Penn Engineering
University of Pennsylvania

GMLR

CIS 6280 · FALL 2026

WORLD MODELS

LECTURE 15

3D and 4D Representations

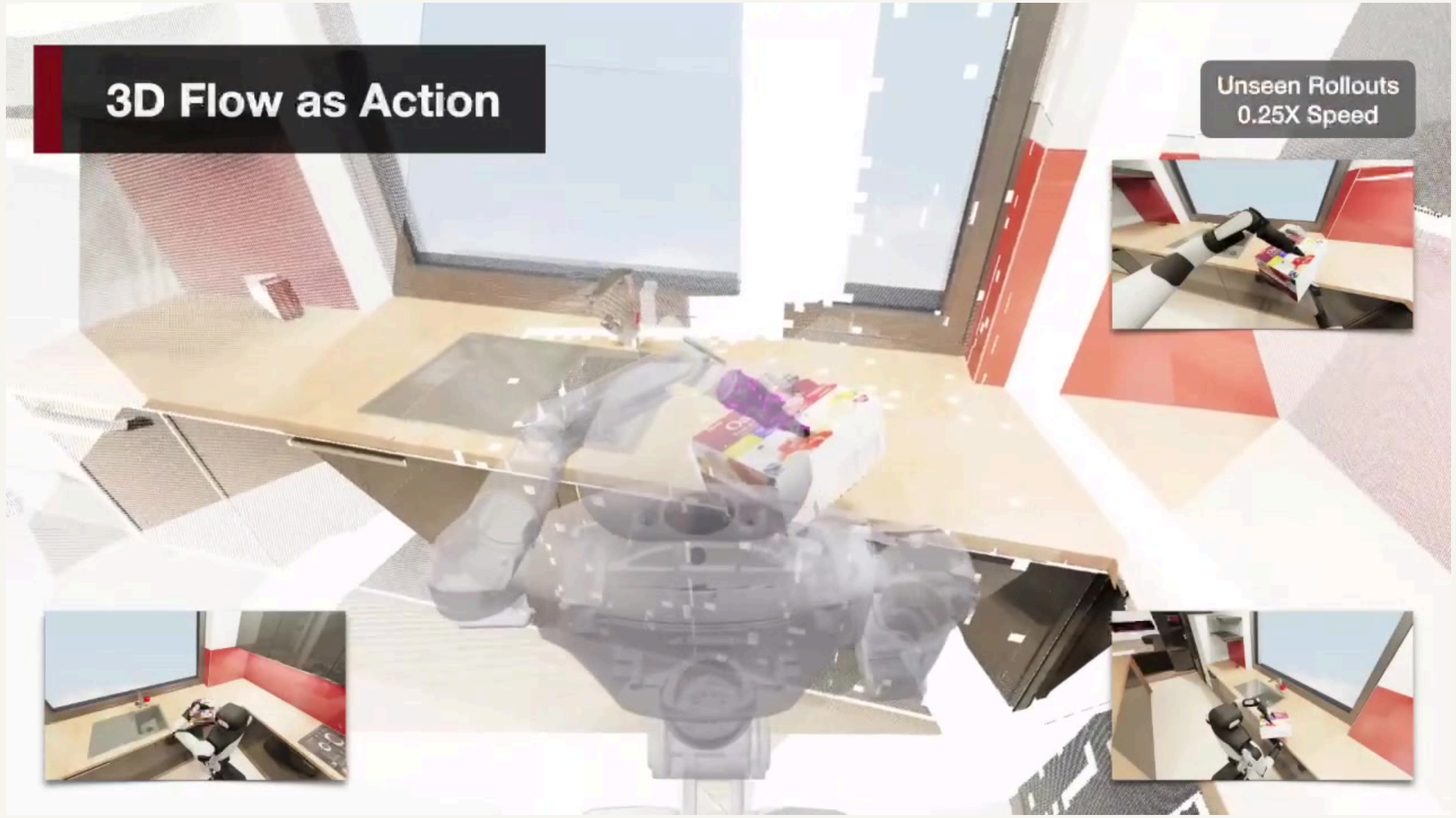
Jiatao Gu

University of Pennsylvania · October 2026

Course announcements

Assignment 3	Release delayed to November 2 Due: November 18 (unchanged)
Final project	Data released Instructions and updates on Canvas
Office hours	Thursday, 4:30–5:30 PM AGH 423

