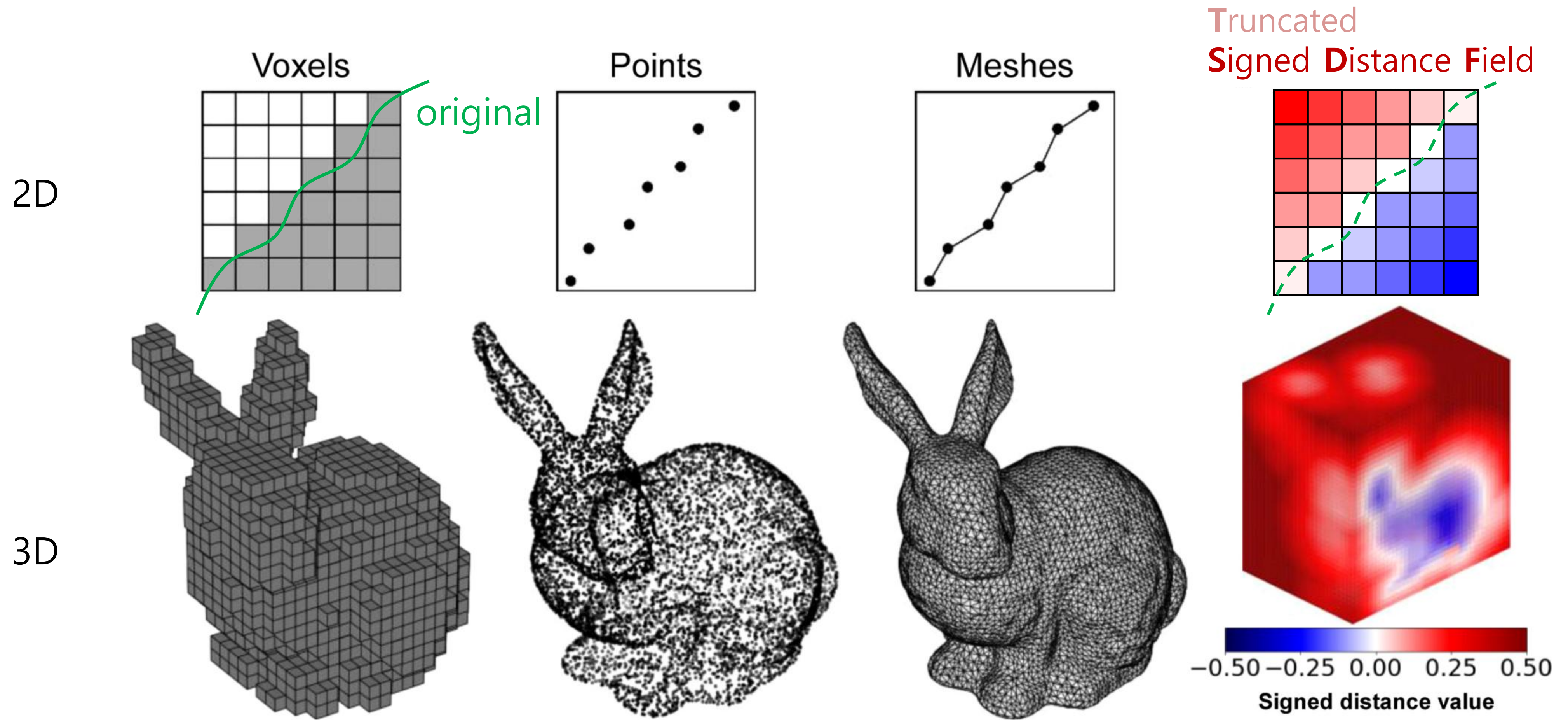
Note) More detailed slides and example codes are available:
https://github.com/mint-lab/3dv_tutorial

An Invitation to 3D Vision:
3D Representations

Sunglok Choi, Assistant Professor, Ph.D.
Dept. of Computer Science and Engineering, SEOULTECH
sunglok@seoultech.ac.kr | https://mint-lab.github.io/

Motivation) 3D Shape Representations

- Explicit vs. Implicit representations



Why not a neural network?

Image: Attar et al., "Implicit Neural Representations of Sheet Stamping Geometries with Small-Scale Features", EAAI, 2023 DOI

NeRF (Neural Radiance Field; 2020)

- **NeRF** is a **multi-layer perceptron** for rendering an image from a new viewpoint.
- Network: **11 fully-connected (shortly FC) layers**
  - Input: Spatial location $\mathbf{x} = (x, y, z)$ and viewing direction $\mathbf{d} = (d_x, d_y, d_z)$ on a unit sphere
  - Output: RGB color (RGB) and density (σ)

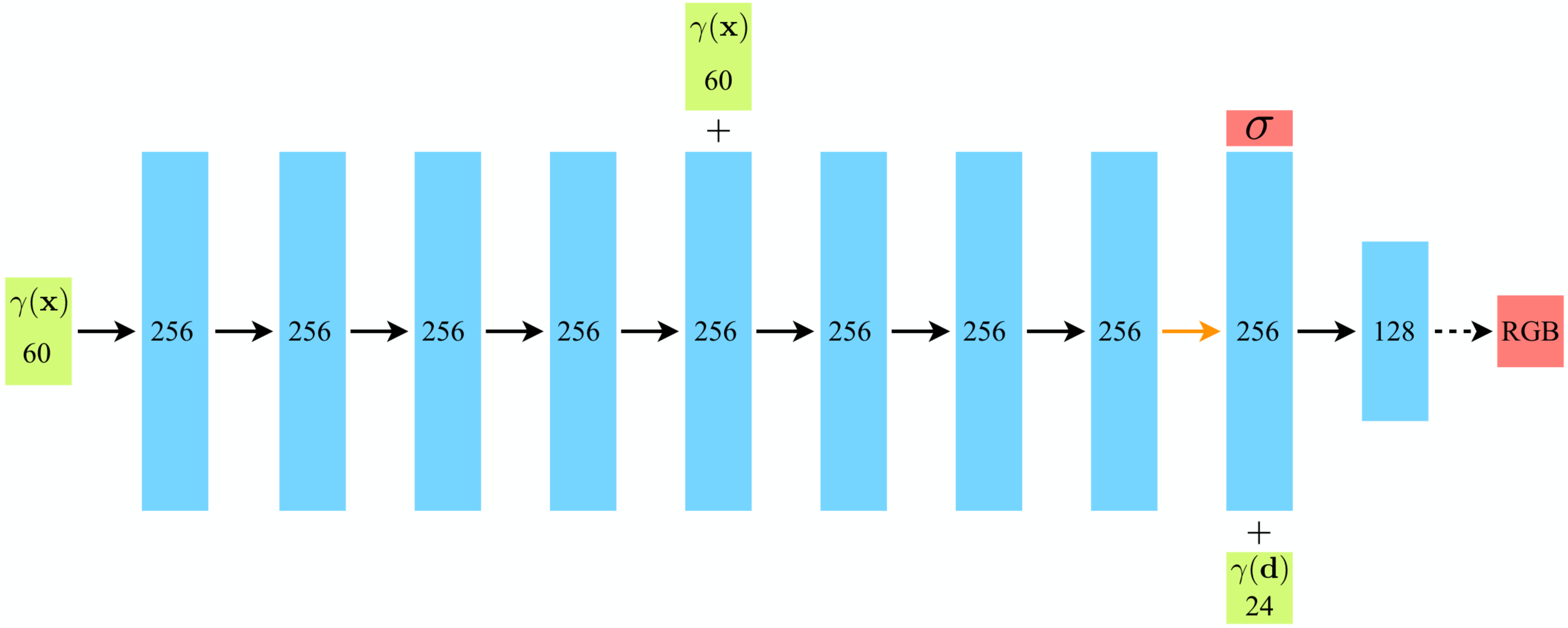
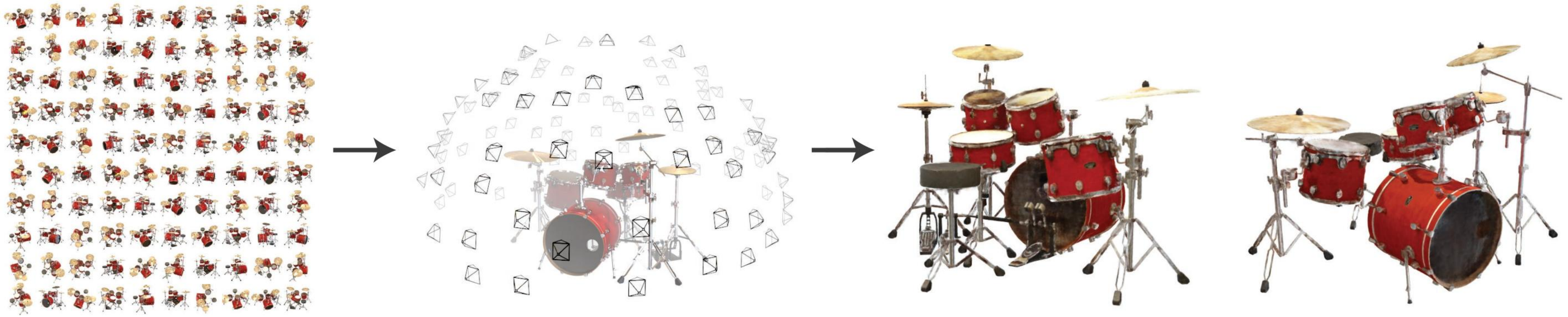


Image: Mildenhall et al., "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV, 2020 Homepage

NeRF (Neural Radiance Field; 2020)

implicitly representing a 3D scene.

