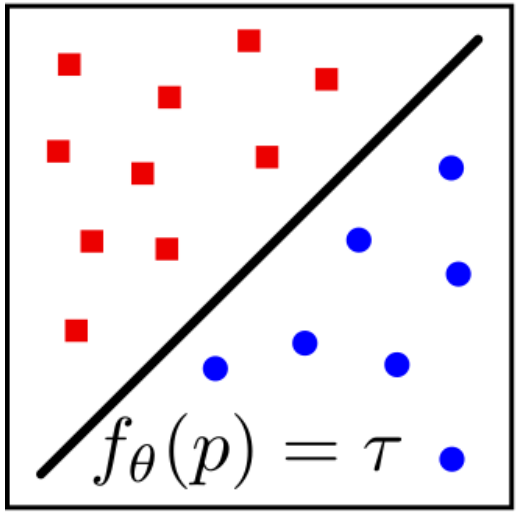
Neural Network

$f(x) = ?$



Represent 3D Scene as Continuous functions

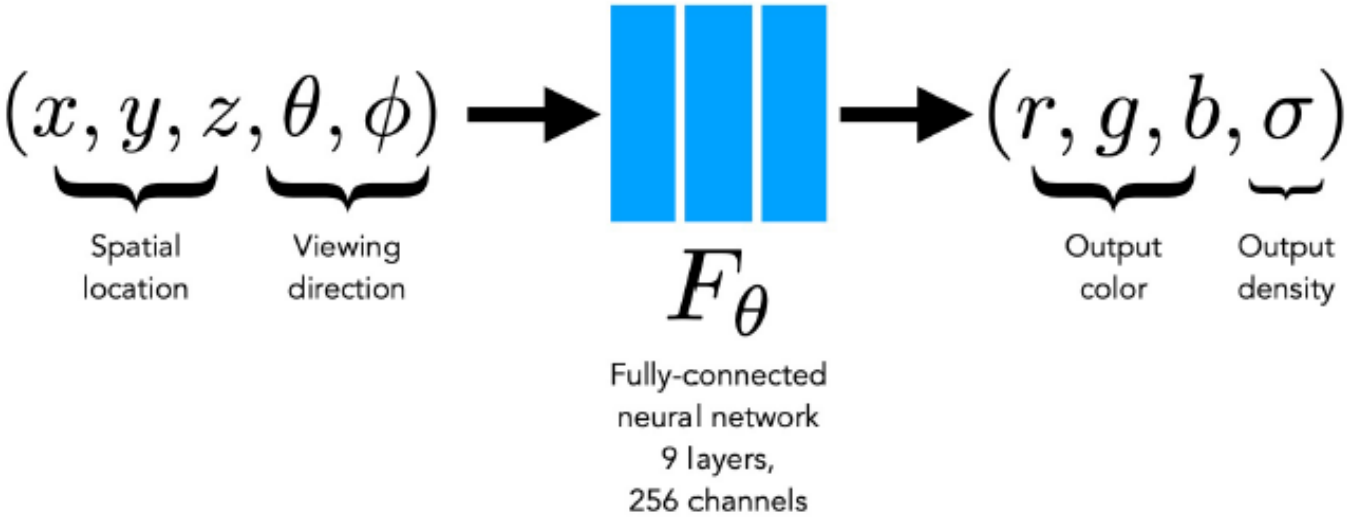
Signed Distance Function (SDF) or Occupancy Fields



Occupancy networks, Mescheder et al.

NeRF 3D Representations

Neural Network (MLPs) as a continuous shape representation.

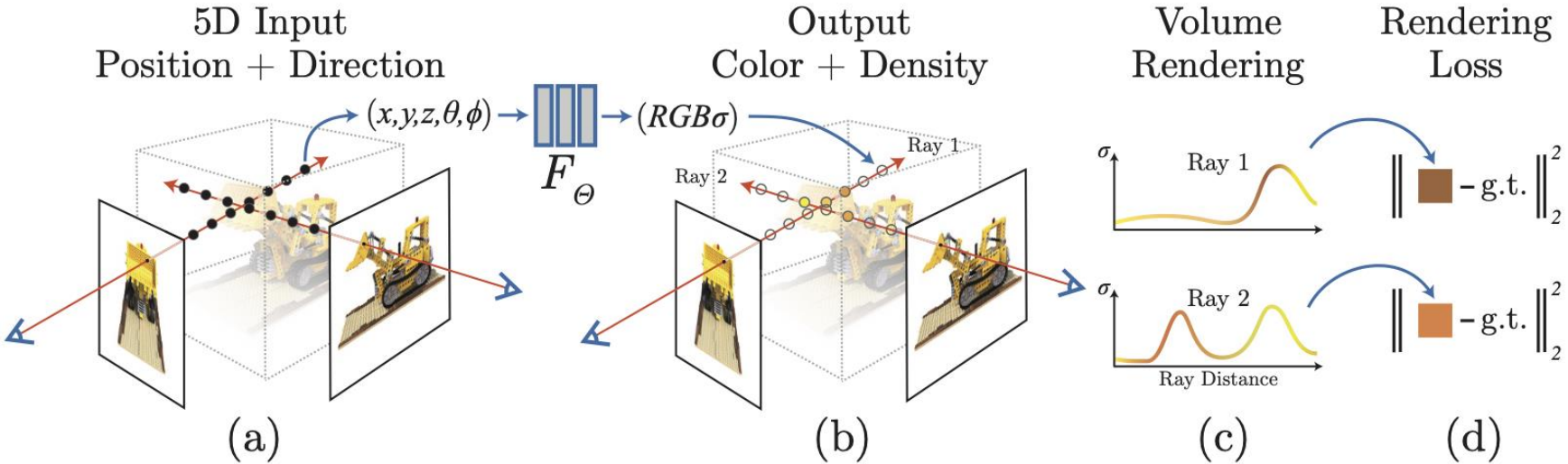


How do we learn 3D representations from 2D images?

https://www.matthewtancik.com/nerf

Method Overview

Cast Rays => Estimate 3D Representations => **Volume Rendering** => 2D Photometric Loss



https://www.matthewtancik.com/nerf

Neural Volumetric Rendering

https://sites.google.com/berkeley.edu/nerf-tutorial/home

Neural Volumetric **Rendering**

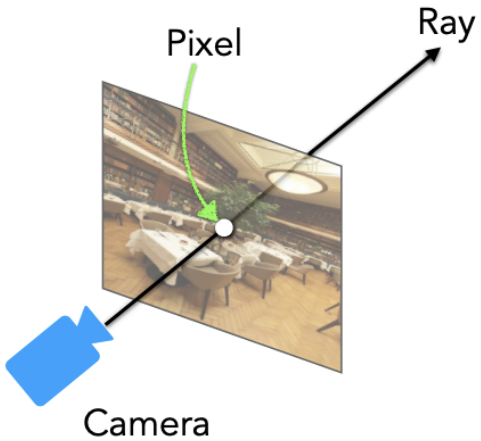
computing color along rays
through 3D space

What color is this pixel?

https://sites.google.com/berkeley.edu/nerf-tutorial/home

Cameras and rays

- We need the mathematical mapping from (*camera*, *pixel*) → *ray*
- Then can abstract underlying problem as learning the function *ray* → *color* (the “plenoptic function”)



https://sites.google.com/berkeley.edu/nerf-tutorial/home

Coordinate frames + Transforms: world-to-camera

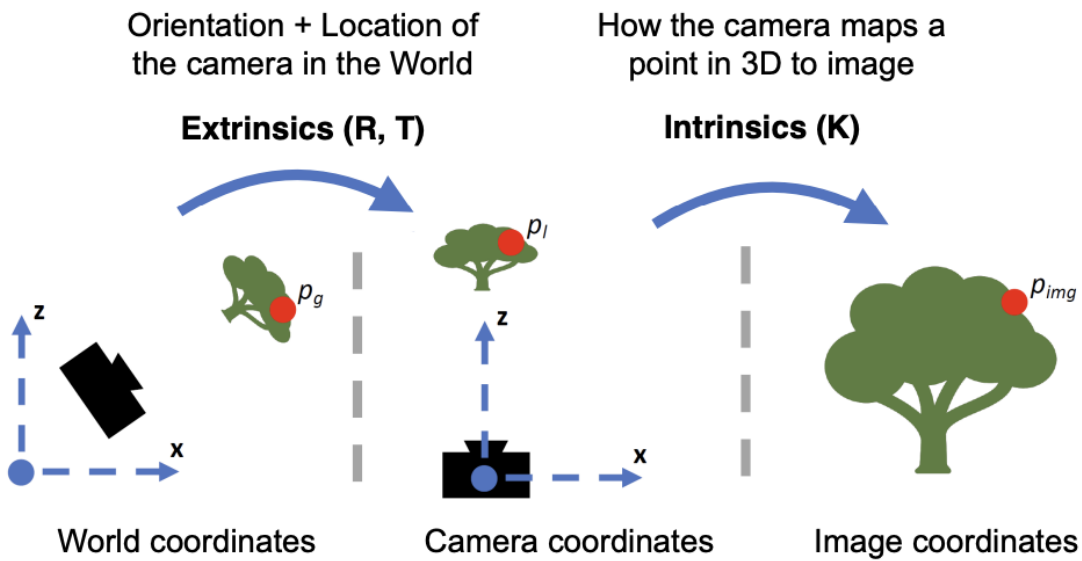
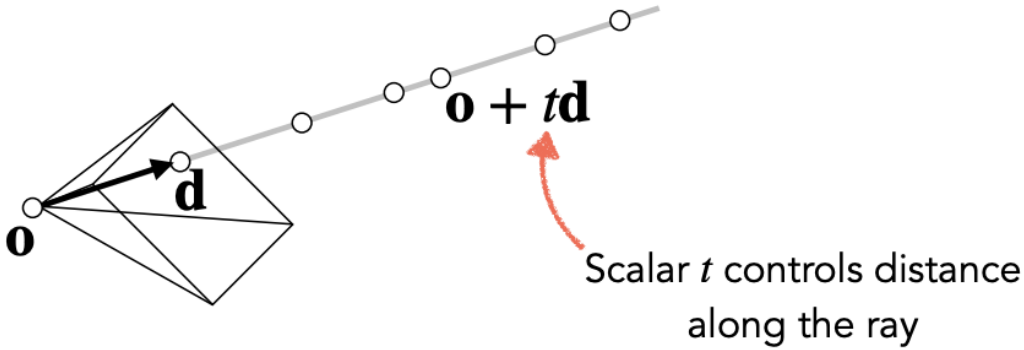


Figure credit: Peter Hedman

Calculating points along a ray



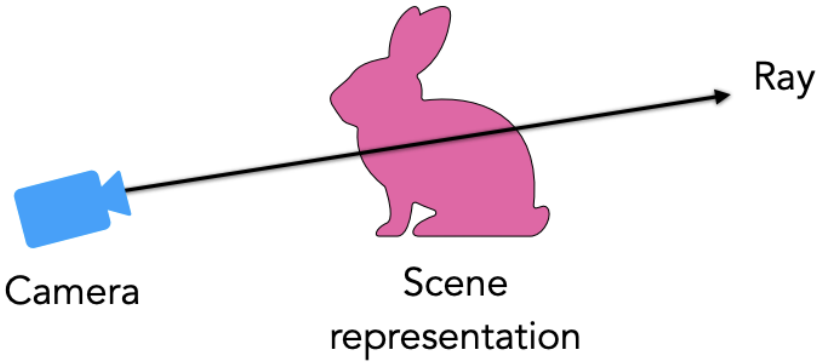
https://sites.google.com/berkeley.edu/nerf-tutorial/home