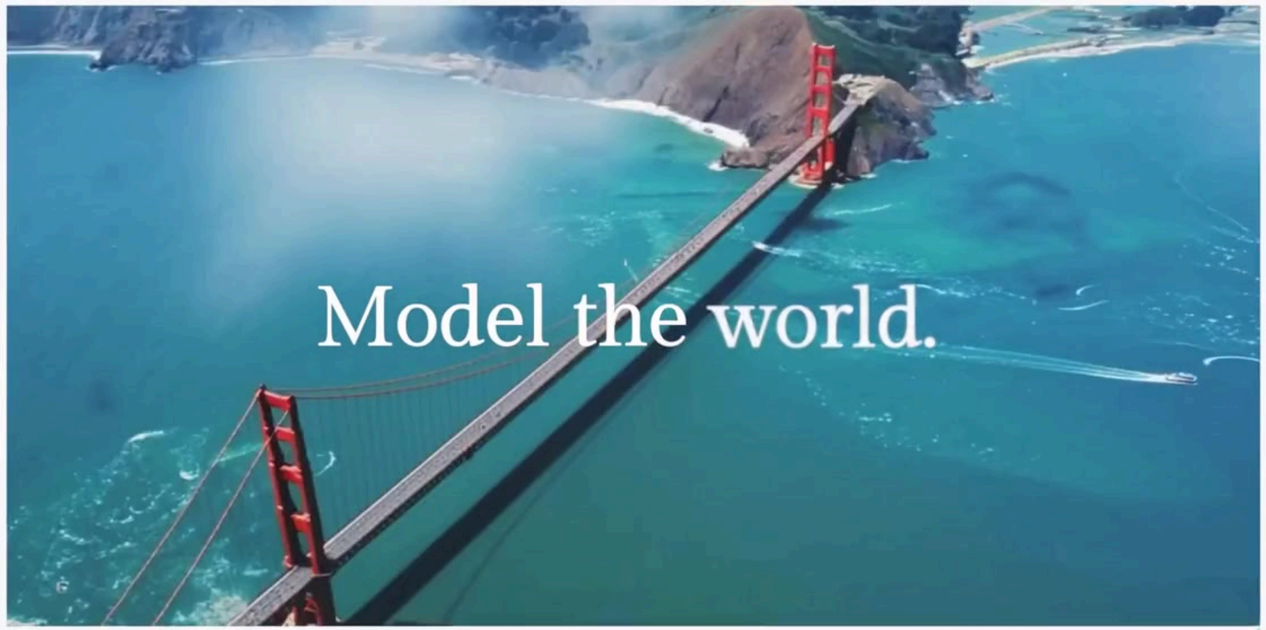
Predicting an interaction



PointWorld · RGB-D observations and proposed robot motion condition the predicted 3D response.

What will move if the robot takes this action?

Exploring a world

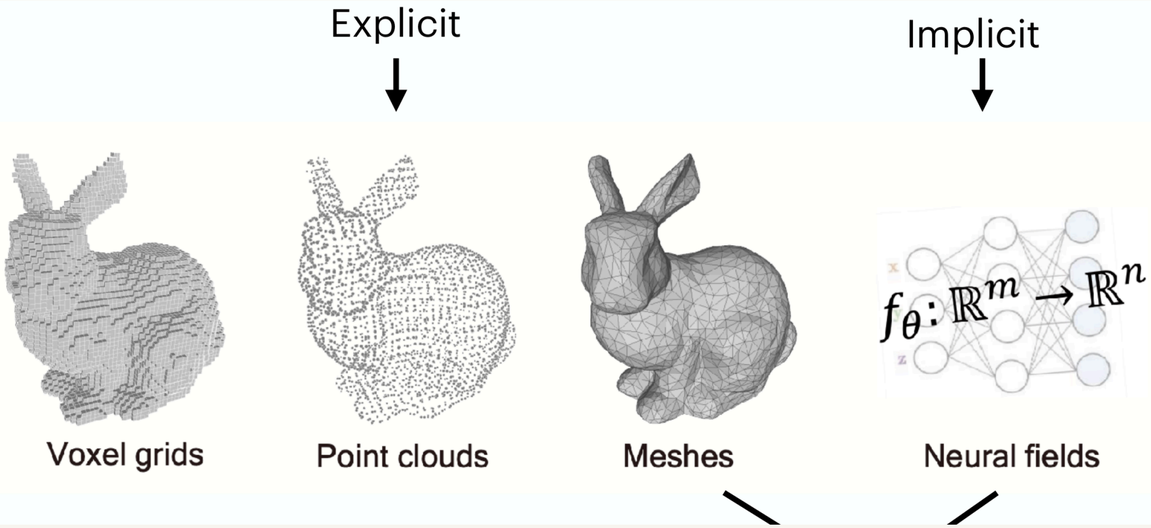


Atlas · World Labs official demonstration of camera-controlled spatial generation.

What should the same world look like from another camera?

World Labs, Atlas (September 2026)

3D representations



Several representations of the same kind of 3D object. Adapted from last year's course.

What information does the model store?

Geometry

Where are the surfaces and empty regions?

Appearance

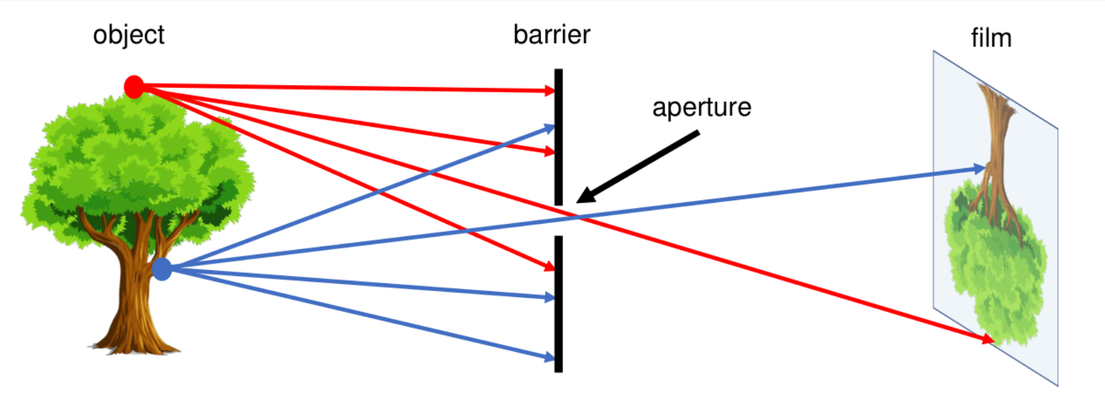
What would a camera observe?

Structure

Which elements belong together?

Gu, CIS 7000 · 3D Generative Models (2025) · Mescheder et al., Occupancy Networks (2019)

A camera connects a 3D scene to a 2D image



Pinhole model: light passes through a small opening and reaches the image plane.

The opening is the **camera center**.
The sensor is the **image plane**.

A camera model tells us where a 3D point appears in an image.

3D → 2D

Follow the line from a scene point through the camera center. It lands at one image location.