- **NeRF** is a multi-layer perceptron for ~~rendering an image from a new viewpoint~~.
- **Training**: Learning the 3D scene
  - Input: Images with their 3D viewpoints $(\mathrm{R}_j, \mathbf{t}_j)$
    - Note) 3D viewpoints can be retrieved by SfM (e.g. COLMAP).
- **Inference**: Synthesizing a 2D image with a *new* viewpoint
  - Input: A new camera viewpoint $(\mathrm{R}_n, \mathbf{t}_n)$



Input images

Training the NeRF network

Rendering new views

Image: Mildenhall et al., "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV, 2020 [Homepage]

NeRF (Neural Radiance Field; 2020)

- **NeRF** is a multi-layer perceptron for **rendering an image from a new viewpoint**.
- **Inference**: Synthesizing a 2D image with a *new* viewpoint
  - Input: A new camera viewpoint $(\mathrm{R}_n, \mathbf{t}_n)$

Image: Mildenhall et al., "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV, 2020 Homepage

NeRF (Neural Radiance Field; 2020)

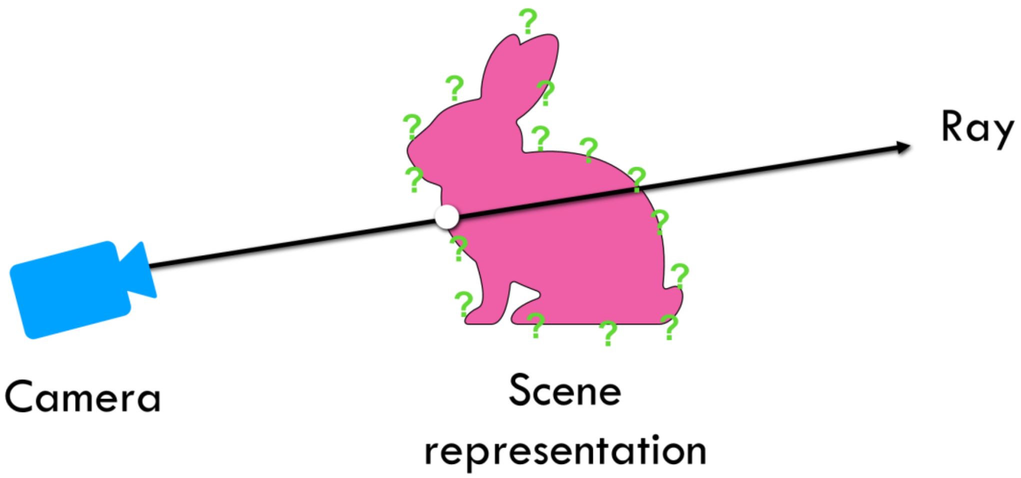
- Key idea: **Neural Volumetric Rendering**
  - \+ Continuous rendering
  - \+ Differentiable rendering
  - \+ Model without concrete ray/surface intersections

Slide: Ben Mildenhall, "Deep Dive into the Volumetric Rendering Function", NeRF Tutorial, ECCV 2022

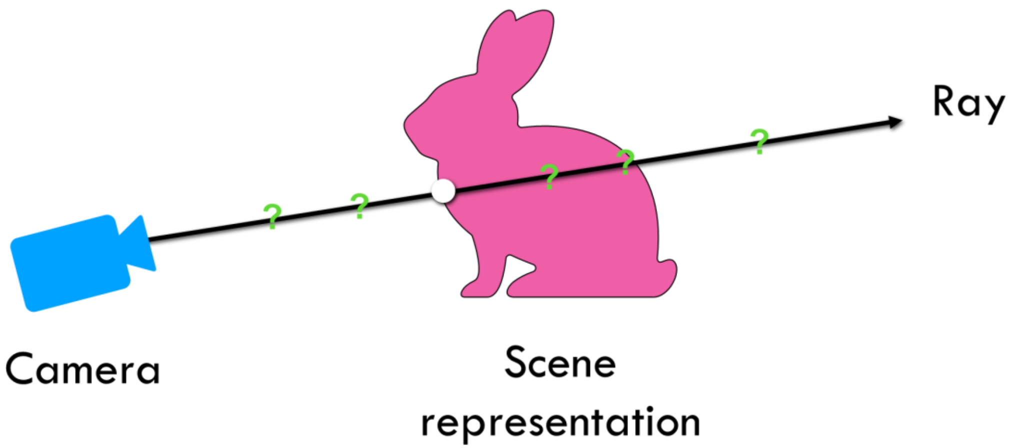
NeRF (Neural Radiance Field; 2020)

- Key idea: **Neural Volumetric Rendering**

Surface rendering
(loop over geometry, check for ray hits)



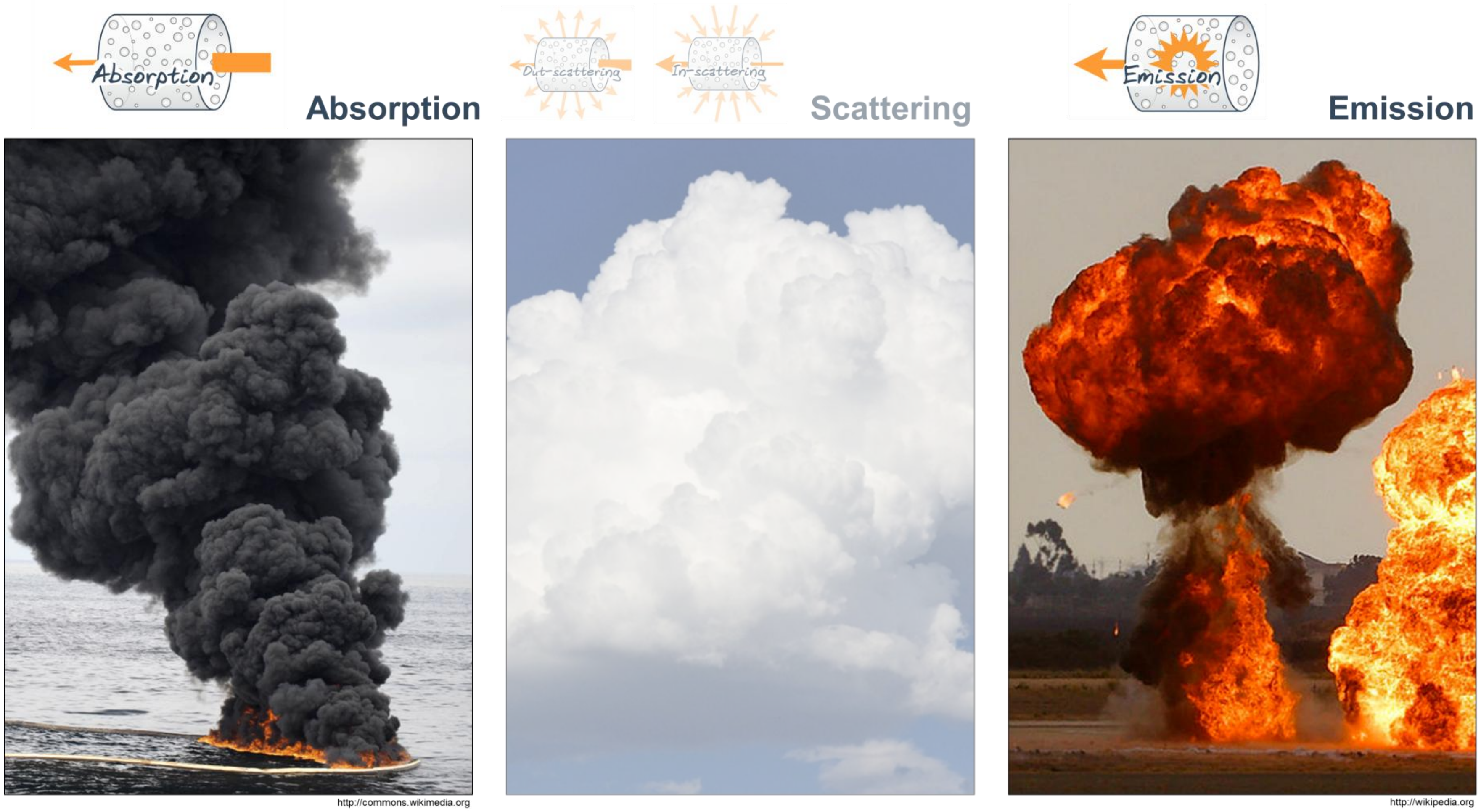
Volume rendering
(loop over ray points, query geometry)



Slide: Ben Mildenhall, "Deep Dive into the Volumetric Rendering Function", NeRF Tutorial, ECCV 2022

NeRF (Neural Radiance Field; 2020)

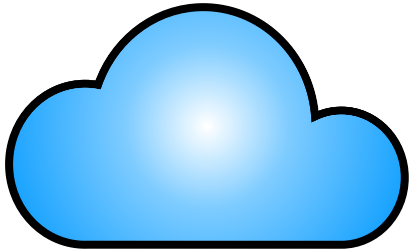
- Key idea: **Neural Volumetric Rendering**
  - Based on the simplified physics (ignoring **scattering**)



Slide: Ben Mildenhall, "Deep Dive into the Volumetric Rendering Function", [NeRF Tutorial](), ECCV 2022

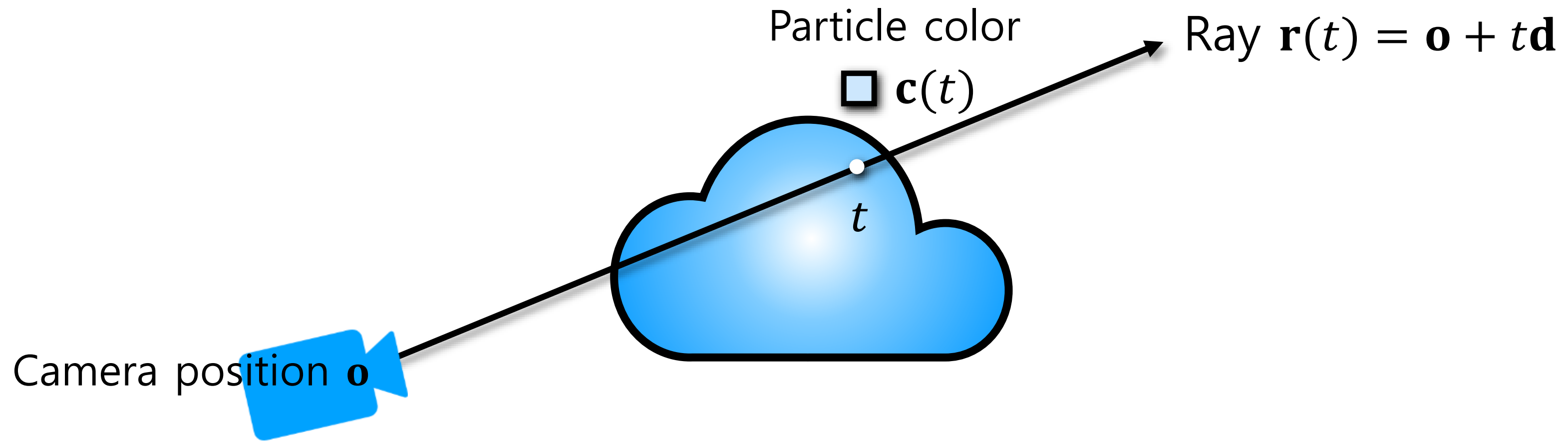
NeRF (Neural Radiance Field; 2020)

- Key idea: **Neural Volumetric Rendering**
  - The scene is a cloud composed of tiny colored particles.