Neural **Volumetric** Rendering

continuous, differentiable
rendering model without
concrete ray/surface intersections

https://sites.google.com/berkeley.edu/nerf-tutorial/home

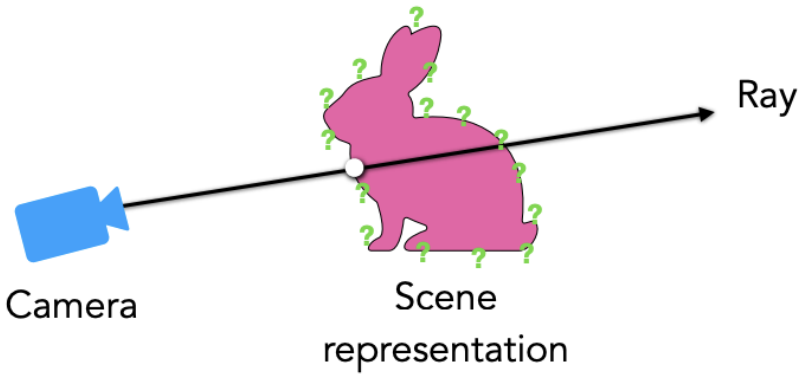
Surface vs. volume rendering



Want to know how ray interacts with scene

https://sites.google.com/berkeley.edu/nerf-tutorial/home

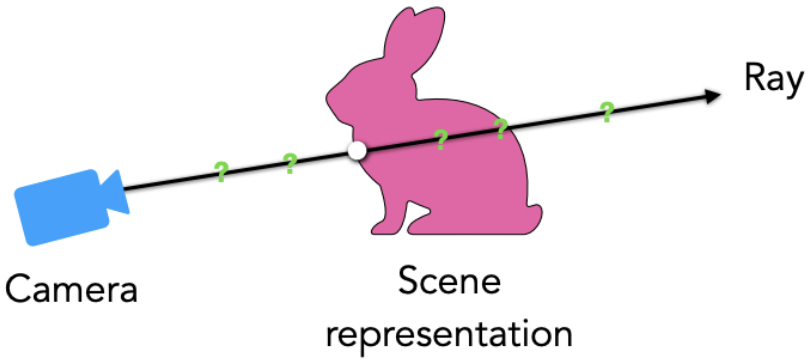
Surface vs. volume rendering



Surface rendering — loop over geometry, check for ray hits

https://sites.google.com/berkeley.edu/nerf-tutorial/home

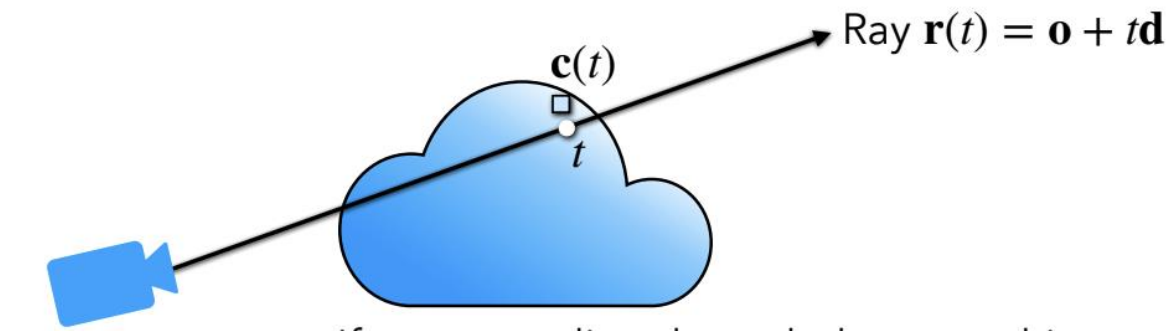
Surface vs. volume rendering



Volume rendering — loop over ray points, query geometry

https://sites.google.com/berkeley.edu/nerf-tutorial/home

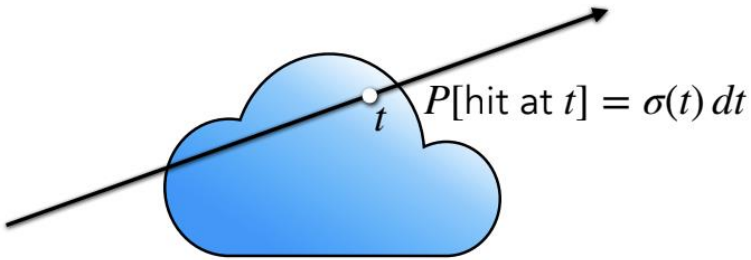
Volumetric formulation for NeRF



If a ray traveling through the scene hits a particle at distance t along the ray, we return its color $\mathbf{c}(t)$

https://sites.google.com/berkeley.edu/nerf-tutorial/home

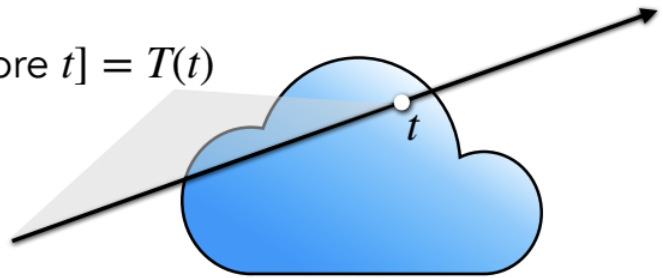
What does it mean for a ray to "hit" the volume?



This notion is *probabilistic:* chance that ray hits a particle in a small interval around t is $\sigma(t)\,dt$.
σ is called the "volume density"

Probabilistic interpretation

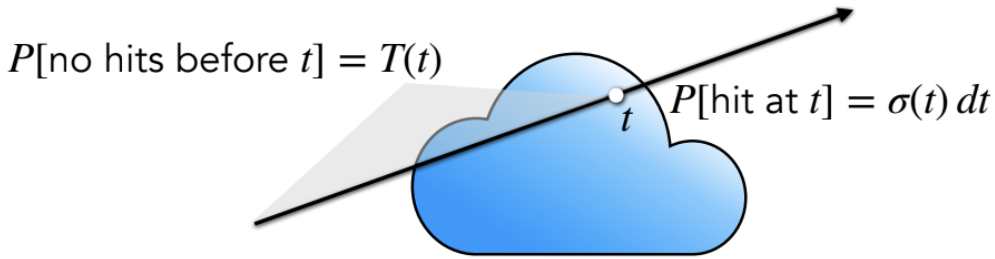
$P[\text{no hits before } t] = T(t)$



To determine if t is the *first* hit along the ray, need to know $T(t)$: the probability that the ray makes it through the volume up to t.
$T(t)$ is called "transmittance"

https://sites.google.com/berkeley.edu/nerf-tutorial/home

PDF for ray termination



Finally, we can write the probability that a ray terminates at t as a function of only sigma

$$
\begin{aligned}
P[\text{first hit at } t] &= P[\text{no hit before } t] \times P[\text{hit at } t] \\
&= T(t)\sigma(t)dt \\
&= \exp\left(-\int_{t_0}^{t} \sigma(s)\,ds\right)\sigma(t)\,dt
\end{aligned}
$$

https://sites.google.com/berkeley.edu/nerf-tutorial/home

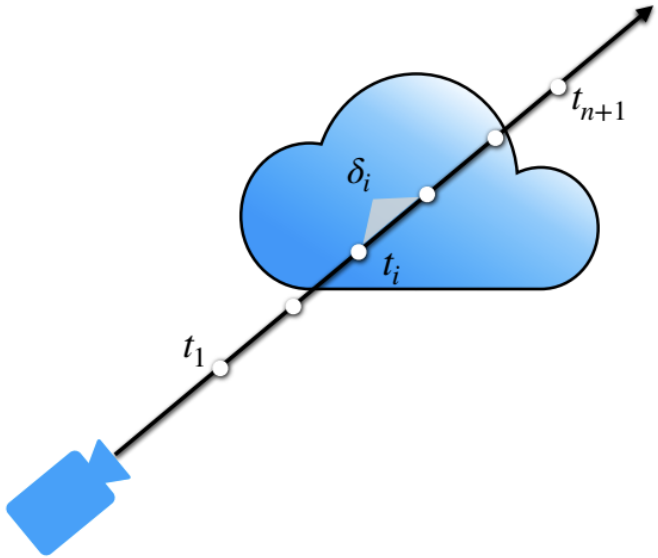
Expected value of color along ray

This means the expected color returned by the ray will be

$$\int_{t_0}^{t_1} \underbrace{T(t)\sigma(t)}_{\text{weighted}} \underbrace{\mathbf{c}(t)}_{\text{color}}\, dt$$

Note the nested integral!

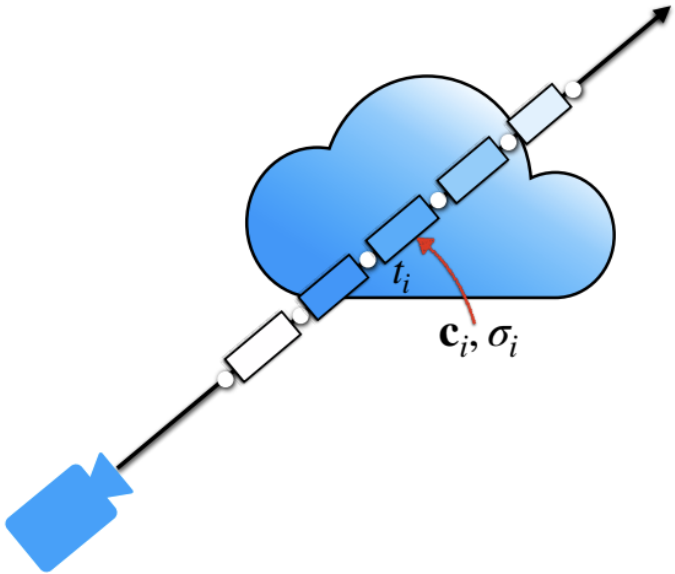
Approximating the nested integral



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

https://sites.google.com/berkeley.edu/nerf-tutorial/home

Approximating the nested integral



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

Summary: volume rendering integral estimate

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

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

colors

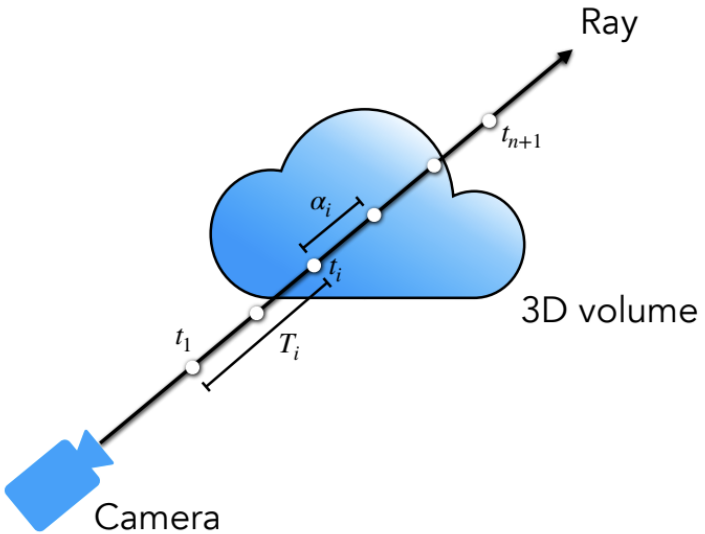
weights

How much light is blocked earlier along ray:

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

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

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



Detailed derivation

Volume rendering is trivially differentiable

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

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

colors

weights

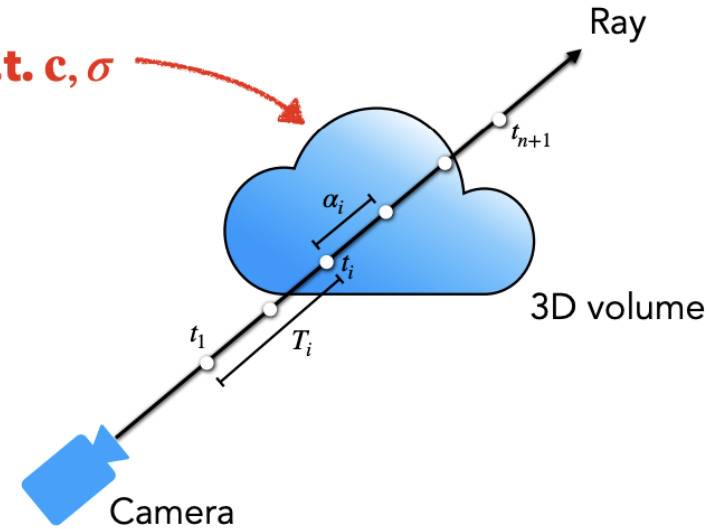
differentiable w.r.t. $\mathbf{c}, \sigma$

How much light is blocked earlier along ray:

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

How much light is contributed by ray segment i:

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



Novel View Synthesis & View Dependency

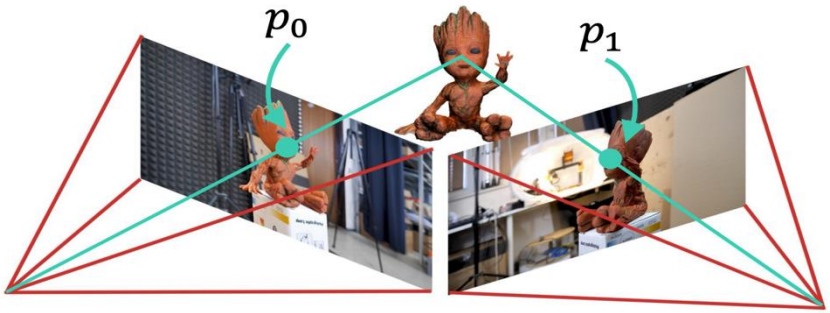
NeRF is slow both in training and rendering.

https://www.matthewtancik.com/nerf

Explicit Representation

- Estimate Geometry with Multi-View Stereo
- Fixing errors in a triangle mesh (*very challenging*) or point clouds

Mesh-based representation



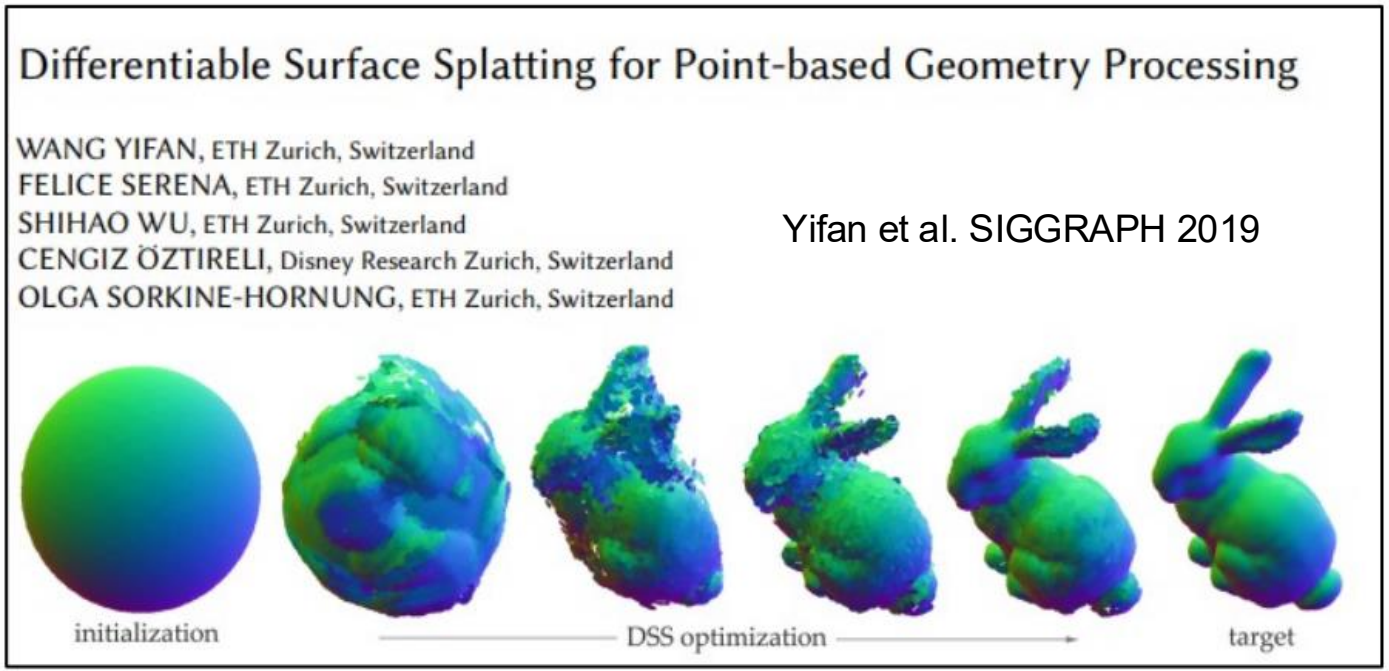
Point-based representation

2. Explicit Representation

(3D Gaussian Splatting)

Point-based Rendering (PBR)

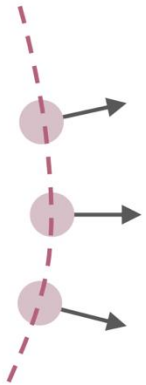
Surface Splatting – Zwicker et al. 2001



https://3dgstutorial.github.io/

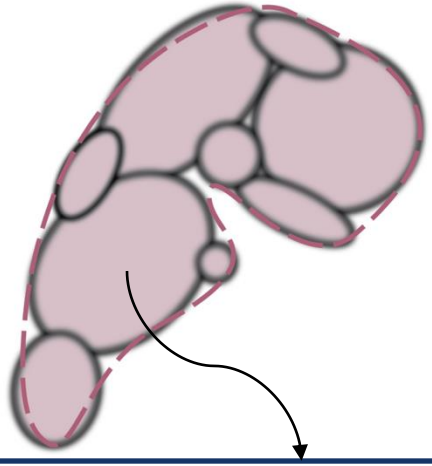
Point-based Rendering (PBR)

Surface Splatting vs Volume Splatting (3DGS)



position normal

$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \quad n = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \quad \sigma \text{ std dev} \quad f \text{ appearance}$$



$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \quad \Sigma = \begin{bmatrix} \sigma_x & \sigma_{xy} & \sigma_{xz} \\ & \sigma_y & \sigma_{yz} \\ & & \sigma_z \end{bmatrix} \quad o \quad SH \text{ spherical harmonics}$$

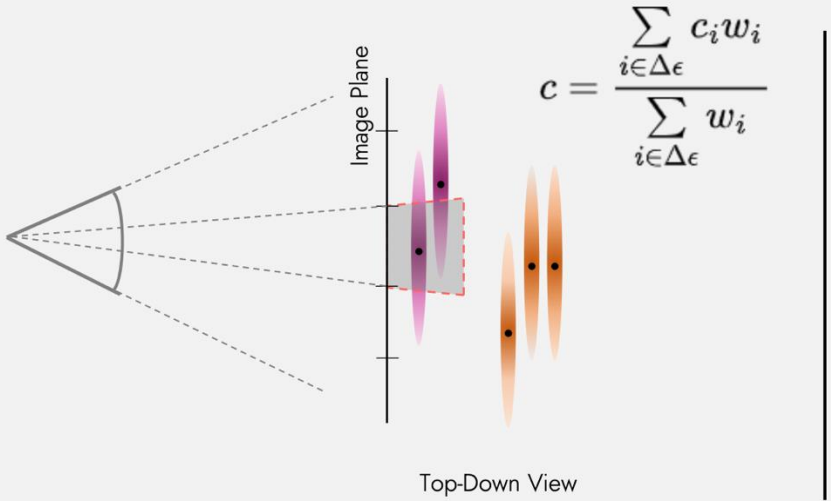
covariance matrix

https://3dgstutorial.github.io/

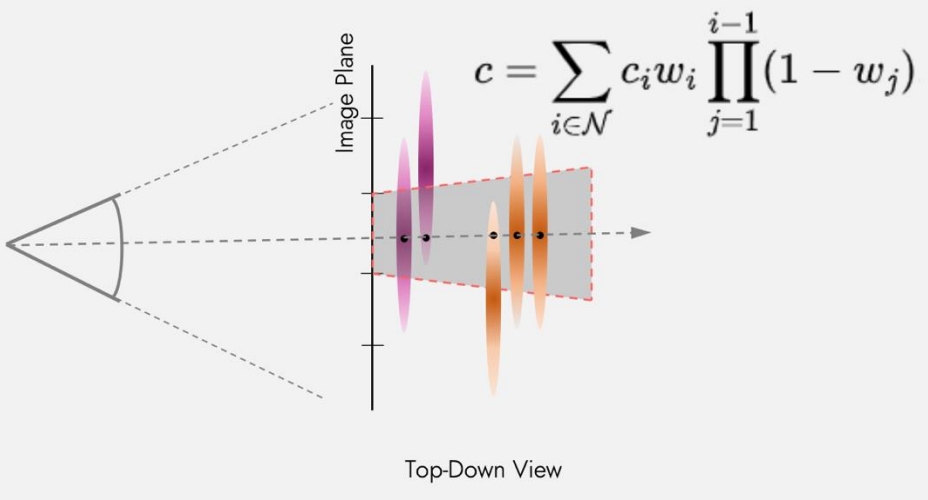
Point-based Rendering (PBR)

Surface Splatting vs Volume Splatting (3DGS)

[Zwicker1 '01] / [Yifan '19]



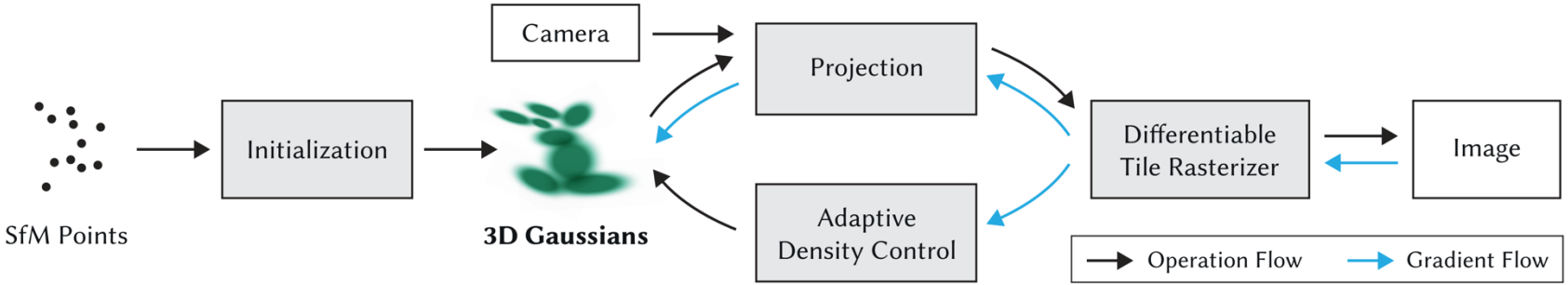
[Zwicker2 '01] / [Kerbl & Kopanas '23]



https://3dgstutorial.github.io/

3D Gaussian Splatting Optimization

3D Gaussian Splatting (3DGS) efficiently reconstructs 3D scenes from posed images, enabling real-time, high-fidelity rendering.