2D → 3D

A pixel gives a viewing ray. We still need depth to locate a point along that ray.

We need this link to recover geometry from images and render a 3D world from a chosen view.

Camera intrinsics K: viewing rays to pixels

Camera point (X_c, Y_c, Z_c) → pixel (u, v)

X_c, Y_c across the view; Z_c is forward depth.

$$u = f_x \frac{X_c}{Z_c} + c_x$$

$$v = f_y \frac{Y_c}{Z_c} + c_y$$

Divide by depth → scale → shift

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Perspective: divide by depth

X_c/Z_c and Y_c/Z_c describe the viewing direction. Greater depth makes the same object appear smaller.

Focal lengths: f_x, f_y

Set the image scale in pixels. Larger values give a more zoomed-in view.

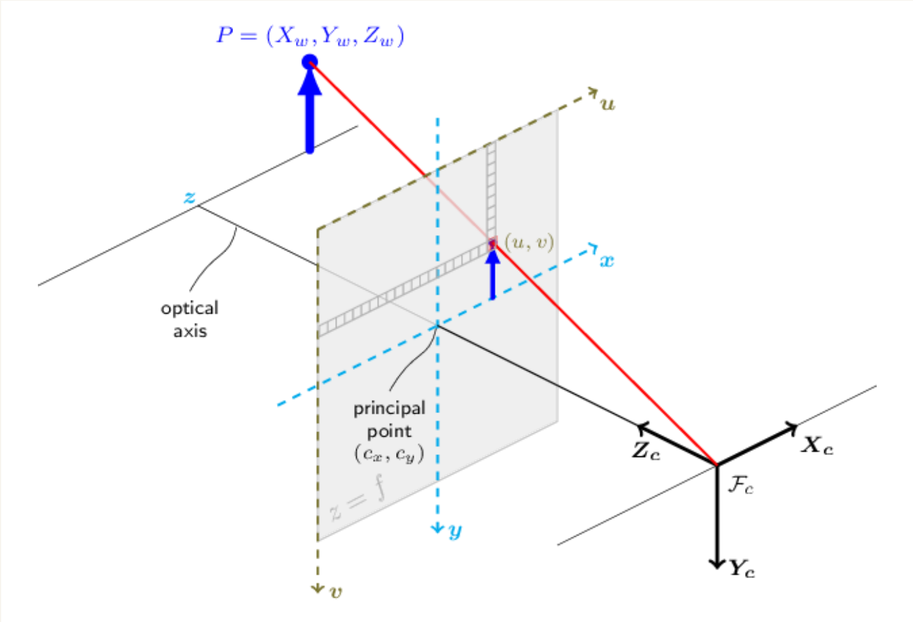
Principal point: (c_x, c_y)

Where the forward-looking axis meets the image, usually near the image center.

Ideal pinhole camera, perpendicular pixel axes, no lens distortion.

K packages image scale and center. It does not tell us where the camera is.

Camera pose and the projection matrix



World coordinates stay fixed; camera coordinates move with the camera.

Pose: where the camera is and which way it faces.

$$\mathbf{X}_c = R\mathbf{X}_w + \mathbf{t}$$

$\mathbf{X}_w$: world coordinates. $\mathbf{X}_c$: camera coordinates.
R rotates the axes; $\mathbf{t}$ shifts the origin.

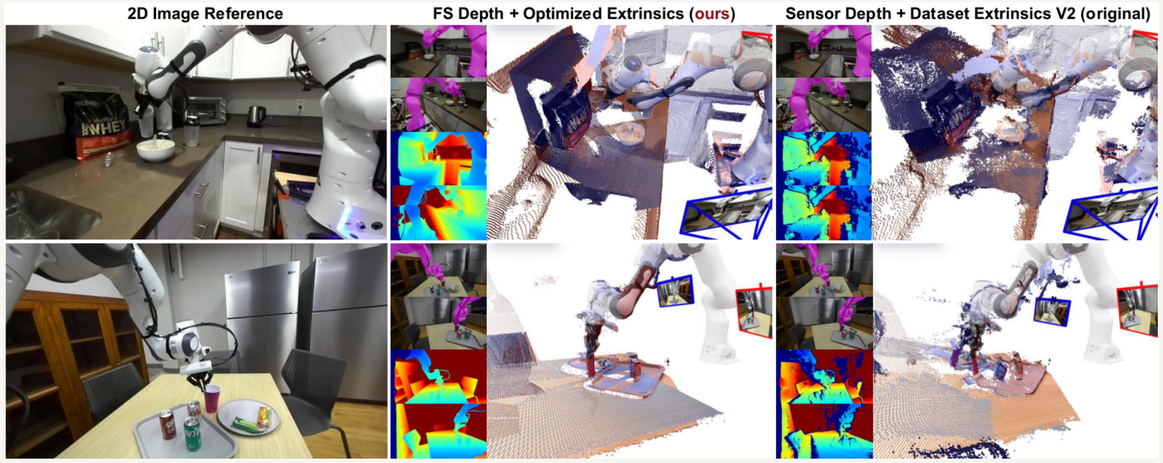
$$P_{\text{cam}} = K[R \mid \mathbf{t}]$$

Apply P_{cam} to $[X_w, Y_w, Z_w, 1]^\top$.
Divide the first two outputs by the third to get (u, v).

The final 1 lets the matrix include translation.

Read $P_{\text{cam}} = K[R \mid \mathbf{t}]$ right to left: world → camera coordinates → image projection.

Depth maps and pointmaps



A view, its depth, and the reconstructed points. PointWorld data annotation example.

Depth map: one forward depth $d = Z_c$ per pixel.

Pointmap: one 3D position $\mathbf{X}_c(u, v)$ per pixel.

$$\mathbf{X}_c(u, v) = d(u, v)\, K^{-1}[u, v, 1]^\top$$

K^{-1} turns the pixel into a viewing direction.
d places the point at the measured depth.