NeRF (Neural Radiance Field; 2020)

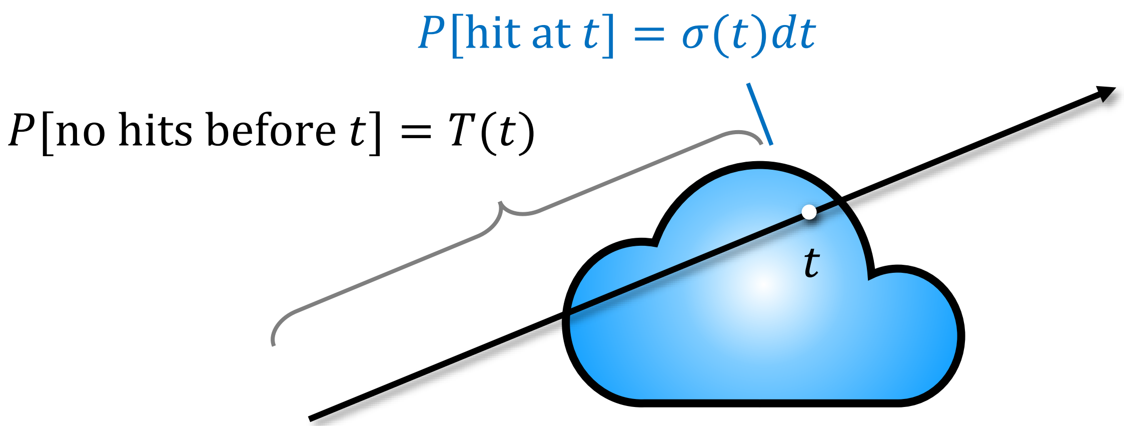
- Key idea: **Neural Volumetric Rendering**
  - If a **ray** (traveling through the scene) hits a **particle** at distance t (along the ray), we can retrieve its color $\mathbf{c}(t)$.



Slide: Ben Mildenhall, "Deep Dive into the Volumetric Rendering Function", NeRF Tutorial, ECCV 2022

NeRF (Neural Radiance Field; 2020)

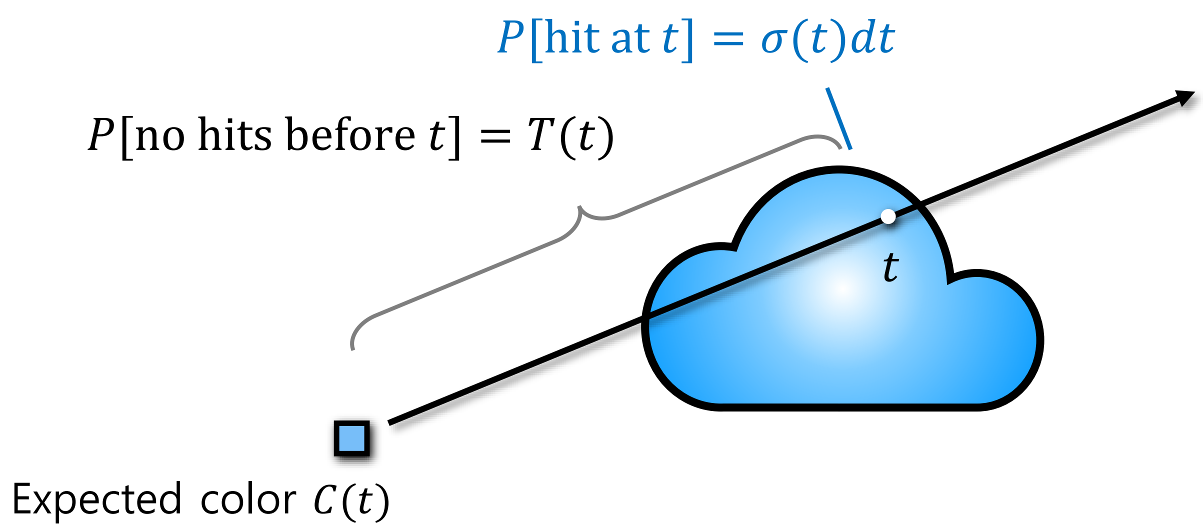
- Key idea: **Neural Volumetric Rendering**
  - The product of these probabilities tells us how much you see the particles at t:
    - $P[\text{first hit at } t] = P[\text{no hit before } t] \times P[\text{hit at } t] = T(t)\sigma(t)dt$
    - $T(t)$: Transmittance (the probability that the ray doesn't hit any particles earlier)



NeRF (Neural Radiance Field; 2020)

- Key idea: **Neural** **Volumetric Rendering**
  - The product of these probabilities tells us how much you see the particles at t:
    - $P[\text{first hit at } t] = P[\text{no hit before } t] \times P[\text{hit at } t] = T(t)\sigma(t)dt$
  - Expected color by a ray $\mathbf{r}$

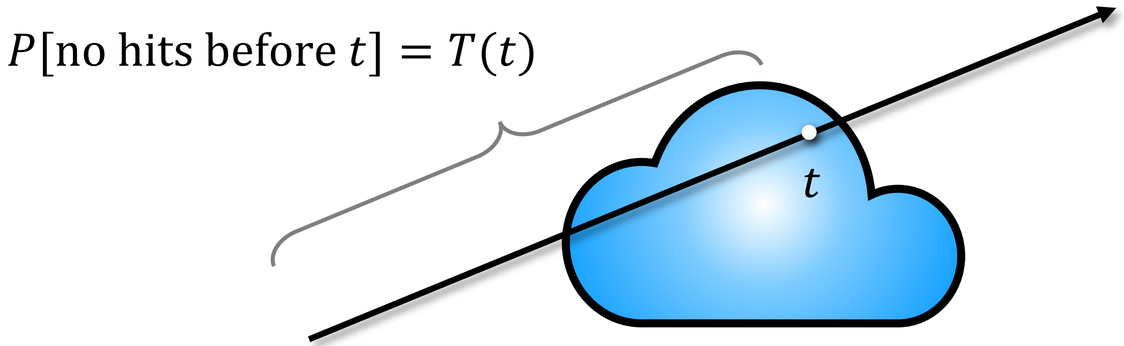
$$C(\mathbf{r}) = \int_{t_0}^{t_1} T(t)\sigma(t)\mathbf{c}(t)dt \approx \sum_{i=1}^{N} T_i \mathbf{c}_i \big(1 - \exp(-\sigma_i \delta_i)\big) \quad \text{where} \quad T_i = \exp(-\sum_{j=1}^{i-1} \sigma_j \delta_j)$$



NeRF (Neural Radiance Field; 2020)

- Key idea: **Neural Volumetric Rendering**
  - Q) Can we represent $T(t)$ using the (known) density function $\sigma(t)$?
    - A recursive form: $P[\text{no hit before } t + dt] = P[\text{no hit before } t] \times P[\text{no hit at } t]$

$$T(t + dt) = T(t)(1 - \sigma(t)dt) \;\; \rightarrow \;\; T(t) = \exp\left(-\int_{t_0}^{t} \sigma(s)ds\right)$$



NeRF (Neural Radiance Field; 2020)

- Key idea: **Neural Volumetric Rendering**
  - Q) Can we represent $T(t)$ using the (known) density function $\sigma(t)$?
    - A recursive form: $P[\text{no hit before } t + dt] = P[\text{no hit before } t] \times P[\text{no hit at } t]$

$$T(t + dt) = T(t)(1 - \sigma(t)dt) \;\rightarrow\; T(t) = \exp\left(-\int_{t_0}^{t} \sigma(s)ds\right)$$

  - Solving the differential equation

$T(t + dt) = T(t)(1 - \sigma(t)dt)$

$T(t) + T'(t)dt = T(t) - T(t)\sigma(t)dt$ (Taylor expansion for T)

$\frac{T'(t)}{T(t)}dt = -\sigma(t)dt$ (Rearrange)

$\log T(t) = -\int_{t_0}^{t} \sigma(s)ds$ (Integrate from t_0 to t)

$T(t) = \exp\left(-\int_{t_0}^{t} \sigma(s)ds\right)$ (Exponentiate)

Slide: Ben Mildenhall, "Deep Dive into the Volumetric Rendering Function", NeRF Tutorial, ECCV 2022

NeRF (Neural Radiance Field; 2020)

- Key idea: **Neural Volumetric Rendering**
  - Q) How can we integrate the color equation?

$$C(\mathbf{r}) = \int_{t_0}^{t_1} T(t)\sigma(t)\mathbf{c}(t)dt \approx \sum_{i=1}^{N} T_i\mathbf{c}_i\big(1 - \exp(-\sigma_i\delta_i)\big)$$

  - Using the quadrature rule (구분구적법 in Korean)

$$\int T(t)\sigma(t)\mathbf{c}(t)dt \approx \sum_{i=1}^{n}\int_{t_i}^{t_{i+1}} T(t)\sigma_i\mathbf{c}_i dt \qquad \text{(Quadrature rule)}$$

$$= \sum_{i=1}^{n} T_i\sigma_i\mathbf{c}_i\int_{t_i}^{t_{i+1}} \exp(-\sigma_i(t - t_i))dt \qquad \text{(Substitute)}$$

$$= \sum_{i=1}^{n} T_i\sigma_i\mathbf{c}_i \frac{\exp(-\sigma_i(t_{i+1}-t_i))-1}{-\sigma_i} \qquad \text{(Integrate)}$$

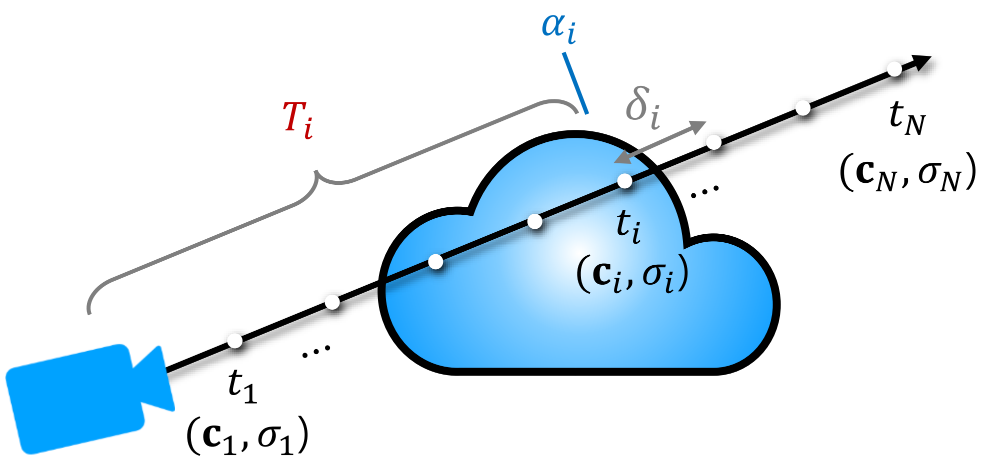
$$= \sum_{i=1}^{n} T_i\mathbf{c}_i(1 - \exp(-\sigma_i\delta_i)) \qquad \text{(Cancel } \sigma_i \text{ and substitute } \delta_i = t_{i+1} - t_i\text{)}$$

NeRF (Neural Radiance Field; 2020)

- Key idea: **Neural Volumetric Rendering**
  - Rendering an image for a ray $\mathbf{r} = \mathbf{o} + t\mathbf{d}$

$$C(\mathbf{r}) = \sum_{i=1}^{N} T_i \alpha_i \mathbf{c}_i \quad \text{where} \quad \alpha_i = 1 - \exp(-\sigma_i \delta_i) \quad \text{and} \quad T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$

    - T_i: How much light is blocked before ray segment i?
    - α_i: How much light is contributed by ray segment i?