[Kerbl et al, SIGGRAPH 2023]

Anisotropic Volumetric 3D Gaussians

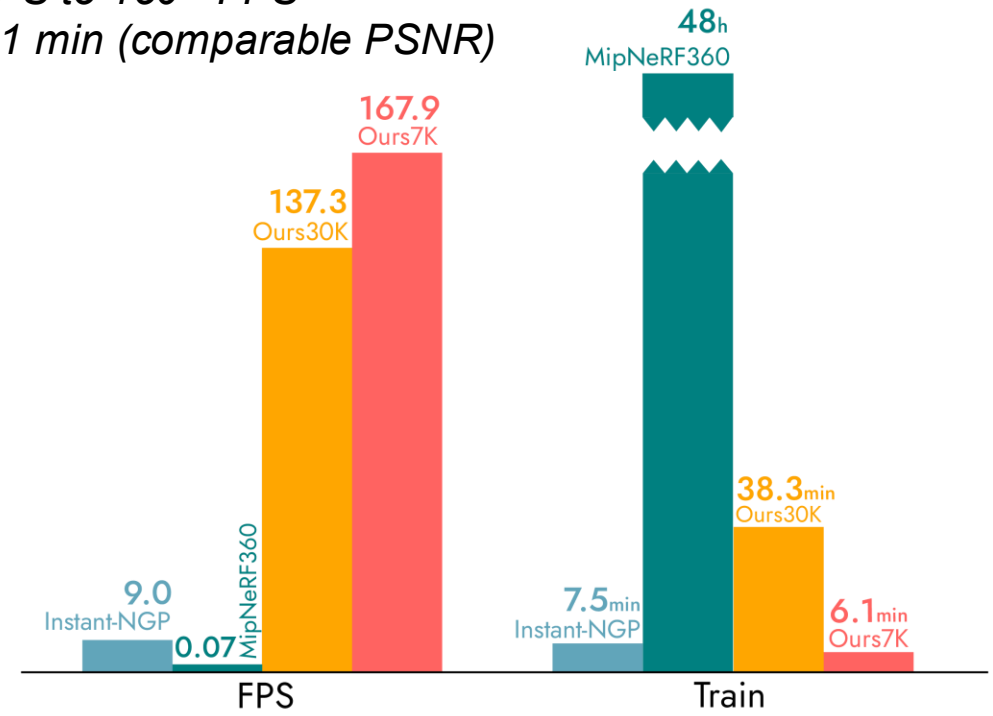
Final Rendering

3D Gaussian Visualization


Timelapse of the Optimization
(NeRF-Synthetic Dataset)

3D Gaussian Splatting Optimization

Rendering: *0.07 FPS to 150+ FPS*
Training: *48h -> 6.1 min (comparable PSNR)*



[Kerbl et al, SIGGRAPH 2023]

What makes this so efficient?

1. Tiling
The image is divided into **small 16×16 pixel tiles**.
Each tile can be processed **independently and in parallel by the GPU**
(perfect for CUDA thread blocks).

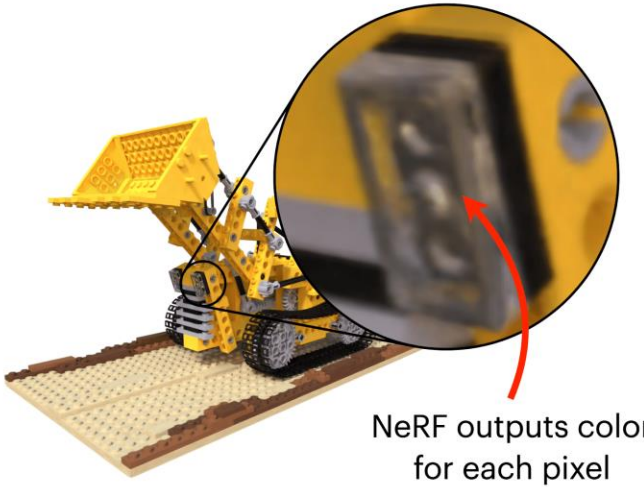
2. Single global sort
All Gaussians are **depth-sorted once globally per frame**, not per pixel.
Sorting is done efficiently on GPU (e.g., radix sort or CUDA Thrust).

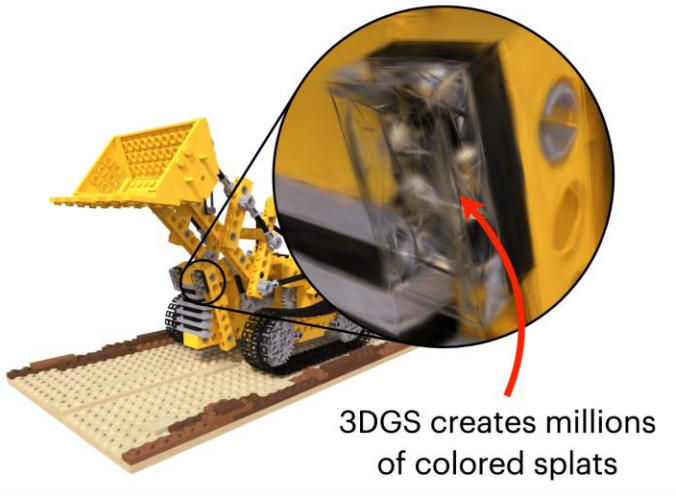
3. On-the-Fly Blending
The rendering is performed by **forward rasterization** — each Gaussian directly writes its contribution to a small number of pixels (within its projected ellipse).

[Kerbl et al, SIGGRAPH 2023]

Different representation, different artifacts

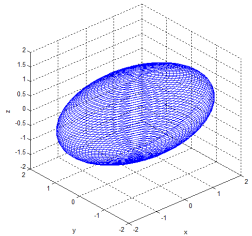
NeRF is implicit representation, while 3DGS is explicit.





Inductive bias (underfitting)

Blurry images



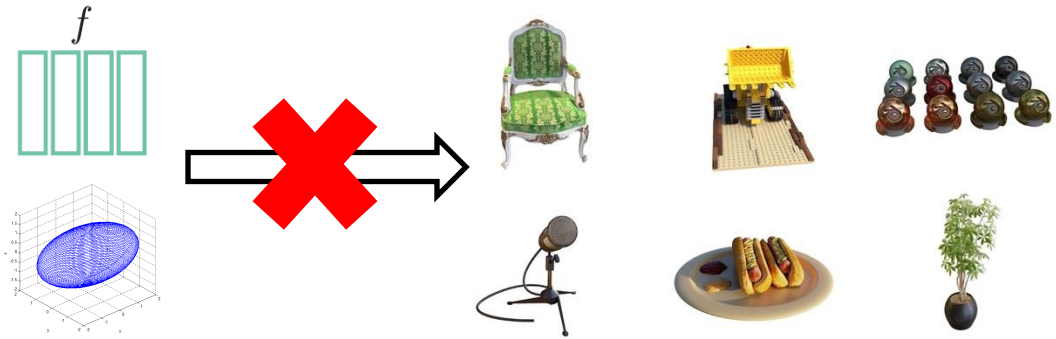
Overfitting to sparse points (noise sensitive)

Floating artifacts

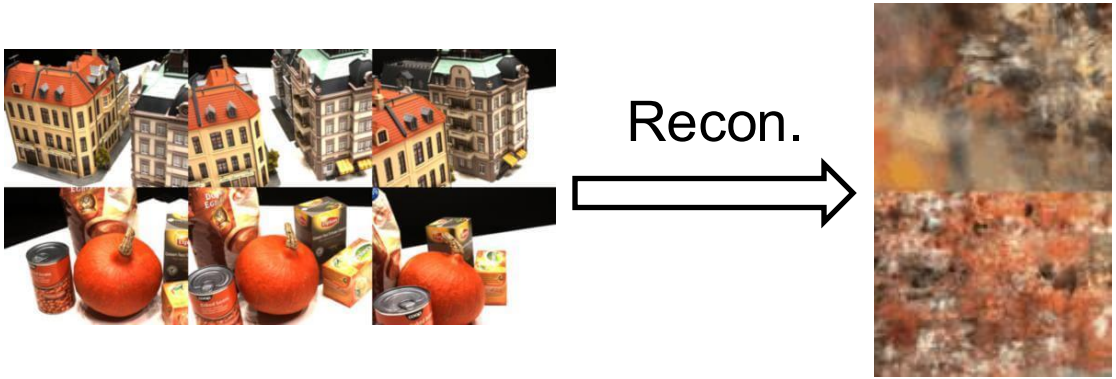
3. Generalizable (≈ amortized, feed-forward) methods

Towards Generalizable Methods

1. Scene-specific representation



2. Sparse input camera viewpoints

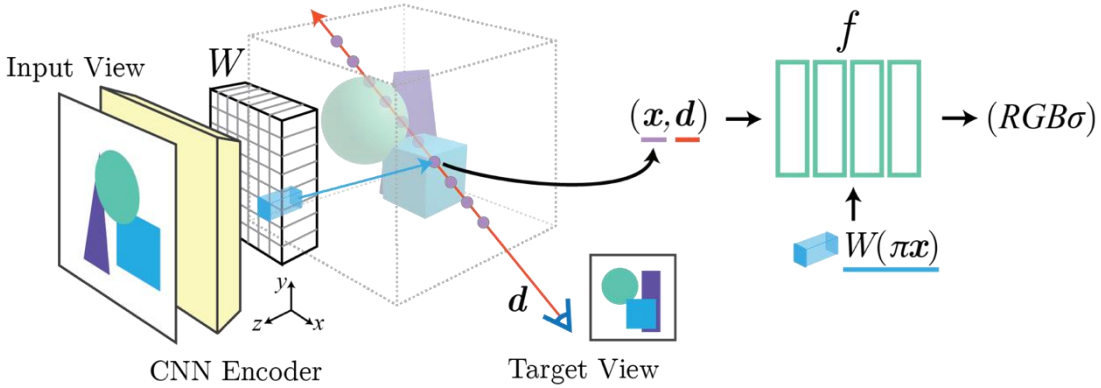


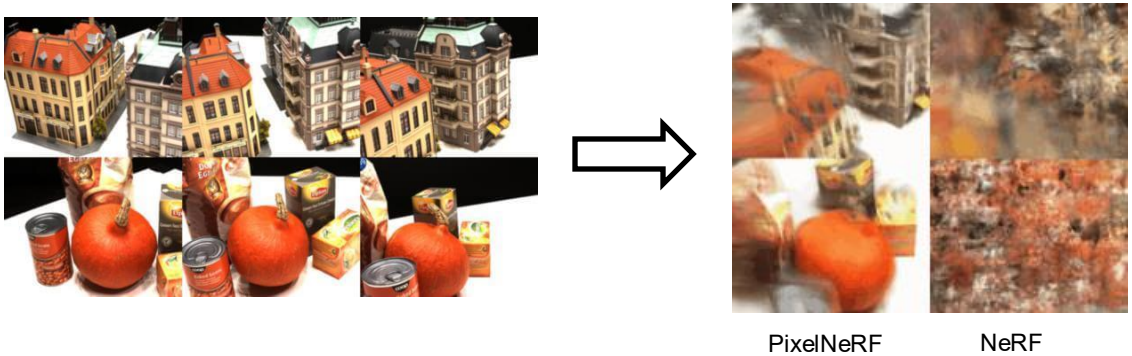
Not Generalizable

Cannot share representations across scenes or views

Feed-Forward NeRF (PixelNeRF, CVPR 2021)