These points are in the camera frame. Use the camera pose to express them in a shared world frame.

Wang et al., DUSt3R (2024) · Huang et al., PointWorld (2026)

Point clouds



Points sample visible surfaces. Each point can carry additional features.

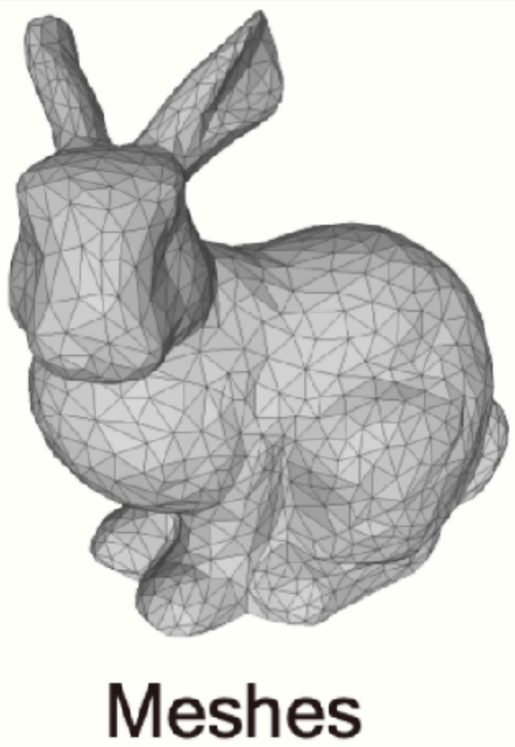
$$\mathcal{P} = \{(\mathbf{x}_i, \mathbf{f}_i)\}_{i=1}^{N}$$

Position $\mathbf{x}_i \in \mathbb{R}^3$ and optional features $\mathbf{f}_i$.

Color and learned visual features can travel with the point.

A single RGB-D view usually reveals only part of the scene.

Meshes



A surface mesh stores vertices and their connectivity.

Vertices

Positions in 3D.

Faces

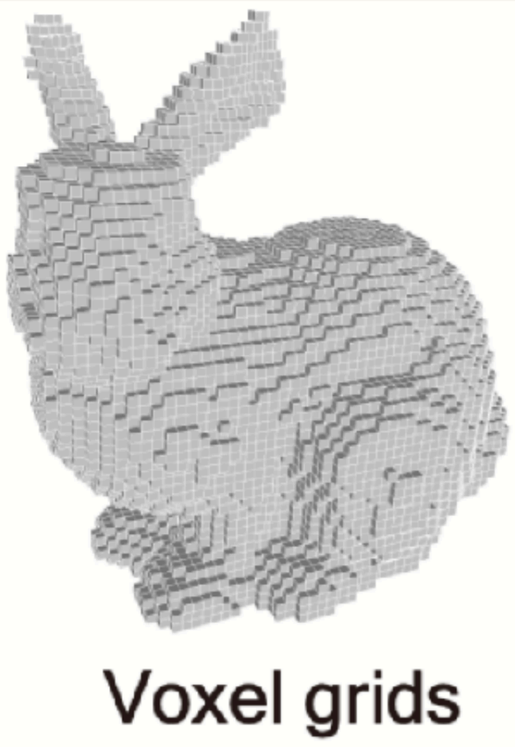
Which vertices form a surface.

Volume mesh

Interior elements can connect a soft solid.

For a deforming mesh, positions change while connectivity provides structure.

Voxel grids



A regular spatial grid discretizes a 3D region.

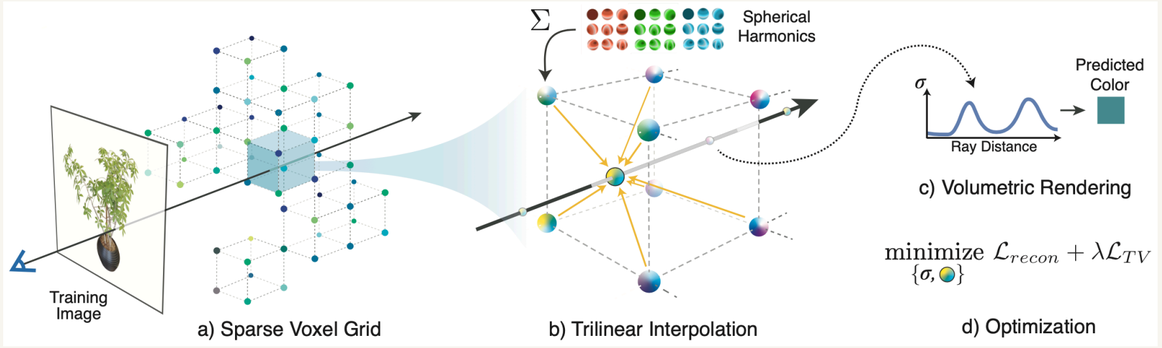
$$V \in \mathbb{R}^{N_x \times N_y \times N_z \times C}$$

Three spatial axes and C attributes per cell.

A voxel may store occupancy, distance, color, density, or learned features.

The grid layout and the stored attribute are separate choices.

Volumes and fields



Plenoxels stores a radiance field in a sparse voxel structure.

Domain

A region of 3D space.

Attribute

A value or vector at each location.