Slide: Ben Mildenhall, "Deep Dive into the Volumetric Rendering Function", NeRF Tutorial, ECCV 2022

NeRF (Neural Radiance Field; 2020)

- **Inference**: Synthesizing a 2D image with a *new* viewpoint
  - Input: A new camera viewpoint $(\mathrm{R}_n, \mathbf{t}_n)$
- **Training**: Learning the 3D scene
  - Input: Images with their 3D viewpoints $(\mathrm{R}_j, \mathbf{t}_j)$
    - Note) 3D viewpoints can be retrieved by SfM (e.g. COLMAP).
  - Loss function: Rendering loss (MSE) between the *input* and *synthesized* images at each $(\mathrm{R}_j, \mathbf{t}_j)$
    - The *synthesized* images are generated by the neural volumetric rendering.

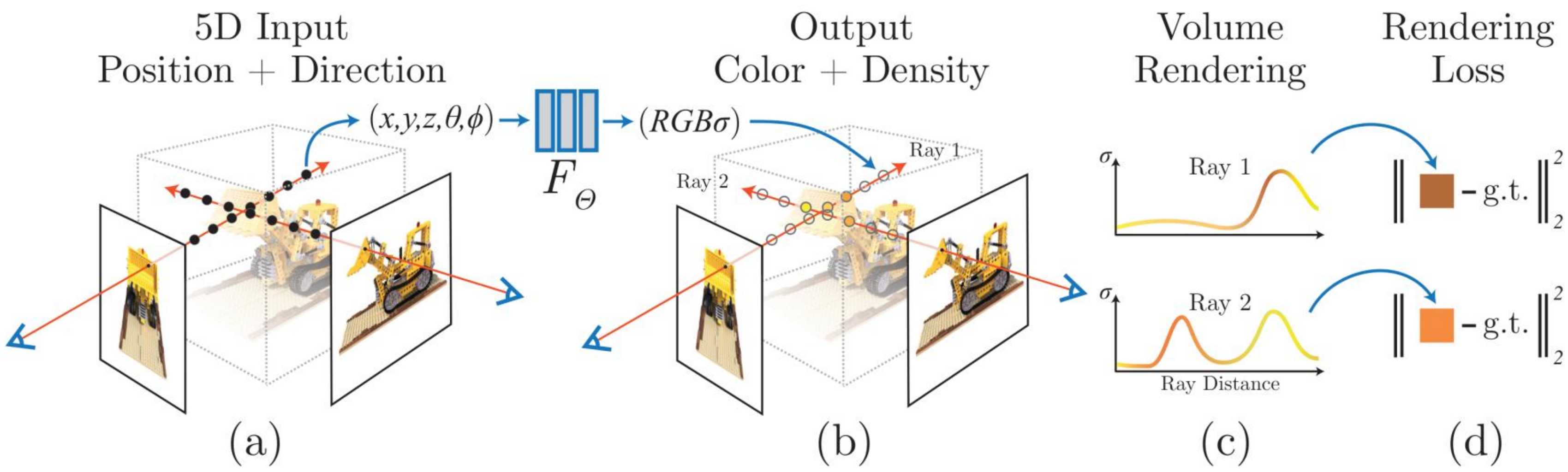


Image: Mildenhall et al., "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV, 2020 Homepage

NeRF (Neural Radiance Field; 2020)

- Network: **11 fully-connected (shortly FC) layers**
  - Input: Spatial location $\mathbf{x} = (x, y, z)$ and viewing direction $\mathbf{d} = (d_x, d_y, d_z)$ on a unit sphere
    - Q) What is a function γ? Why are the input dimensions 60 and 24?
  - Output: RGB color (RGB) and density (σ)

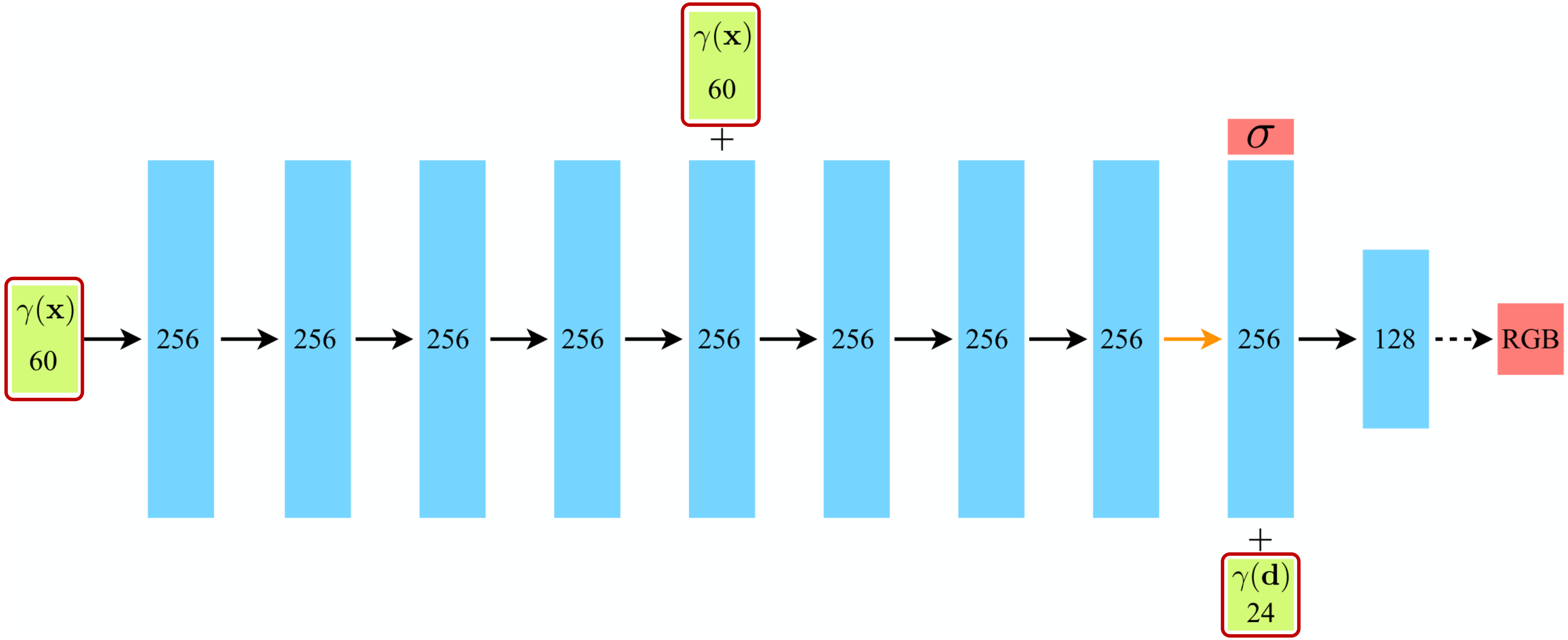


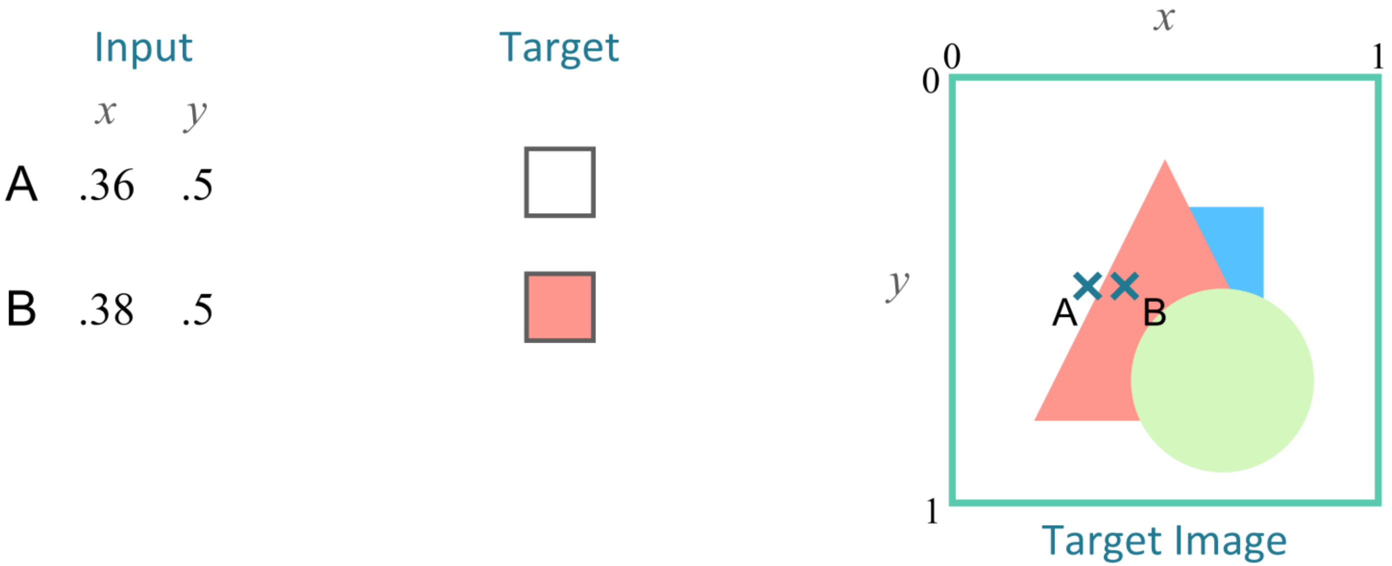
Image: Mildenhall et al., "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV, 2020 Homepage

NeRF (Neural Radiance Field; 2020)

- Key idea: **Positional encoding** transforms a real number $p \rightarrow 2L$-dimensional real numbers.

$$\gamma(p) = \left(\sin(2^0 \pi p), \cos(2^0 \pi p), \cdots \sin(2^{L-1} \pi p), \cos(2^{L-1} \pi p)\right)$$

  – Q) Why? 3.1415 and 3.1414 may generate similar values.