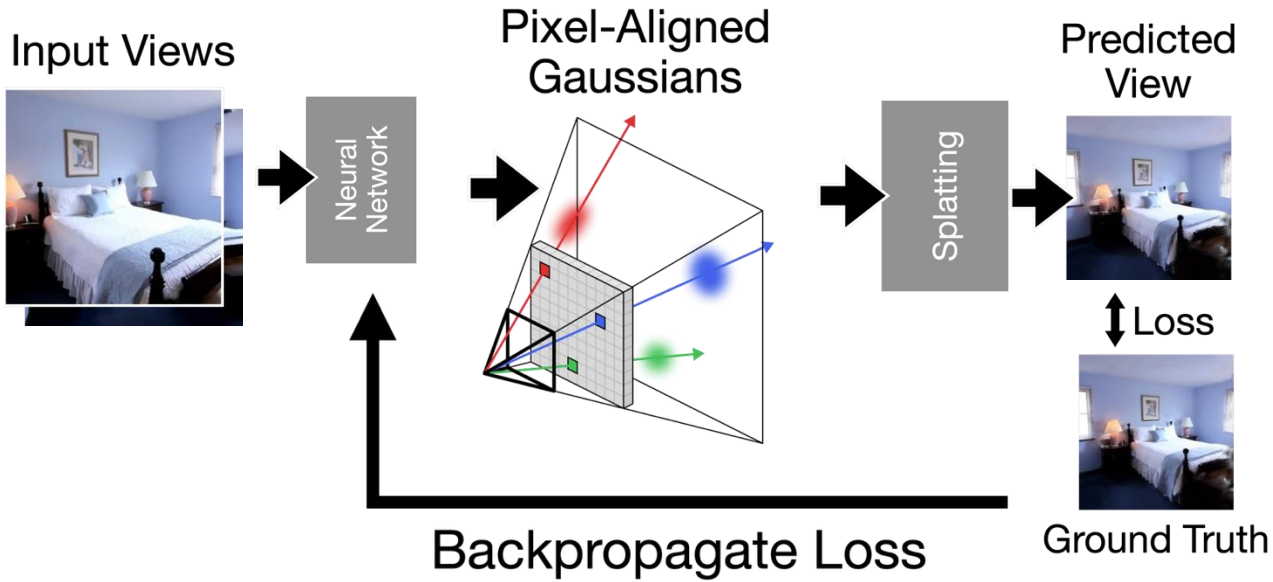
One-Shot NeRF (pixelNeRF [Yu et al. CVPR'21])





Feed-Forward 3DGS (PixelSplat, CVPR2024 best paper honorable mention)

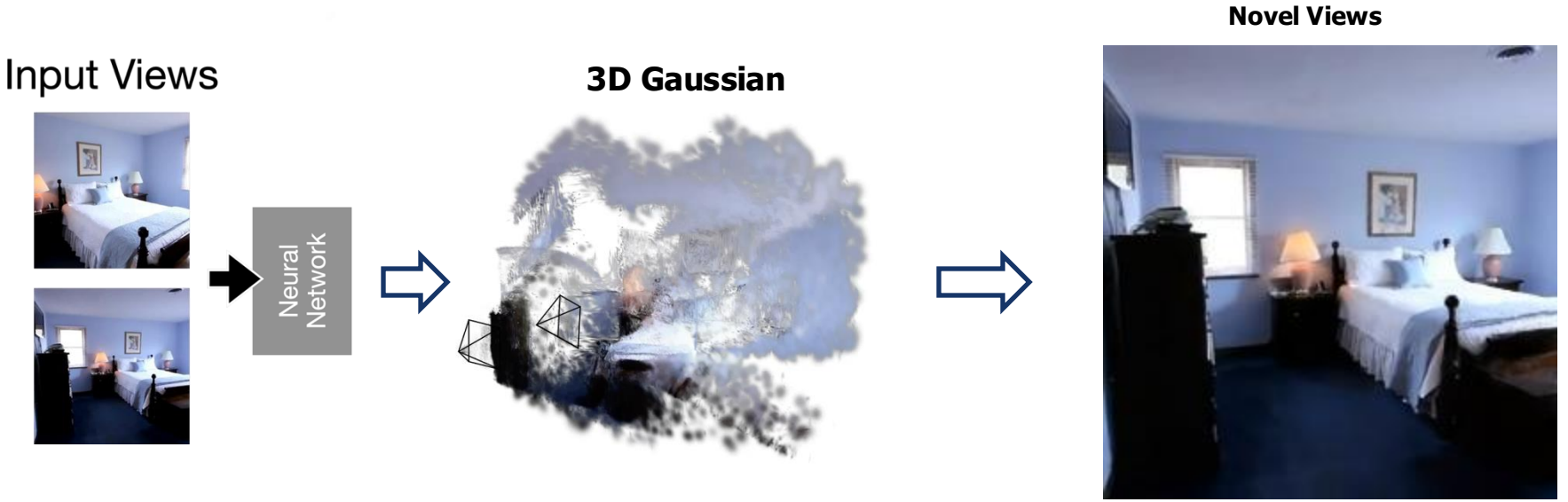
- How does it work?



Slide credit: Charatan et al, Pixelsplat, CVPR 2024

Feed-Forward 3DGS (PixelSplat, CVPR2024 best paper honorable mention)

- Note 1. No Per Scene Optimization ❌, Generalizable ✅
- Note 2. No Dense Views ❌, Only 2-3 images ✅



Slide credit: Charatan et al, Pixelsplat, CVPR 2024

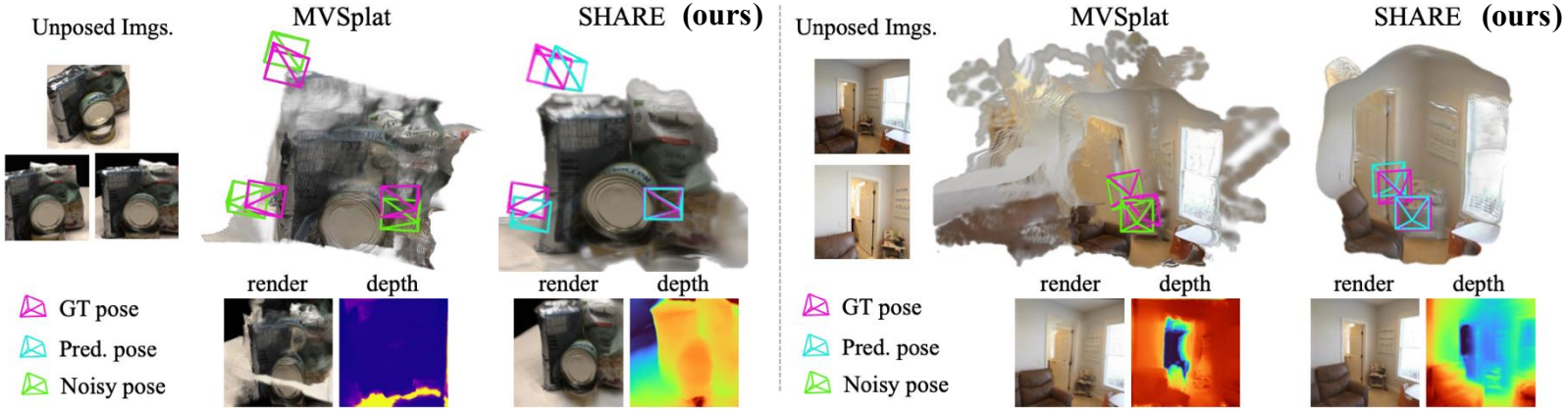
Pose-Free approaches

Previous methods…

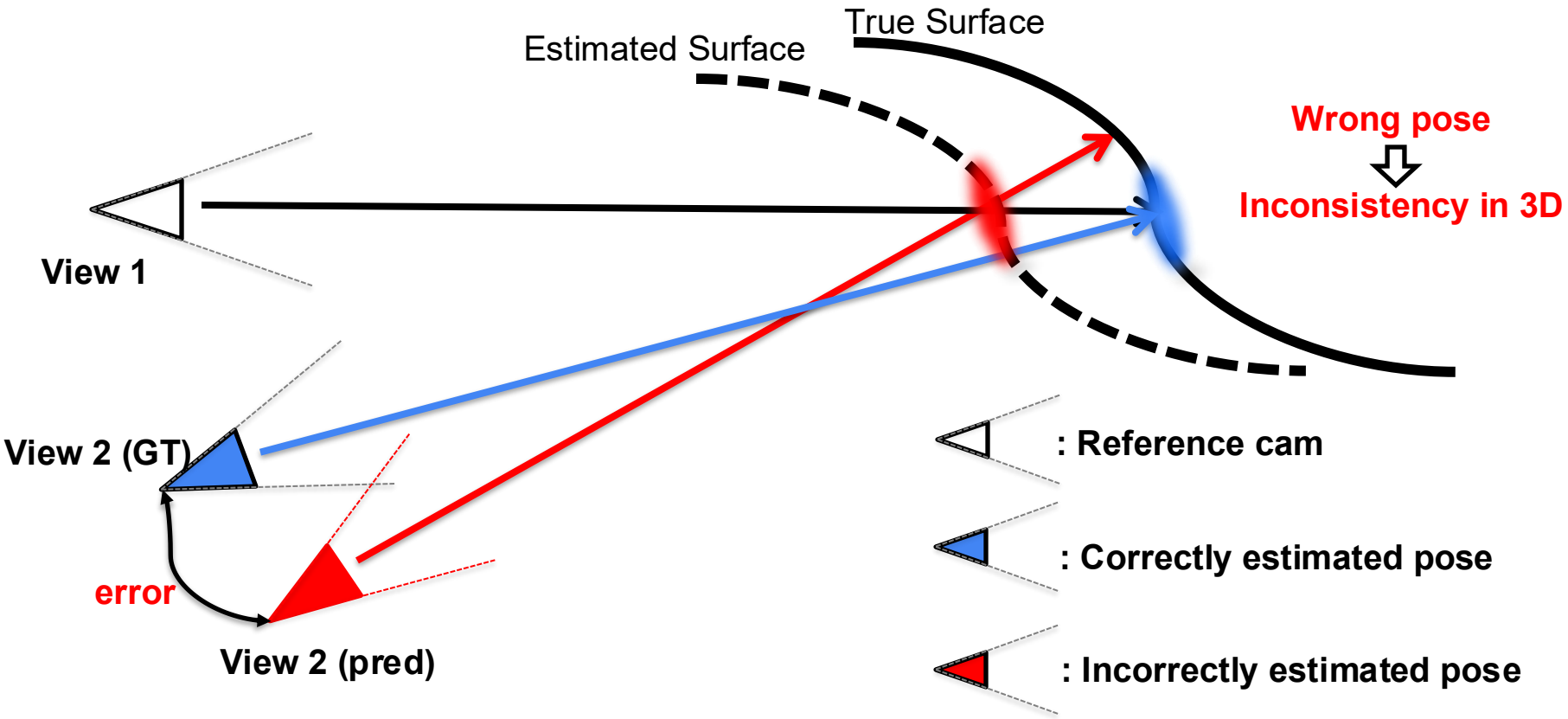
- **Assume accurate (*perfect*) camera pose**
  - Always assume to have relative camera rotation and translation
  - Not practical.

Very sensitive to pose error

- easily leads to geometric misalignment.