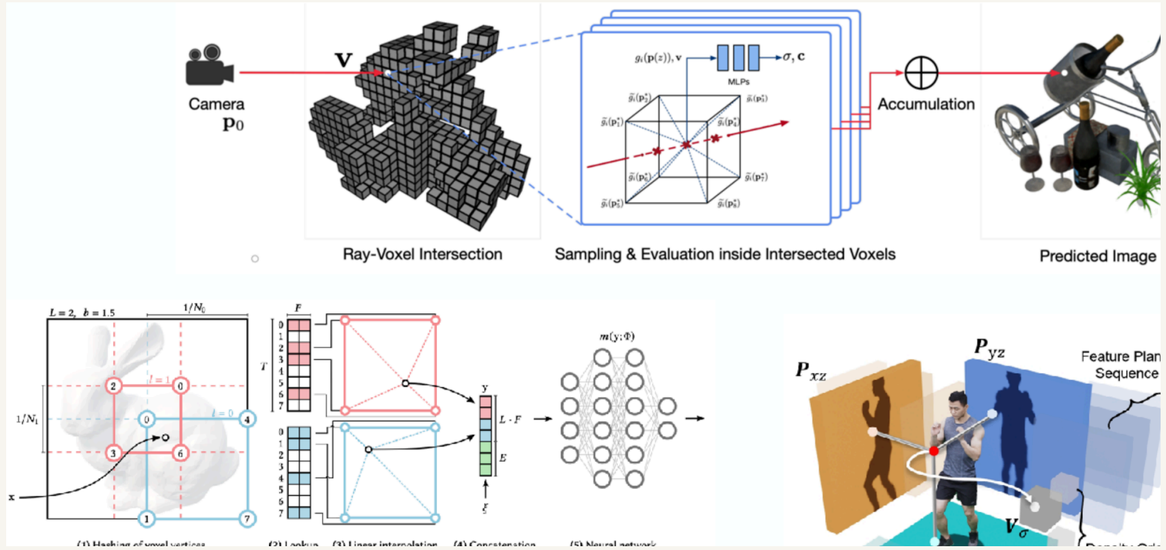
Storage

A grid, a neural function, or a hybrid representation.

A volumetric field can be continuous even when its storage is discrete.

Fridovich-Keil et al., Plenoxels (2022) · Mildenhall et al., NeRF (2020)

Dense and sparse volumes



Dense and sparse structures allocate computation differently. Adapted from the previous course.

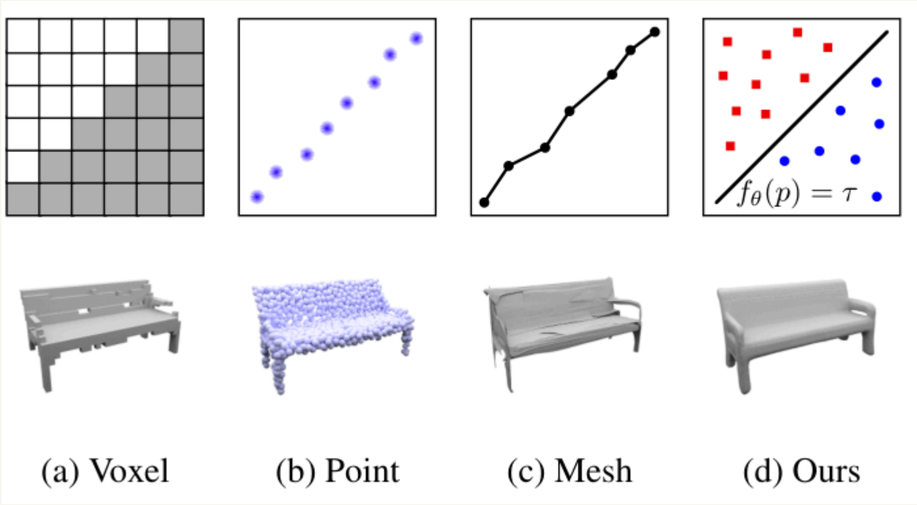
Dense grids allocate every cell.

Sparse grids allocate selected regions.

Octrees and multiresolution features change the balance between detail, memory, and query cost.

An unallocated cell is not automatically a known empty region.

Occupancy and distance fields



A continuous field defines a surface through its queried values.

Occupancy

Is a location inside the shape?

Signed distance

How far is it from the surface, with a sign convention?

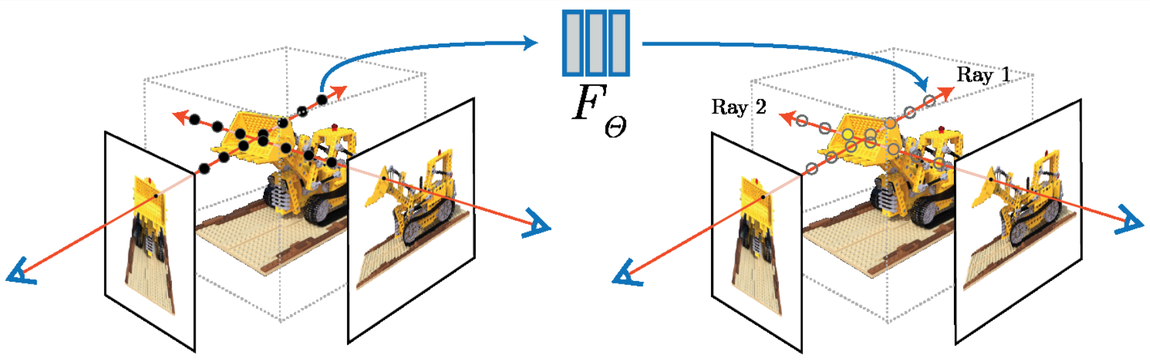
Radiance density

How strongly does a ray attenuate here?

These quantities impose different meanings and training targets.