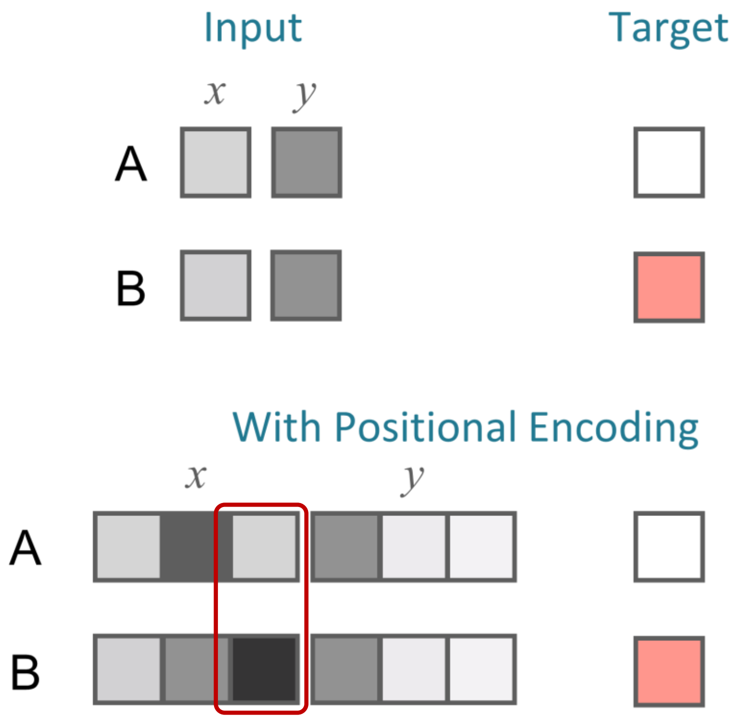
Slide: Matt Tancik, "Encoding and Representing 3D Volumes", [NeRF Tutorial](), ECCV 2022

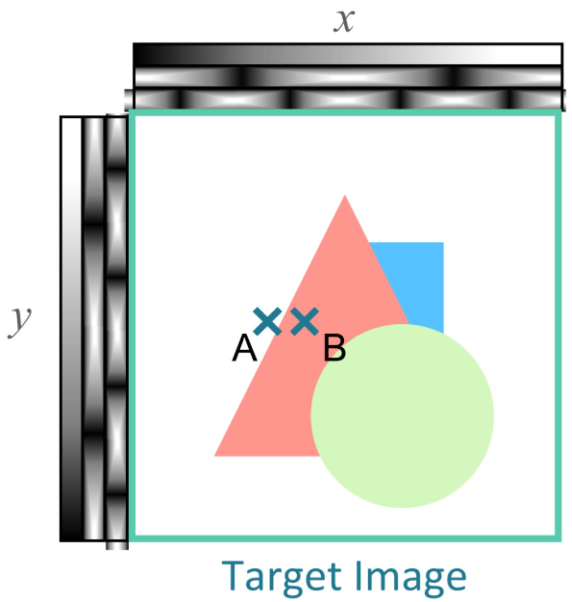
NeRF (Neural Radiance Field; 2020)

- Key idea: **Positional encoding** transforms a real number $p \rightarrow 2L$-dimensional real numbers.

$$\gamma(p) = \left(\sin(2^0 \pi p),\, \cos(2^0 \pi p), \cdots \sin(2^{L-1} \pi p),\, \cos(2^{L-1} \pi p)\right)$$

  – Q) Why? 3.1415 and 3.1414 may generate similar values.





NeRF (Neural Radiance Field; 2020)

- Key idea: **Positional encoding** transforms a real number $p \rightarrow 2L$-dimensional real numbers.

$$\gamma(p) = \left(\sin(2^0 \pi p), \cos(2^0 \pi p), \cdots \sin(2^{L-1} \pi p), \cos(2^{L-1} \pi p)\right)$$

  - Q) Why? 3.1415 and 3.1414 may generate similar values.
  - A) Positional encoding highlights not only large values but also small fraction numbers.
  - Note) $L = 10$ for $\gamma(\mathbf{x})$ and $L = 4$ for $\gamma(\mathbf{d})$

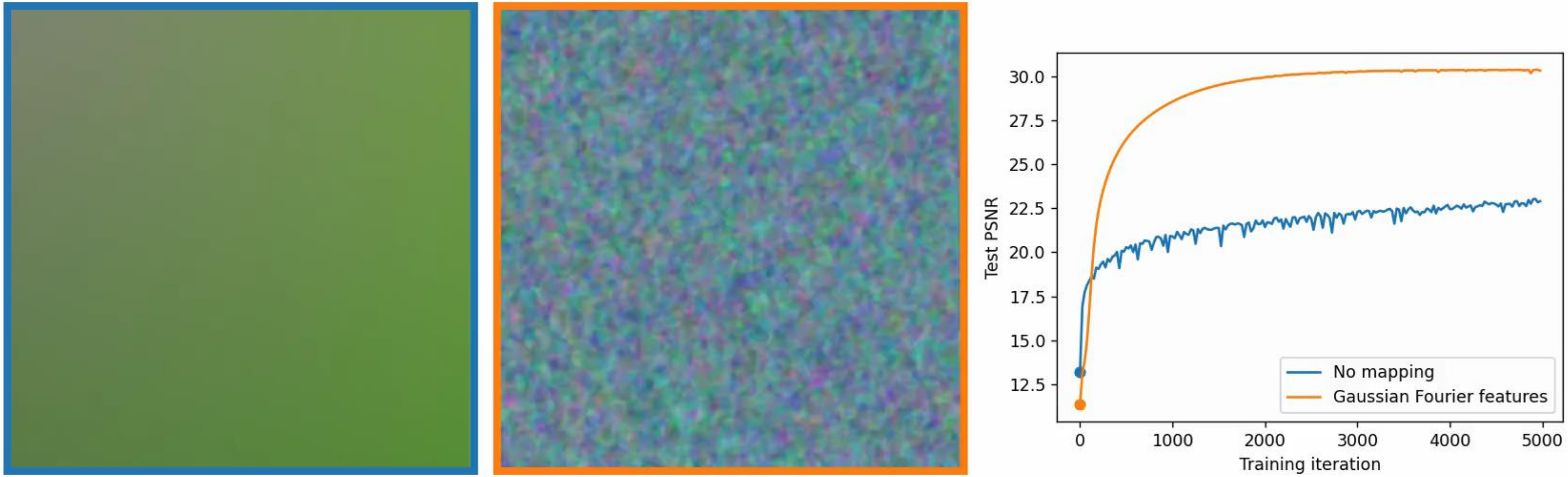
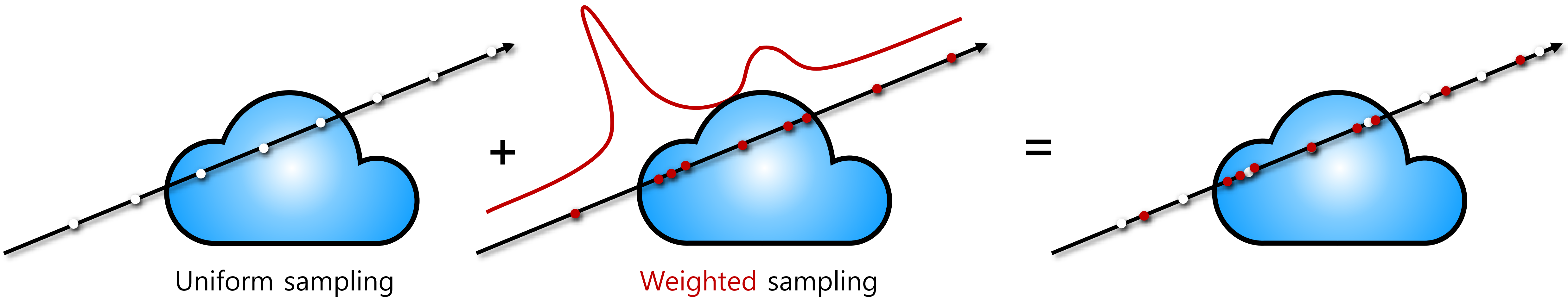


Image: Mildenhall et al., "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV, 2020 Homepage

NeRF (Neural Radiance Field; 2020)

- Key idea: **Hierarchical volume sampling** selects points according to uniform and non-uniform weights.
  - Q) How to assign non-uniform weights?

$$C(\mathbf{r}) = \sum_{i=1}^{N} T_i \alpha_i \mathbf{c}_i = \sum_{i=1}^{N} w_i \mathbf{c}_i \rightarrow \hat{w}_i = \frac{w_i}{\sum w_i}$$



NeRF (Neural Radiance Field; 2020)

- **Inference**: Synthesizing a **2D image** with a *new* viewpoint
  - Input: A new camera viewpoint $(\mathrm{R}_n, \mathbf{t}_n)$
  - Neural volumetric rendering
- **Inference**: Retrieval of a **3D model** from density values
  - e.g. Normal vectors from analytic gradient of density