Utilizing “Predicted Camera Poses” in Multi-View Scenario



POSE-FREE 3D GAUSSIAN SPLATTING VIA SHAPE-RAY ESTIMATION

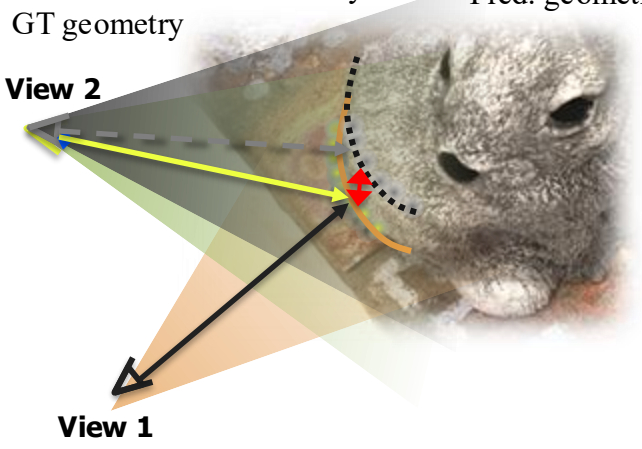
Youngju Na[*], *Taeyeon Kim*[*], *Jumin Lee, Kyu Beom Han, Woo Jae Kim, Sung-Eui Yoon*[†]

School of Computing, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea

ICIP 2025 Best Student Paper Award

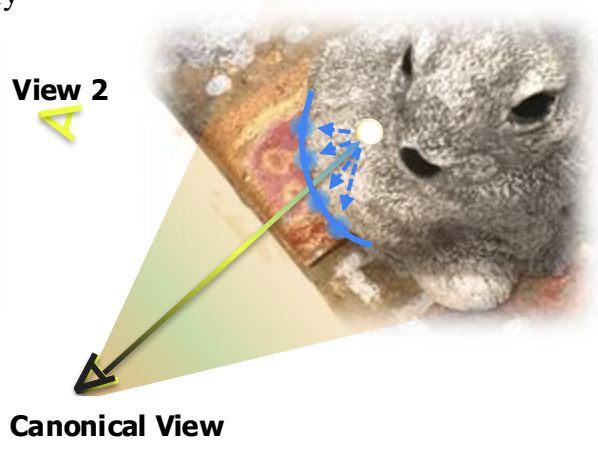
Previous Approach ("estimate → fuse")

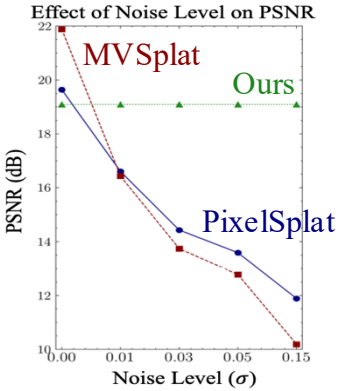
Source view 1
Source view 2
GT geometry
Camera error
Geometry error
Pred. view 2
Pred. geometry



SHARE ("fuse → estimate")

Canonical view
Fused feature
Geometry prediction
Pred. geometry

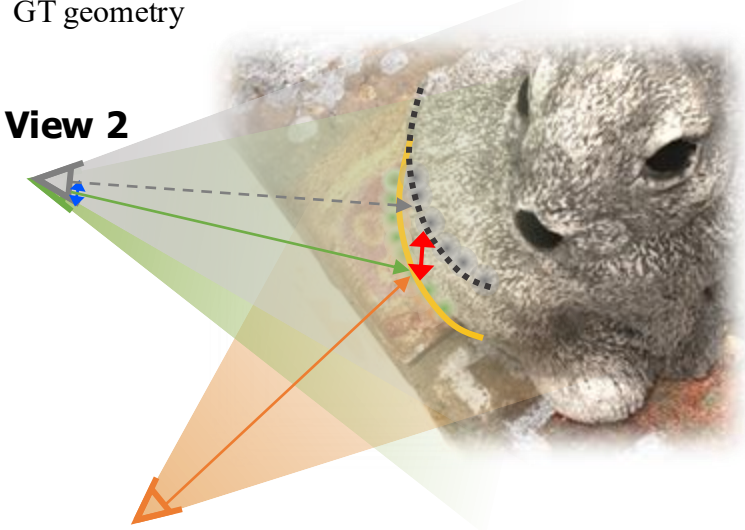




Canonical Space

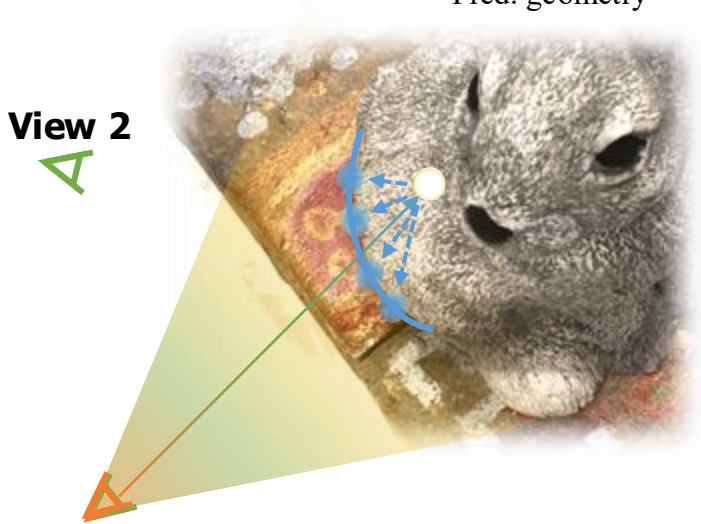
Previous Approach ("estimate → fuse")

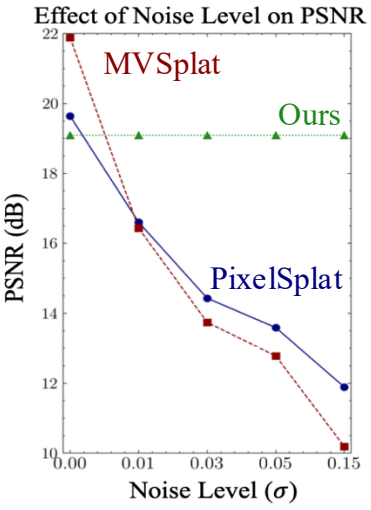
Source view 1
Source view 2
GT geometry
Camera error
Geometry error
Pred. view 2
Pred. geometry



Ours ("fuse → estimate")

Canonical view
Fused feature
Geometry prediction
Pred. geometry





Method: Canonical Space

Fusing the features from different camera viewpoints are not easy…

How do we embed "Pose-awareness" into the fused features?

(-) No Pose-awareness

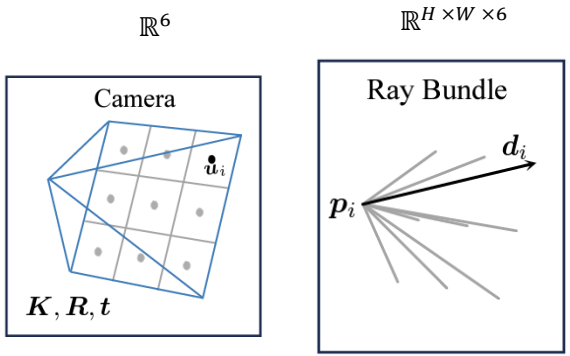
Feature Fusion

Global Features

[Cameras as rays, ICLR'24]

Method: Plücker Ray Estimation and Embedding

1. Conventional camera pose: 6-DoF extrinsics (rotation & translation)
2. Plücker ray field : $R^{\{H \times W \times 6\}}$



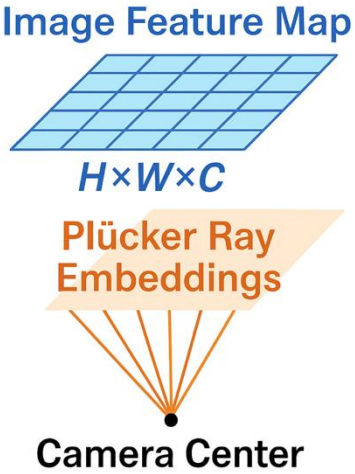
A Plücker ray = (d, m)

$d \in R^3$

$m = o \times d \in R^3$

Lightweight U-Net

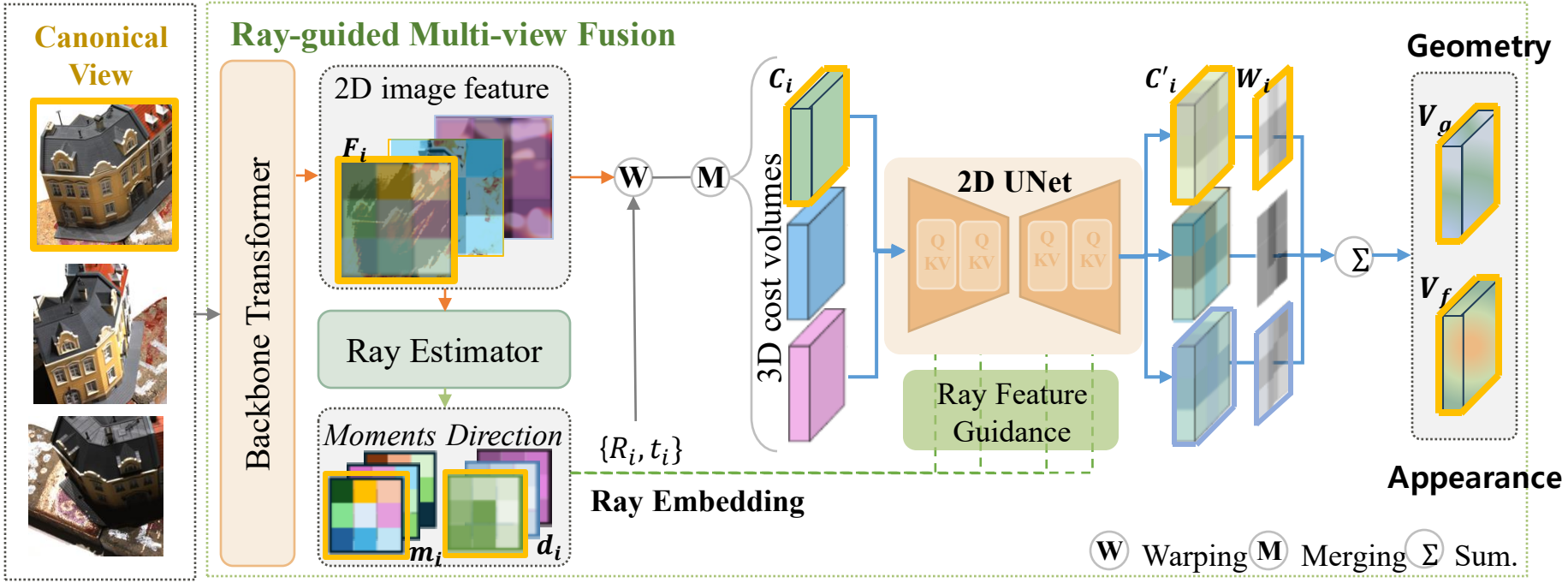
Plücker Ray Embedding



Per-pixel geometric embedding aligned with image features

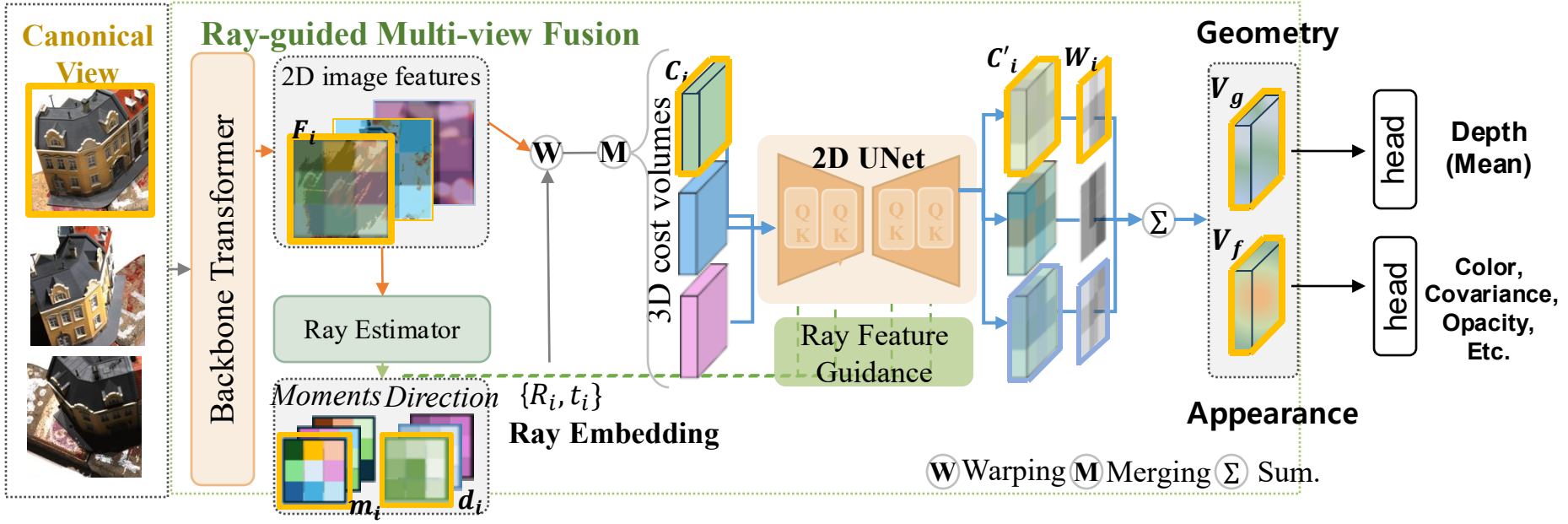
Overview: Ray-Guided View Fusion Pipeline

Our method unifies the two separate problems (pose estimation, geometry estimation) into a single problem.