Mescheder et al., Occupancy Networks (2019) · Mildenhall et al., NeRF (2020)

Neural radiance fields



NeRF queries a continuous scene representation along camera rays.

$$f_\theta(\mathbf{x}, \mathbf{d}) = (\mathbf{c}, \sigma)$$

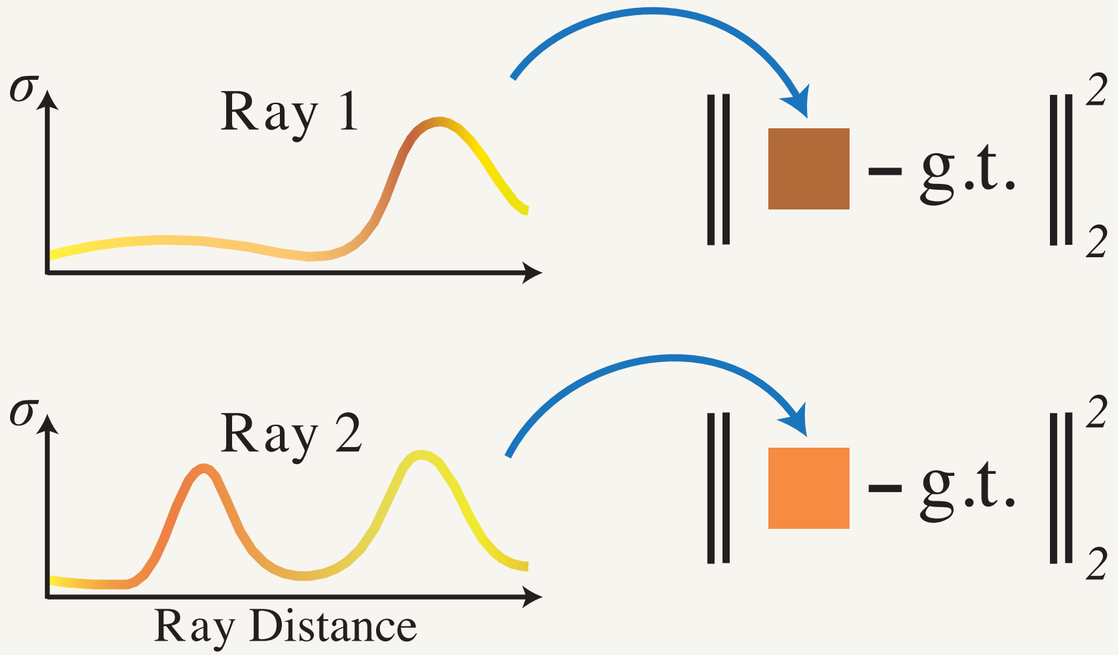
$\mathbf{x}$: location, $\mathbf{d}$: viewing direction.

$\mathbf{c}$: color, σ: optical density.

The scene is represented by a function that can be rendered from new cameras.

Mildenhall et al., NeRF (2020)

Volume rendering



$$\widehat{\mathbf{C}}(\mathbf{r}) = \sum_i T_i \alpha_i \mathbf{c}_i$$

$$\alpha_i = 1 - e^{-\sigma_i \Delta_i}$$

T_i: transmittance before sample i.

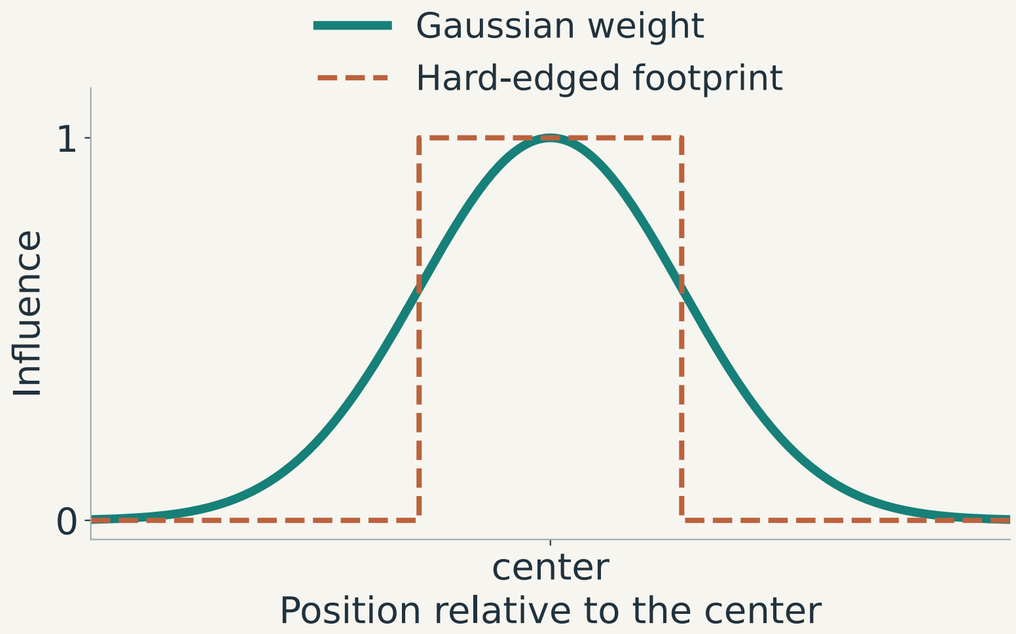
α_i: segment opacity. Δ_i: sample spacing.

Samples along a ray contribute color according to visibility and density.

The renderer accounts for which samples are visible along the ray.

Mildenhall et al., NeRF (2020)

Why Gaussians?



Cross-section through a footprint.

Spatial extent

A point needs a footprint to cover image pixels.

Smooth falloff

Small changes in position or size change its contribution smoothly.