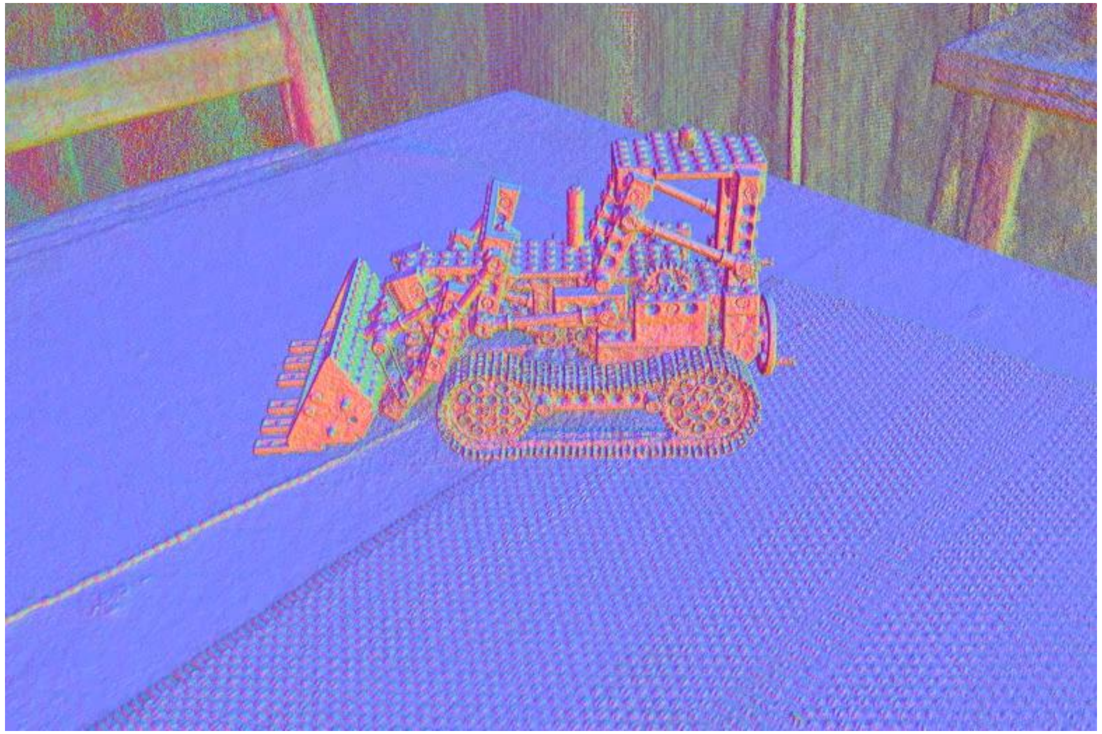
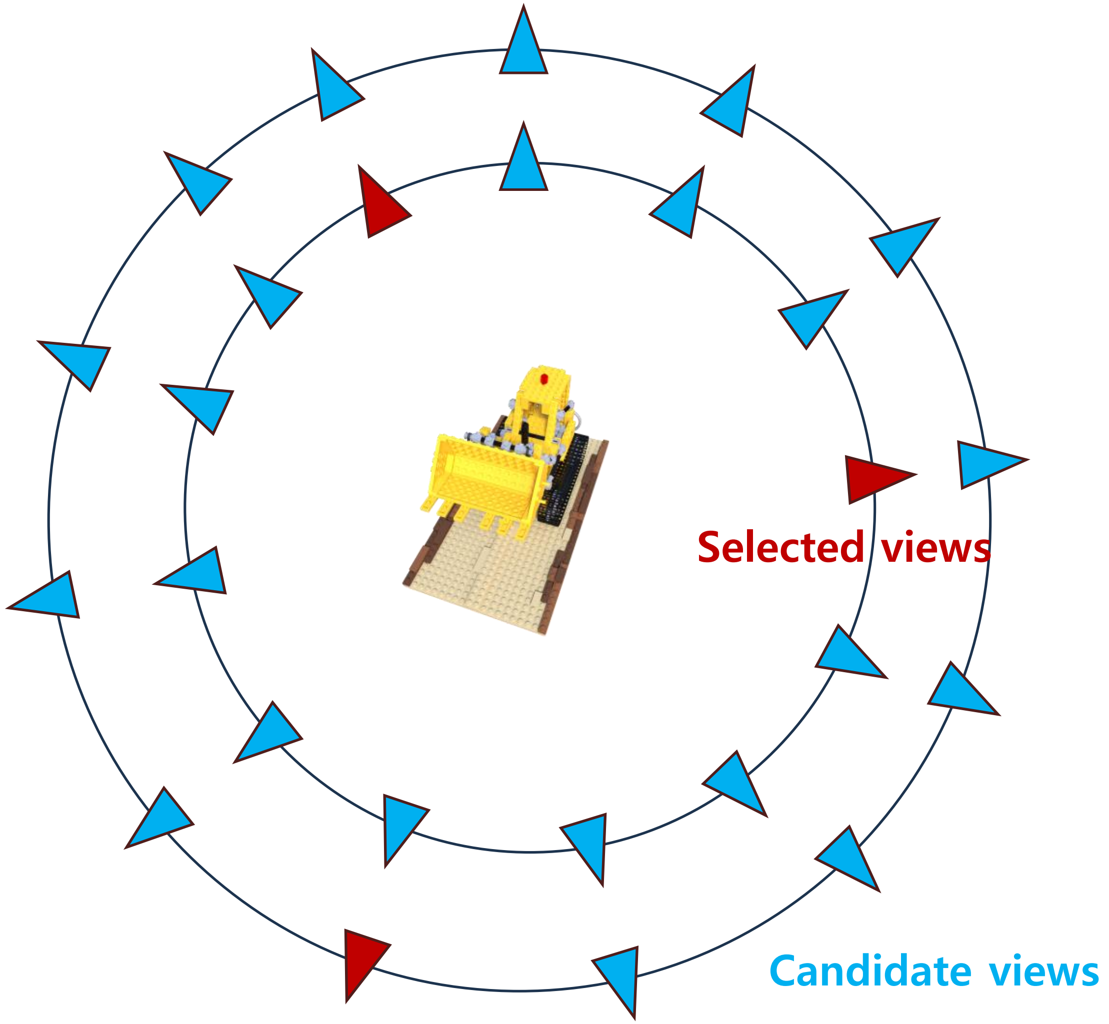
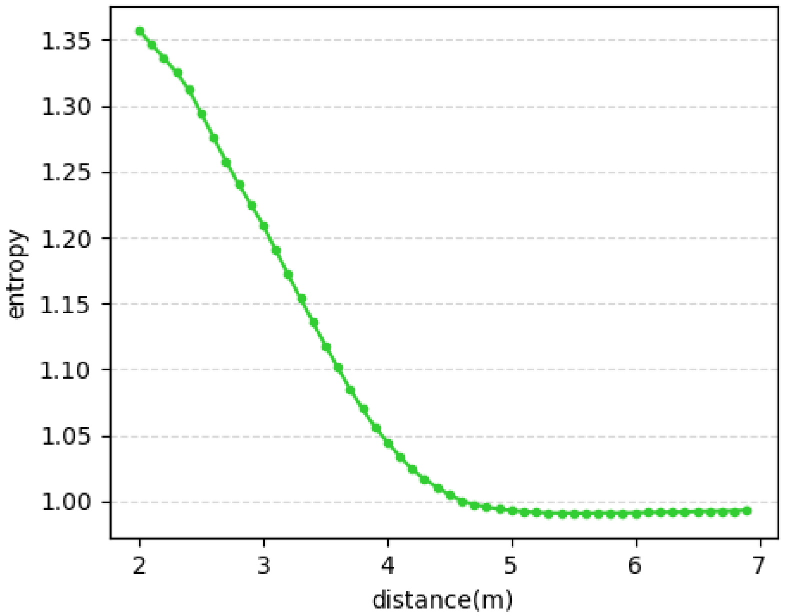


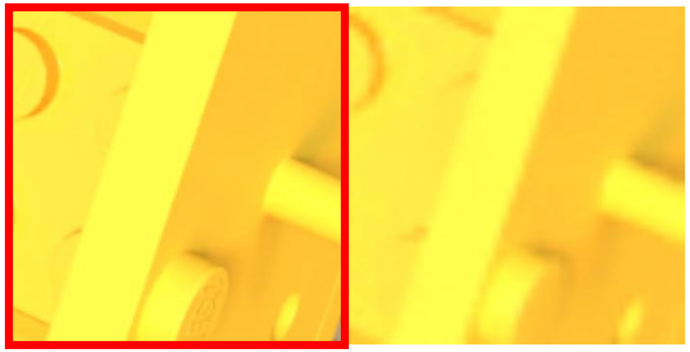
Image: Ben Mildenhall, "Deep Dive into the Volumetric Rendering Function", NeRF Tutorial, ECCV 2022

ETRI-3DV Project: Next-Best-View Selection for Complete 3D Reconstruction

- 3D representation: **NeRF**
- Uncertainty measure: **Entropy** (interpreting density ρ as probability) with **distance-based regularization**







ETRI-3DV Project: Next-Best-View Selection for Complete 3D Reconstruction

- Evaluation: [Lee et al., RA-L, 2023] vs. Proposed Method

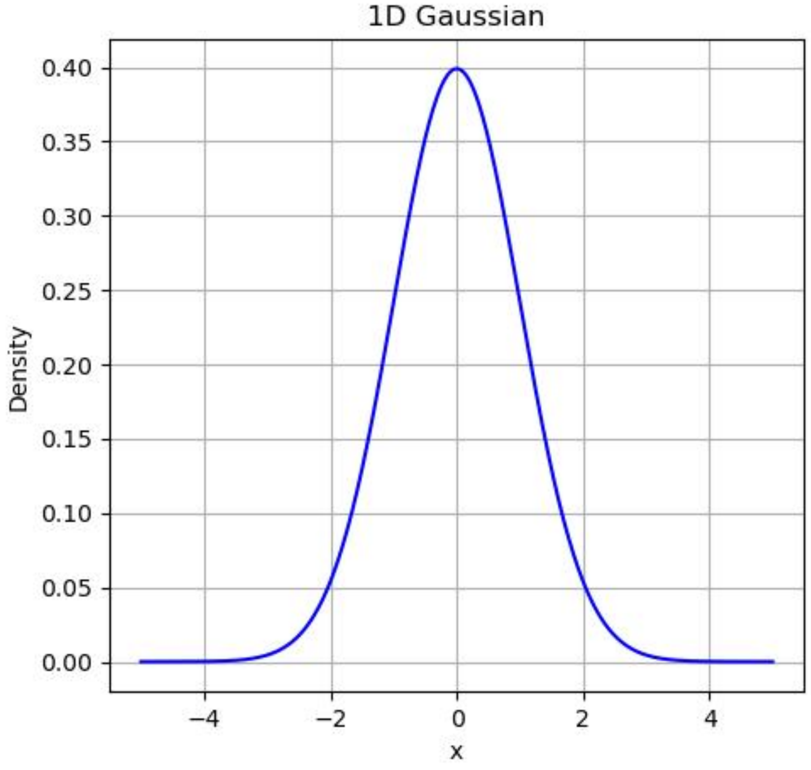
Objects	The Number of NBV Images					
	20		40		60	
	Entropy [2]	Ours	Entropy [2]	Ours	Entropy [2]	Ours
Chair	0.1538	0.0561	0.0892	0.0239	0.0536	0.0173
Hotdog	0.1026	0.0950	0.0746	0.0426	0.0327	0.0258
Mic	0.2521	0.0108	0.0134	0.0080	0.0065	0.0041

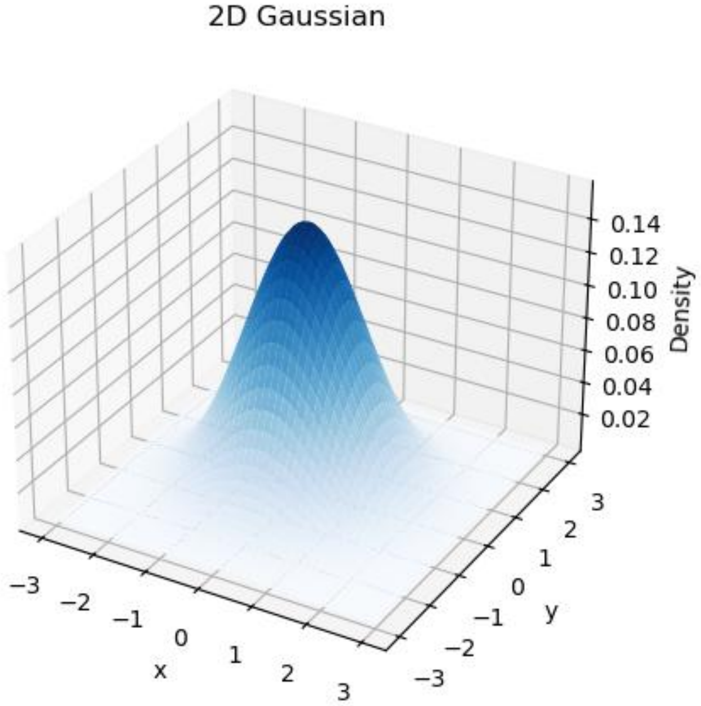
[Table. 1] Chamfer distance (Note: lower (↓) is better.) of the original entropy-based method [2] and our proposed entropy-based methods with distance regularization on three different object datasets