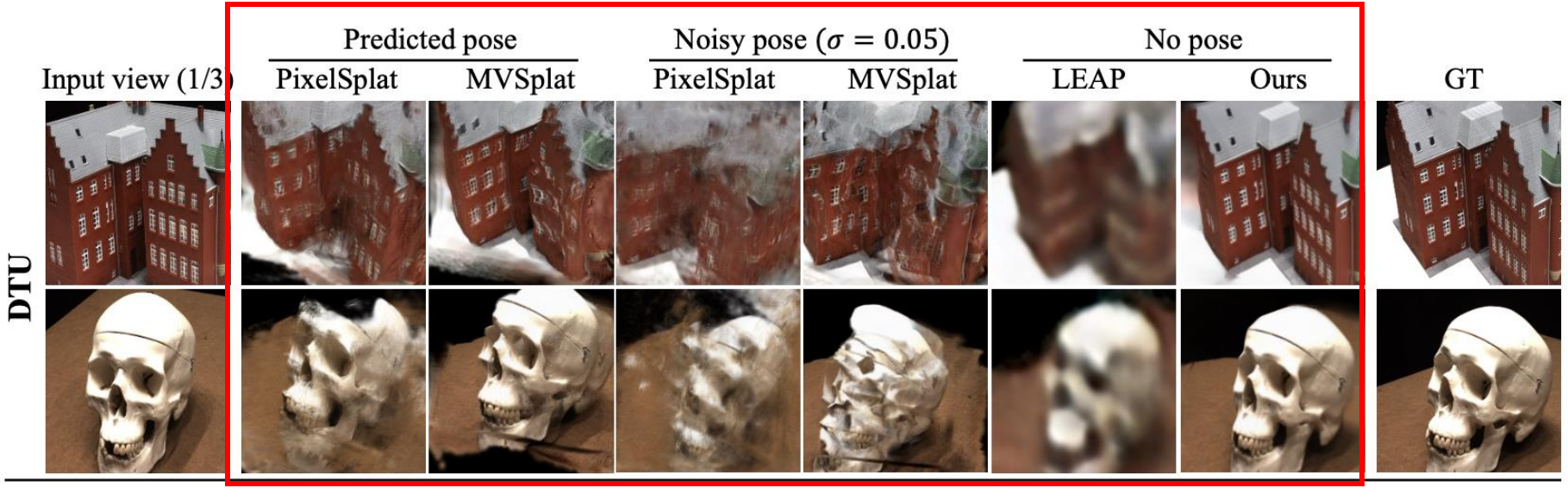
Overview: Ray-Guided View Fusion Pipeline

Our method unifies the two separate problems (pose estimation, geometry estimation) into a single problem.



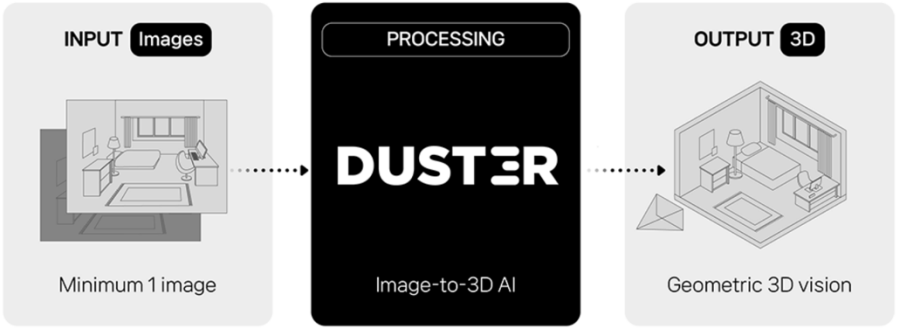
Experimental Results

- Datasets: DTU (small scale), Re10K (large-scale) scene-scale real data.
- Predicted pose: Dust3R, corrupted pose: "inject small noise to GT pose"



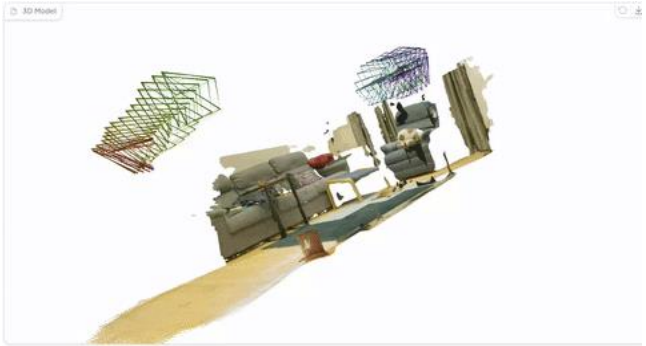
3D Reconstruction Foundation Models

Dust3R (CVPR2024), VGGT (CVPR 2025 best paper award)



Point map -> Gaussian means

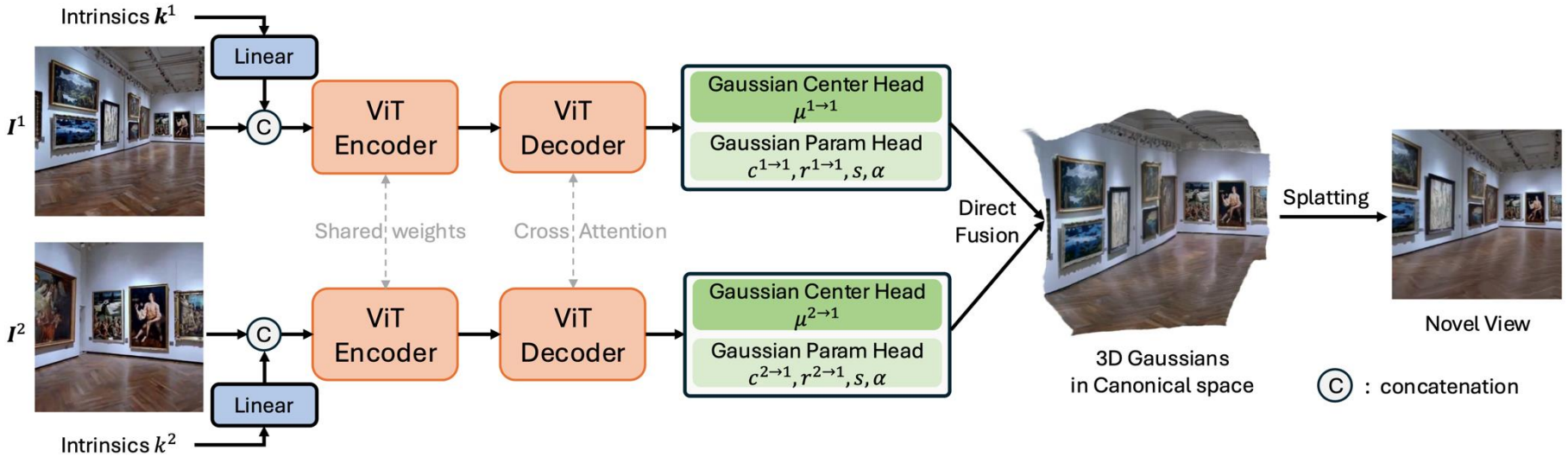
Dust3R



VGGT

3D Reconstruction Foundation Models

NoPoSplat, ICLR 2025 Oral



3D Reconstruction Foundation Models

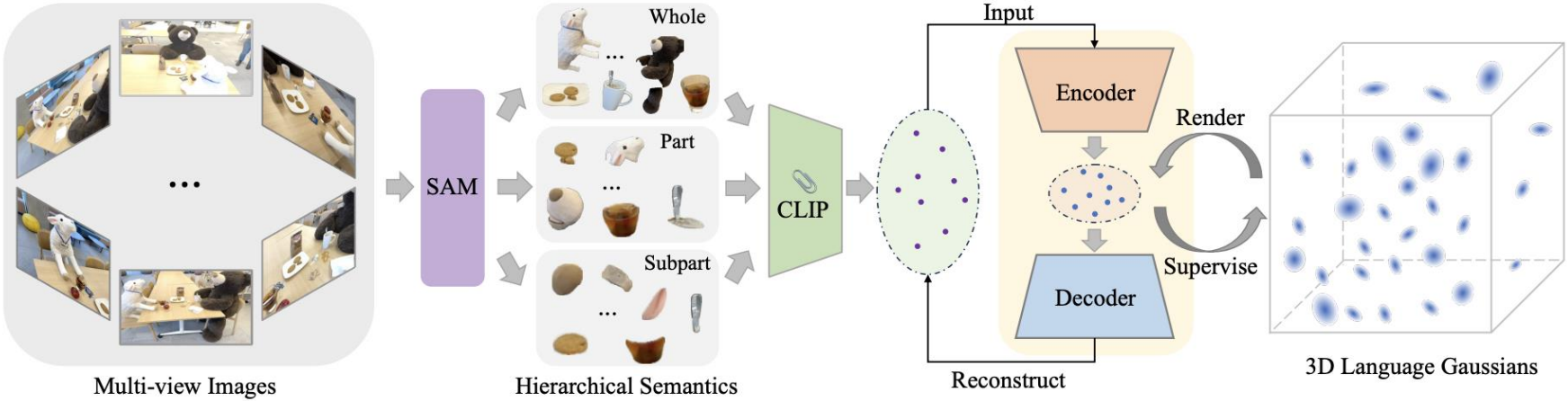
NoPoSplat, ICLR 2025 Oral

4. More Scene Representations

(Semantic Fields, Kinematic Fields)

Scene Understanding: “Semantic (Feature) Fields”