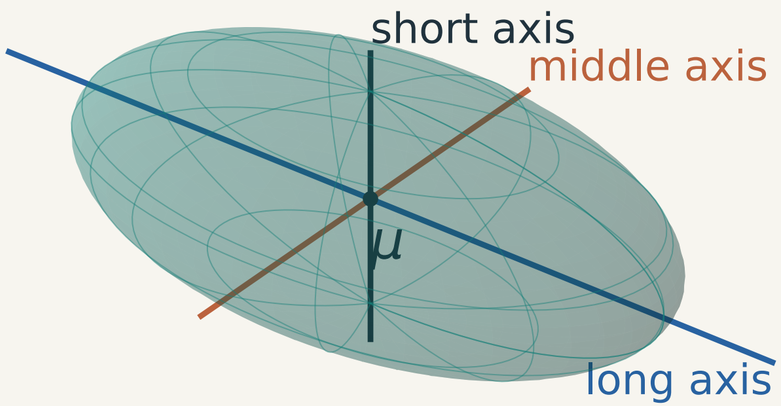
Flexible shape

Stretch and rotate the Gaussian to fit local scene structure.

Smooth weights, flexible shape, and a simple form under local camera projection.

Kerbl et al., 3D Gaussian Splatting (2023)

One 3D Gaussian



$\boldsymbol{\mu}$ **Center**: where the element sits.

$\boldsymbol{\Sigma}$ **Shape**: its three axis lengths and orientation.

o **Opacity**: how strongly it contributes.

$\mathbf{c}(\mathbf{d})$ **Color**: may depend on viewing direction $\mathbf{d}$.

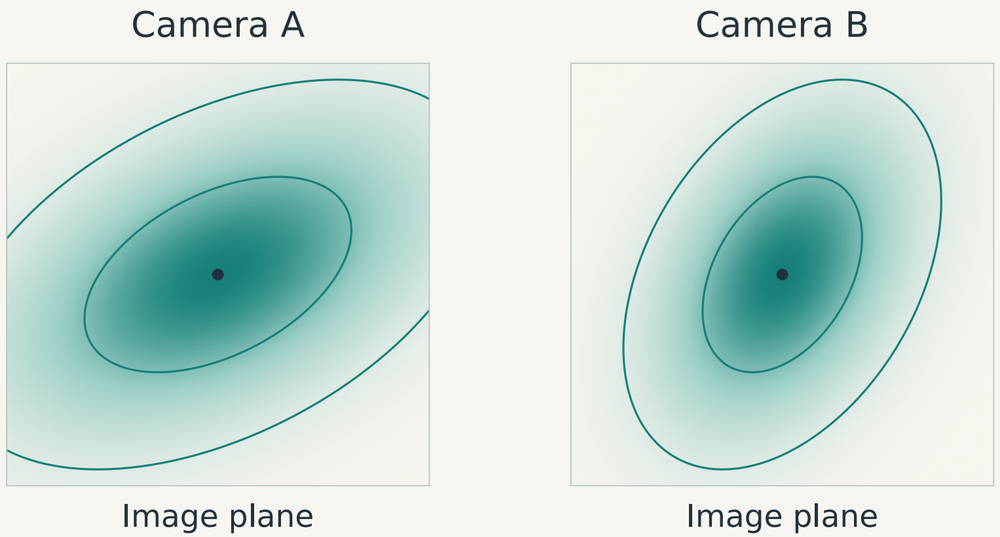
A constant-weight surface. The Gaussian extends smoothly beyond it.

$$G(\mathbf{x}) = \exp\left[-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]$$

$\mathbf{x}$ is a 3D query position. G is strongest at the center and decreases with distance in the ellipsoid's coordinates.

Kerbl et al., 3D Gaussian Splatting (2023)

Projecting a Gaussian into the image



The same 3D Gaussian produces different 2D footprints in different views.

Project the center with the camera model we already introduced.

$$\boldsymbol{\mu}_{\text{px}} = \pi_K(R\boldsymbol{\mu} + \mathbf{t})$$

Transform its spatial extent into a pixel-space ellipse.

$$\Sigma_{\text{px}} \approx JR\Sigma R^\top J^\top$$

$R, \mathbf{t}$: world-to-camera pose. π_K: perspective division and intrinsics.

J: the local linear approximation of π_K at the Gaussian center.

An affine projection preserves Gaussian form. Perspective uses this local approximation.

Kerbl et al., 3D Gaussian Splatting (2023)

Color, opacity, and occlusion

For a pixel $\mathbf{p}$, Gaussian i contributes opacity $\alpha_i(\mathbf{p}) = o_i \, G_i^{2D}(\mathbf{p})$.

Front Gaussian

$$\alpha_1 \mathbf{c}_1$$

Contributes first.

Next Gaussian

$$(1-\alpha_1)\alpha_2 \mathbf{c}_2$$

Receives the visibility left by the first.

Further back

$$(1-\alpha_1)(1-\alpha_2)\alpha_3 \mathbf{c}_3$$

Receives what remains after both.

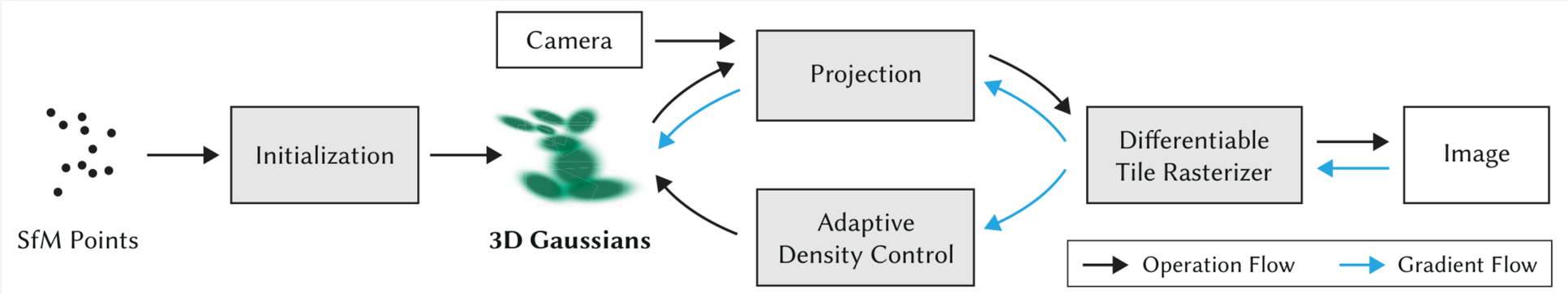
$$\widehat{\mathbf{C}}(\mathbf{p}) = \sum_i T_i(\mathbf{p}) \, \alpha_i(\mathbf{p}) \, \mathbf{c}_i, \qquad T_i(\mathbf{p}) = \prod_{j<i} \big(1 - \alpha_j(\mathbf{p})\big)$$