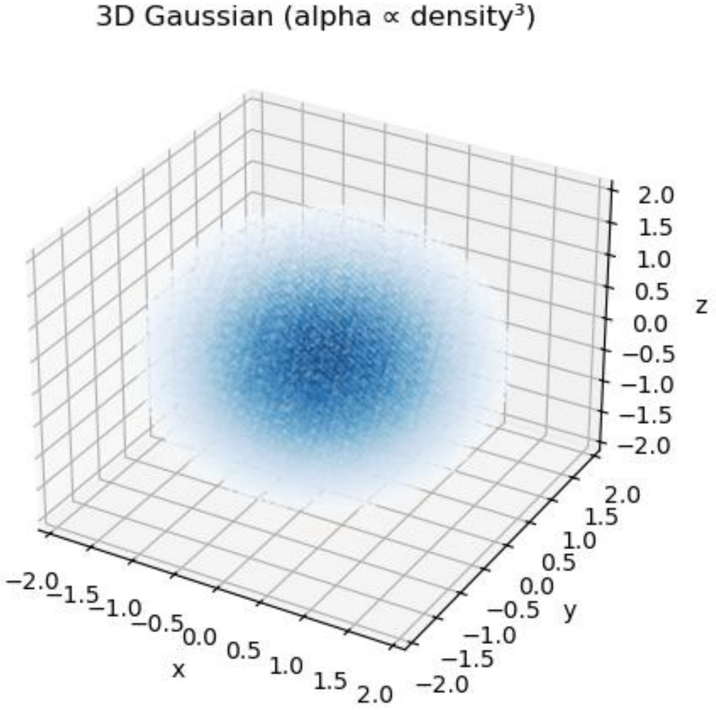
Reference: Choi et al., "Distance and Entropy Based Image Viewpoint Selection for Accurate 3D Reconstruction with NeRF", JKROS, 2014 DOI

3D Gaussian Splatting (3DGS; 2023)

- **3DGS** is a *explicit* 3D representation with **a collection of** [3D Gaussians]() for fast and high-quality rendering.
  - **Gaussians**: $g(\mathbf{x}) = \exp(-\frac{1}{2}\,\mathbf{x}^{\intercal}\Sigma^{-1}\,\mathbf{x})$



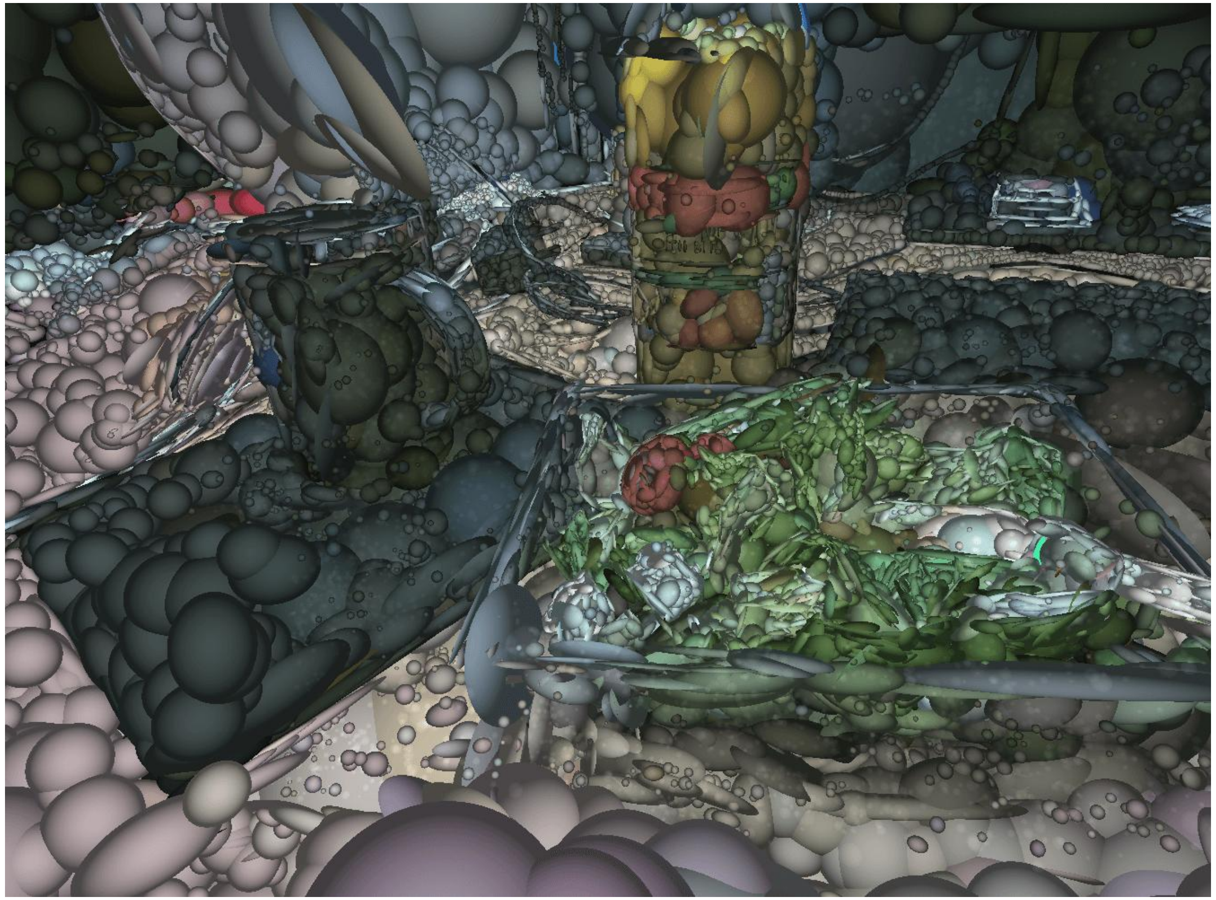




Reference: Kerbl et al., "3D Gaussian Splatting for Real-Time Radiance Field Rendering", SIGGRAPH, 2023 Homepage

3D Gaussian Splatting (3DGS; 2023)

- **3DGS** is a *explicit* 3D representation with **a collection of <u>3D Gaussians</u>** for fast and high-quality rendering.
  - **3D Gaussians** ~ 3D *ellipsoids* (an extension of *point cloud*)



3D Gaussian visualization

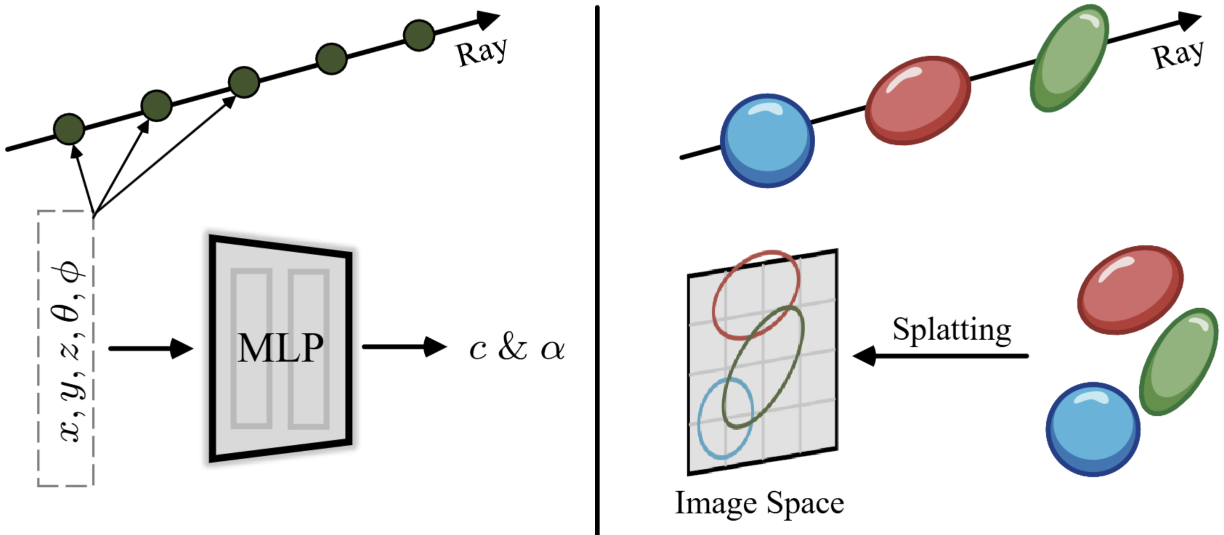
Its rasterized image

Image: Matsuki et al., "Gaussian Splatting SLAM", CVPR, 2024 [Homepage]

3D Gaussian Splatting (3DGS; 2023)

- **3DGS** is a *explicit* 3D representation with **a collection of <u>3D Gaussians</u>** for fast and high-quality rendering.
  - **3D Gaussians**: $g(\mathbf{x}) = \exp(-\frac{1}{2}\, \mathbf{x}^\intercal \Sigma^{-1}\, \mathbf{x})$
    - Mean: 3D position $\mathbf{x} = [x, y, z]^\intercal$
    - Covariance: 3D distribution $\Sigma = RSS^\intercal R^\intercal$ (S: scale (anisotropic), R: rotation matrix)
    - Extra: Color (RGB), opacity (α), (optionally) view-dependent appearance (via spherical harmonics)
  - **Rendering** (Rasterization) = **Sort** (based on depth) + **Projection** (a.k.a. splatting) + (alpha-weighted) **Blending**

$$C(\mathbf{r}) = \sum_{i=1}^{N} \mathbf{c}_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)$$



$$C(\mathbf{r}) = \sum_{i=1}^{N} \mathbf{c}_i g_i \alpha_i \prod_{j=1}^{i-1} (1 - g_i \alpha_i)$$

(a) NeRF

(b) 3D GS

Image: Chen and Wang, "A Survey on 3D Gaussian Splatting", arXiv, 2025 arXiv

3D Gaussian Splatting (3DGS; 2023)

- **3DGS** is a *explicit* 3D representation with **a collection of 3D Gaussians** for fast and high-quality rendering.
  - **Rendering** (→ Operation Flow) = **Sort** (based on depth) + **Projection** (a.k.a. splatting) + (alpha-weighted) **Blending**
  - **Training** (→ Operation Flow + ← Gradient Flow)
    - Loss function: $\mathcal{L} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\mathrm{D-SSIM}}$

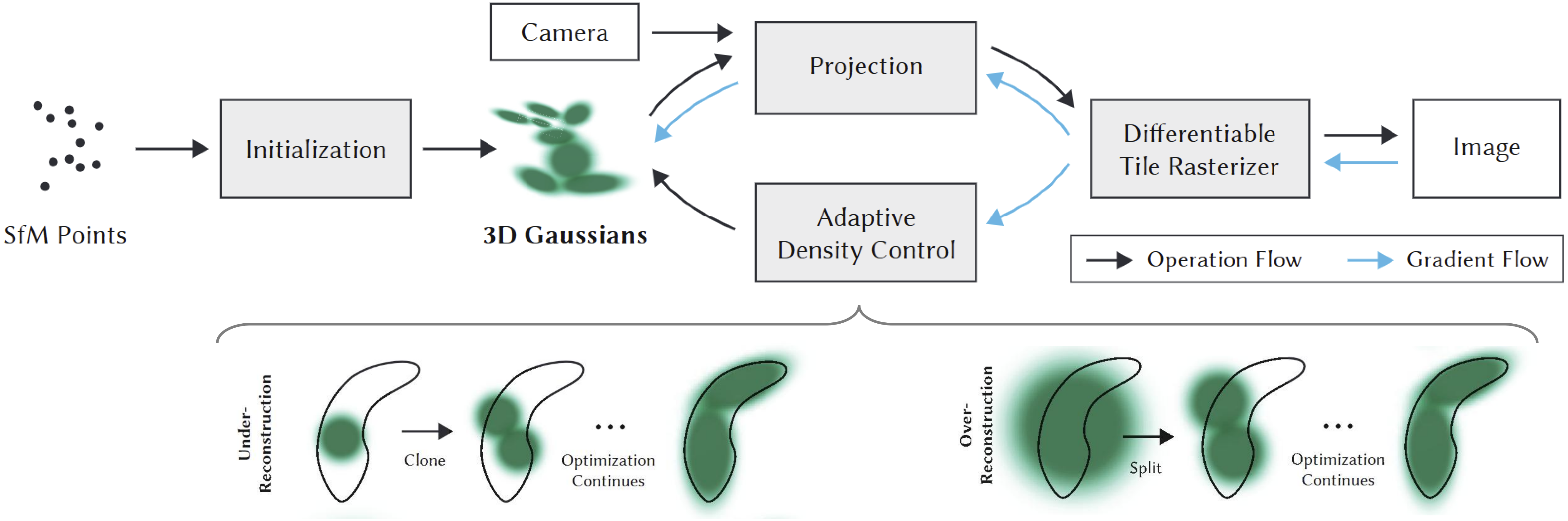


Image: Kerbl et al., "3D Gaussian Splatting for Real-Time Radiance Field Rendering", SIGGRAPH, 2023 [Homepage]

3D Gaussian Splatting (3DGS; 2023)

- **3DGS** is a *explicit* 3D representation with **a collection of 3D Gaussians** for fast and high-quality rendering.