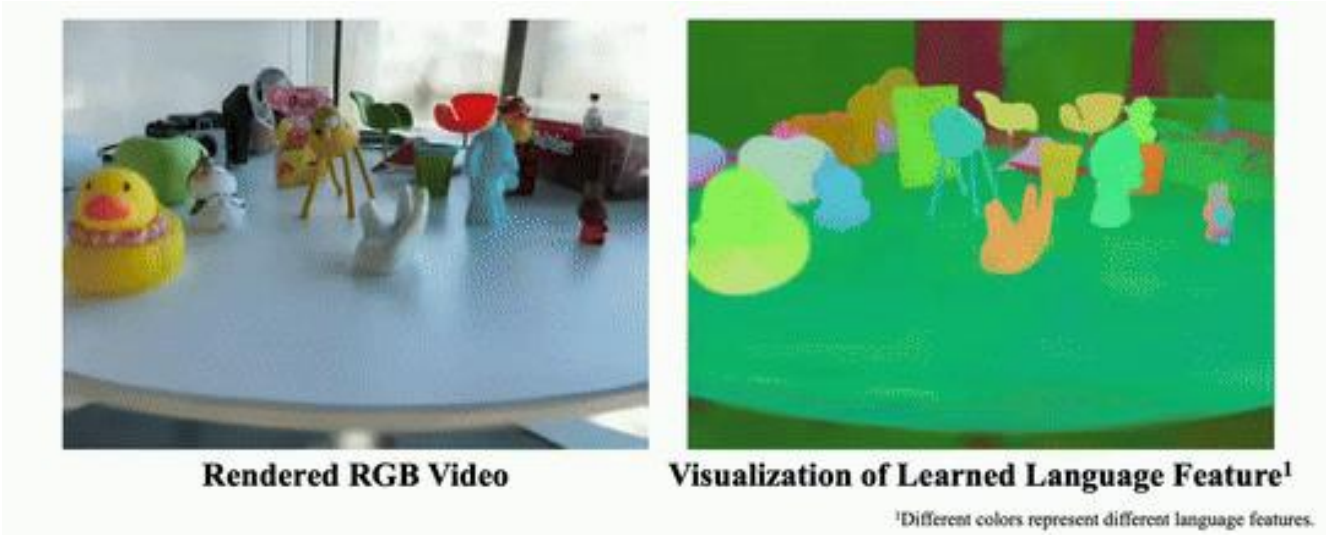
LangSplat, CVPR 2024 Highlight
Geometry + Appearance + Semantics



Scene Understanding: “Semantic (Feature) Fields”

LangSplat, CVPR 2024 Highlight
Geometry + Appearance + Semantics

Unproject 2D features to 3D



Rendered RGB Video

Visualization of Learned Language Feature[1]

[1]Different colors represent different language features.

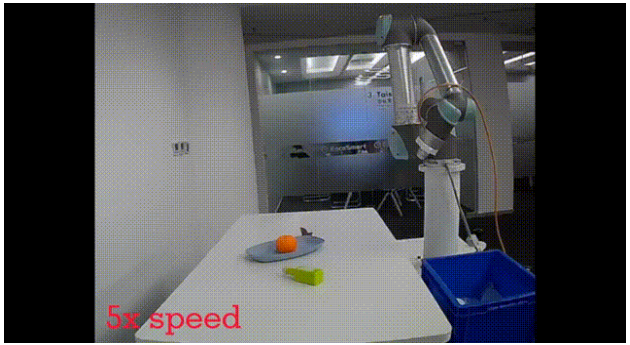
Scene Understanding: “Semantic (Feature) Fields”

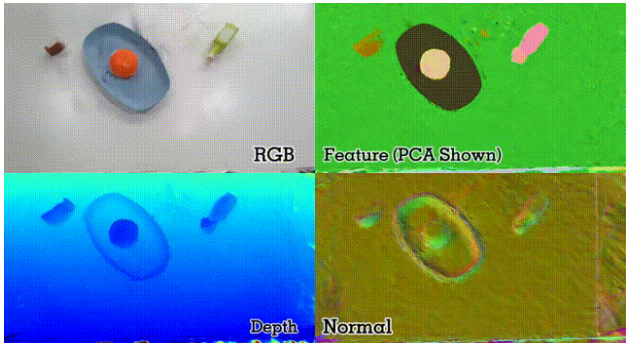
GaussianGrasper, RA-L 2024

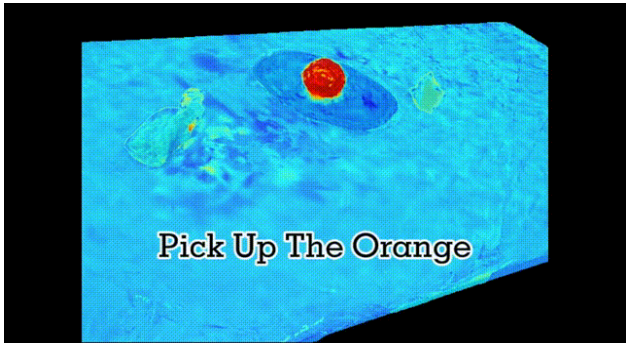
Scene Reconstruction -> Text prompt -> Robot grasping

Scene Representation for Understanding

After grasping -> Scene changes -> Re-optimization







Robot Kinematics Representation

Controlling Diverse Robots by Inferring Jacobian Fields
(Nature, 2025)

Neural Jacobian Fields

Learning to move by seeing yourself

Sizhe Lester Li, Annan Zhang, Boyuan Chen,
Hanna Matusik, Chao Liu, Daniela Rus, Vincent Sitzmann

MIT CSAIL SCENE REPRESENTATION GROUP

(TL;DR) Our system learns to control any robot with a single camera as the only sensor by watching it execute random actions

Robot Kinematics Representation

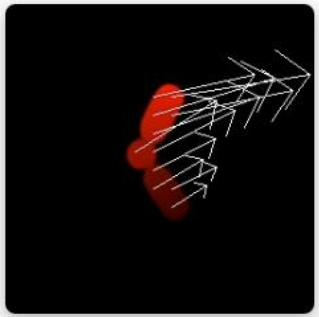
Controlling Diverse Robots by Inferring Jacobian Fields
(Nature, 2025)

$$\begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = J(q) \begin{bmatrix} \delta q_1 \\ \delta q_2 \end{bmatrix}$$

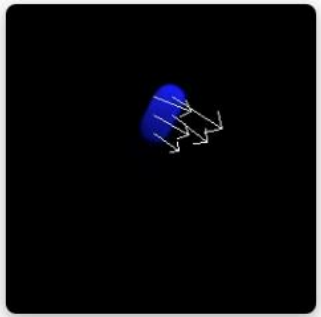
RGB observation

G.T. optical flow

Pred. optical flow



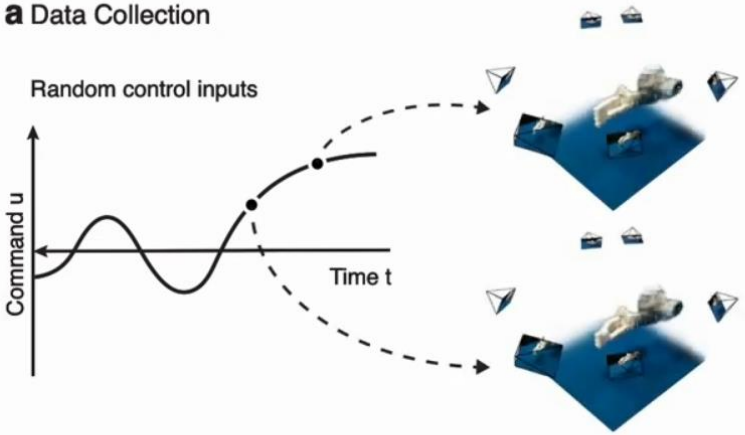
Jac. channel 1



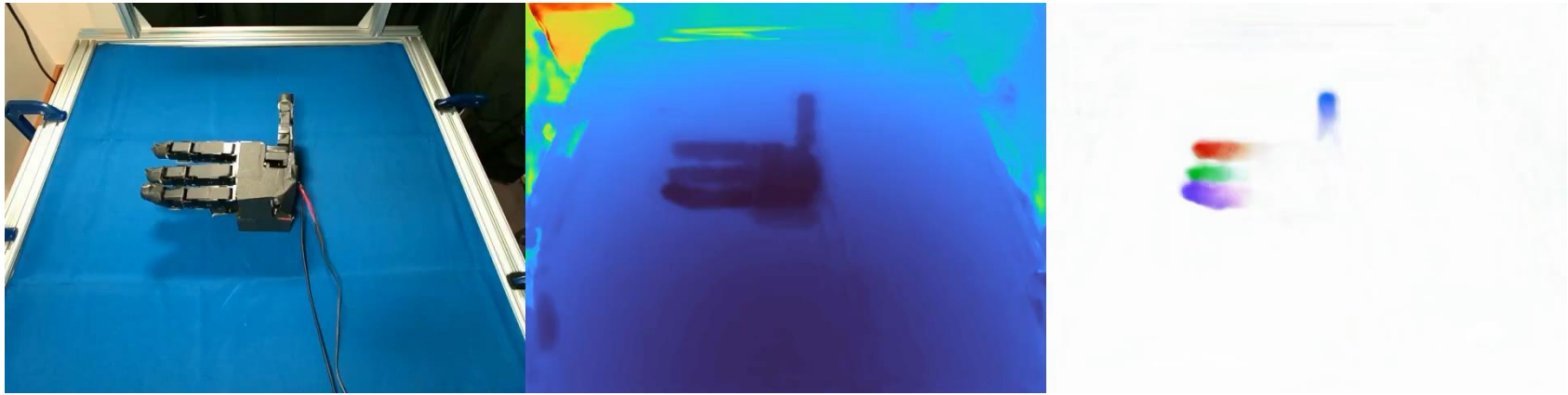
Jac. channel 2

Jacobian Fields (kinematic response)

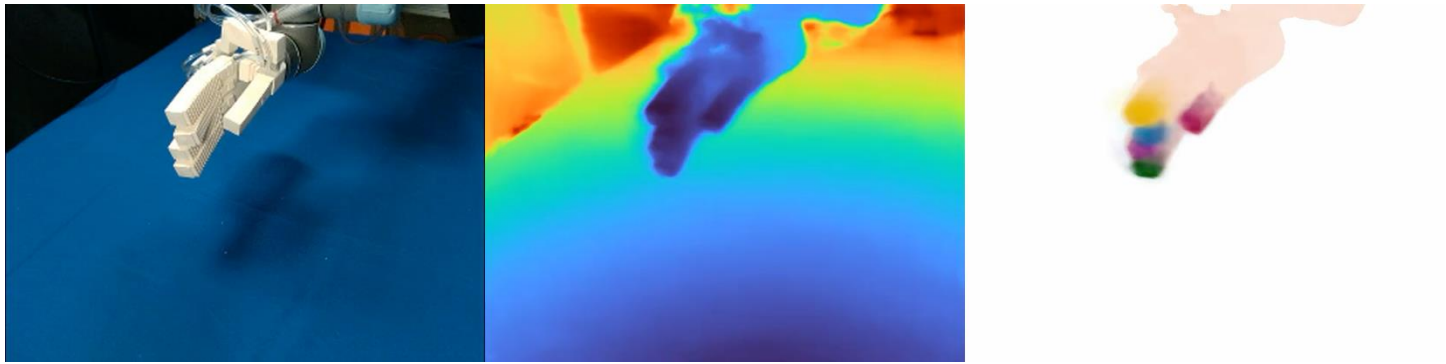
Trained with just multiview videos and random actions



Jacobian Fields Rendering



Results on Allegro Hand. (Left) Input video, (Middle) Depth Prediction, (Right) Jacobian Prediction



Results on Pneumatic Hand. (Left) Input Image, (Middle) Depth Prediction, (Right) Jacobian Prediction

Conclusion

- In the past few years, we have witnessed remarkable progress in training **3D-aware neural scene representations** — from NeRF and 3D Gaussian Splatting to 3D foundation models.

- We are now entering an era where such models can learn directly from unconstrained, real-world videos, jointly reasoning about **geometry, appearance, motion, and even physical causality**.

- These advances mark the transition from reconstructing the world to **simulating and predicting its behavior** through learned physics and dynamics.

- The next frontier lies in building **neural physical (world) models** — unified systems that serve as neural simulators, enabling agents to perceive, reason, and act within complex 3D spaces.

Thank you