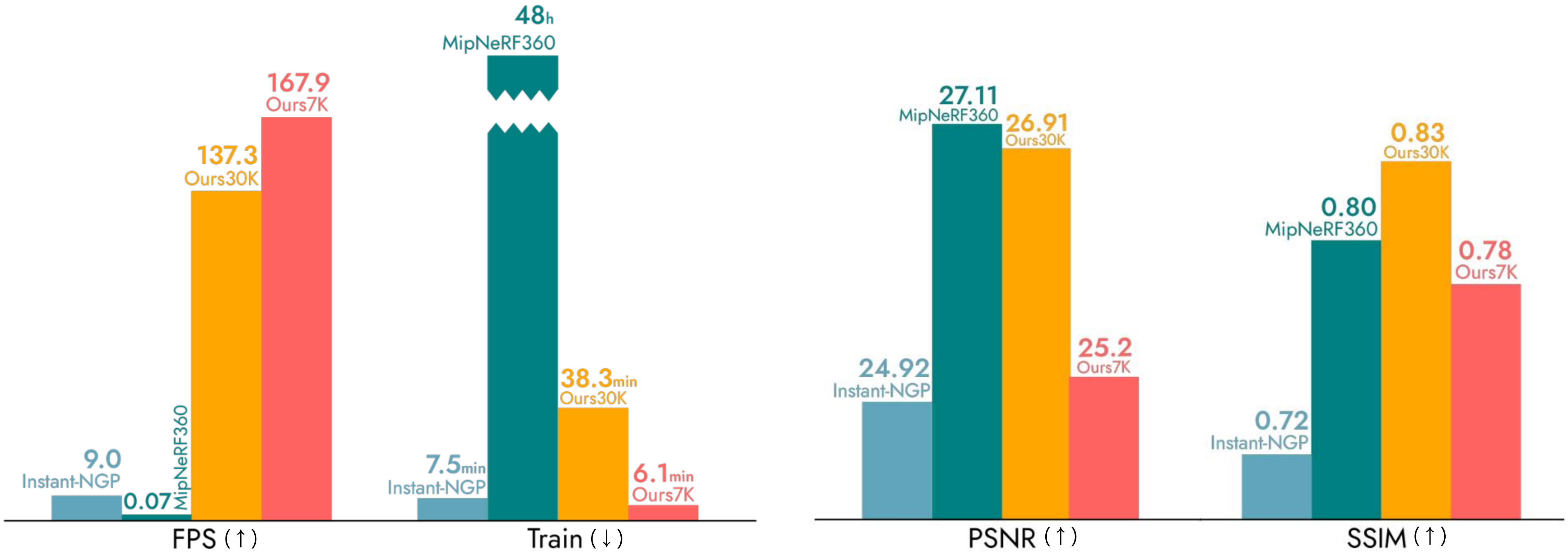
Mip-NeRF360

3DGS

Image: Kerbl et al., “3D Gaussian Splatting for Real-Time Radiance Field Rendering”, SIGGRAPH, 2023 Homepage

3D Gaussian Splatting (3DGS; 2023)

- **3DGS** is a *explicit* 3D representation with **a collection of 3D Gaussians** for fast and high-quality rendering.
  - NeRF suffers from slow training and rendering.
  - Evaluation @ Full MipNeRF360 Dataset + 2 Tanks and Temples + 2 Deep Blending



  - Memory usages (↓): Instant-NGP (15-50MB), MipNeRF360 (8.6MB), 3DGS (350-700MB @ 3-6M of Gaussians)

Image: Kerbl et al., "3D Gaussian Splatting for Real-Time Radiance Field Rendering", SIGGRAPH, 2023 Homepage

Video: Polycam

3D Gaussian Splatting (2023)

- Applications: Real-time 3D engines, 3D capture tools, VFX, …
  - Unreal Plugin: XVERSE 3D-GS UE Plugin
  - Unity Plugin: Gaussian Splatting Playground in Unity, SplatVFX
  - Polycam: Gaussian Splat Tool (community works)

3D Gaussian Splatting (2023)

- Applications: SfM, Visual SLAM/odometry, …
  – SfM, Visual SLAM/odometry: COLMAP-Free 3DGS, Gaussian Splatting SLAM, GS-SLAM

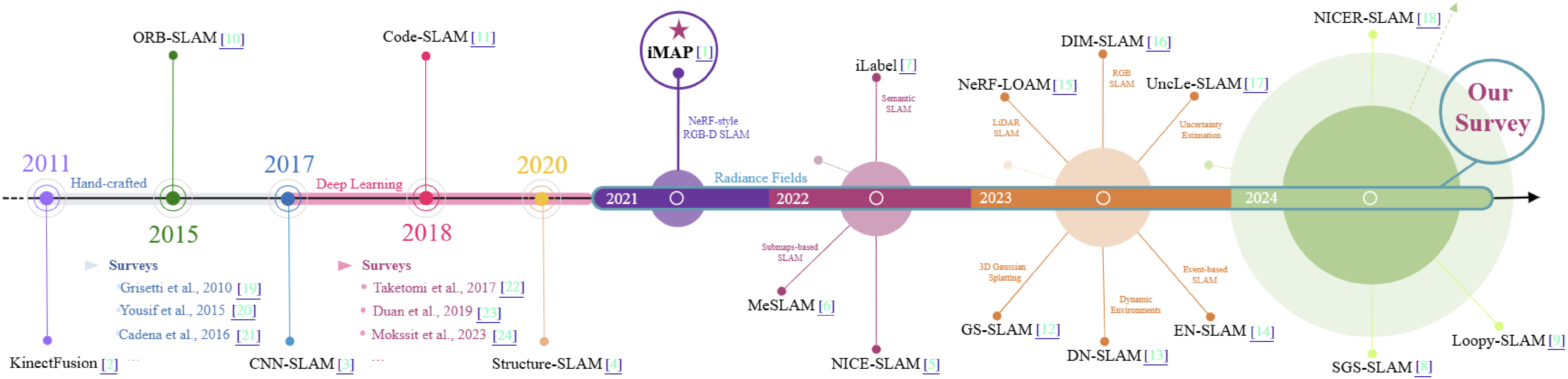


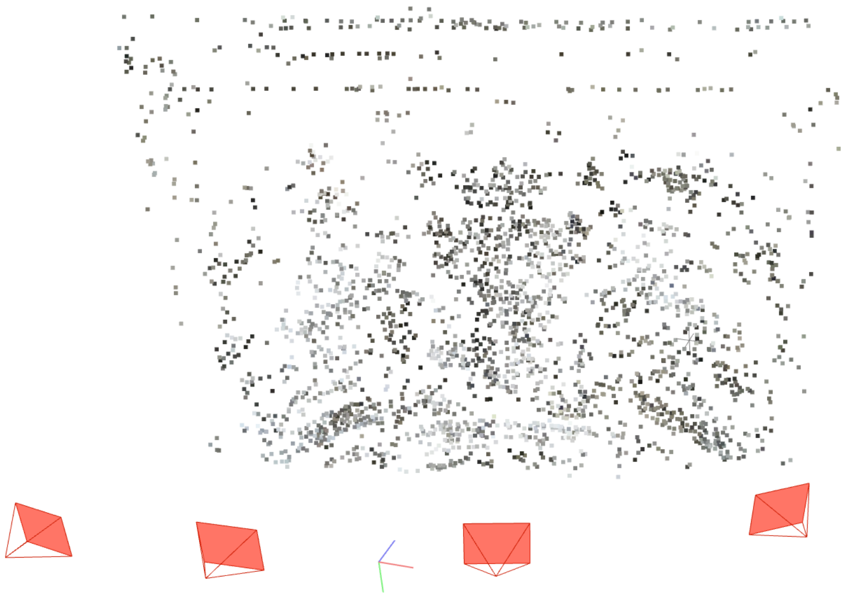
Fig. 1: **Timeline SLAM Evolution.** This timeline shows the evolution from hand-crafted to deep learning SLAM, with key surveys marking both periods. A significant shift occurs in 2021 with iMap [1], introducing radiance-field-based approaches. Circle sizes on the right indicate yearly publication volumes, with 2024's outer circle projecting increased interest in NeRF and 3DGS-based SLAM.

Image: Tosi et al., "How NeRFs and 3D Gaussian Splatting are Reshaping SLAM: a Survey", arXiv, 2024 arXiv

3D Gaussian Splatting (2023): Getting Started with gsplat

- Example) gsplat with the relief dataset
  - Reconstruction results

Sparse Reconstruction (SfM) → **Dense Reconstruction (MVS)** or **3D Gaussian Reconstruction**



Viewer: Viser

# of points: 2,889

# of points: 336,223

# of Gaussians: 221,767

Summary

- **3D Representations**: How to represent 3D scenes and models
  - Classical representations: Voxel, point cloud, polygon mesh, signed distance field (SDF)
  - Neural Radiance Field (**NeRF**): 11 fully-connected (shortly FC) layers
    - Key idea: Neural volumetric rendering, positional encoding, hierarchical volume sampling

    → High-quality view synthesis for *continuous* 3D scenes, but *too slow* rendering and training time
  - 3D Gaussian Splatting (**3DGS**): A collection of 3D Gaussians
    - Key idea: Volumetric splatting, (tricky) adaptive density control

    → Faster and more high-quality, but *more memory* consumption
    - Applications: Real-time 3D engines, 3D capture tools, VFX, SfM, visual SLAM/odometry, …