G_i^{2D}: projected Gaussian weight. T_i: transmittance before element i. Contributions follow front-to-back depth order.

The same compositing idea as volume rendering, now applied to projected Gaussians.

Kerbl et al., 3D Gaussian Splatting (2023)

Fitting Gaussians to multiple views



Original 3DGS pipeline: camera calibration, projection, rendering, and image-based optimization.

Input

Photos of one scene and their calibrated cameras.

Initialization

Sparse 3D points from structure from motion (SfM).

Original image loss: pixel L1 error + a structural similarity term (SSIM).

Image supervision

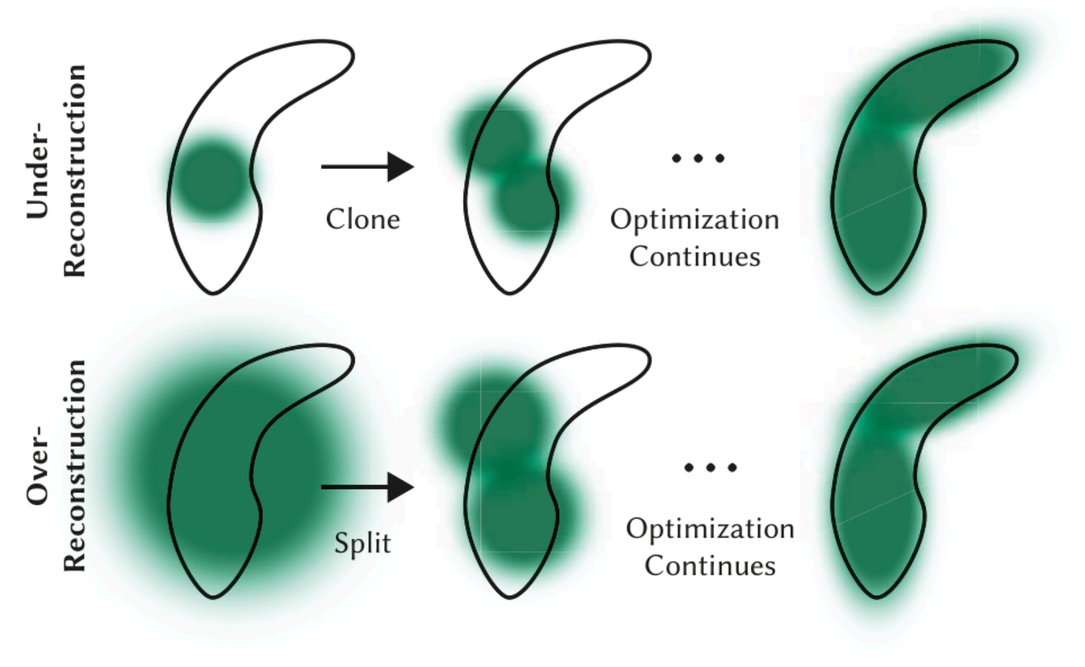
Render a training view and compare it with the observed photo.

Gradient updates

Adjust centers, shapes, opacities, and color coefficients.

Kerbl et al., 3D Gaussian Splatting (2023)

Adding detail: clone, split, and prune



Original 3DGS density control. The black outline represents scene detail to reconstruct.

Clone

Copy small Gaussians where the current set needs more coverage.

Split

Replace large Gaussians with smaller ones where finer detail is needed.

Prune

Remove nearly transparent elements and problematic oversized elements.

Training changes both the Gaussian parameters and the number of elements.

Kerbl et al., 3D Gaussian Splatting (2023)

NeRF and Gaussian splatting

	Original NeRF	3D Gaussian splatting
Scene storage	Neural function of position and direction	Explicit centers, shapes, opacity, and appearance
Rendering	Query points along each ray	Project Gaussians and composite their footprints
Image supervision	Differentiate through volume rendering	Differentiate through the splat renderer
Where computation goes	Many field evaluations along rays	Visible splats, image tiles, and depth sorting

Practical limits: many elements consume memory. Large or overlapping splats can introduce projection and visibility artifacts.

Both represent how an observed scene looks from new views. Predicting its dynamics requires an additional model.

Kerbl et al., 3D Gaussian Splatting (2023)

What the representation stores

Representation	Stored information	Typical query
Points / mesh	Samples; optional connectivity	Where is the surface?
Voxel / volume	Spatial attributes or features	What is at this location?
Radiance field / splats	Geometry and appearance for rendering	What would this camera see?

Hybrid representations combine these choices. Learned features can live on grids, points, or compact tokens.

Mescheder et al., Occupancy Networks (2019) · Müller et al., Instant-NGP (2022) · Kerbl et al., 3D Gaussian Splatting (2023)

4D representations



An observed dynamic video and its reconstructed geometry and point motion.

A 3D world changes over time.

We need geometry, correspondence, and a reference frame.

Feng et al., St4RTrack (2025)

A sequence of point clouds



Observed geometry at one time



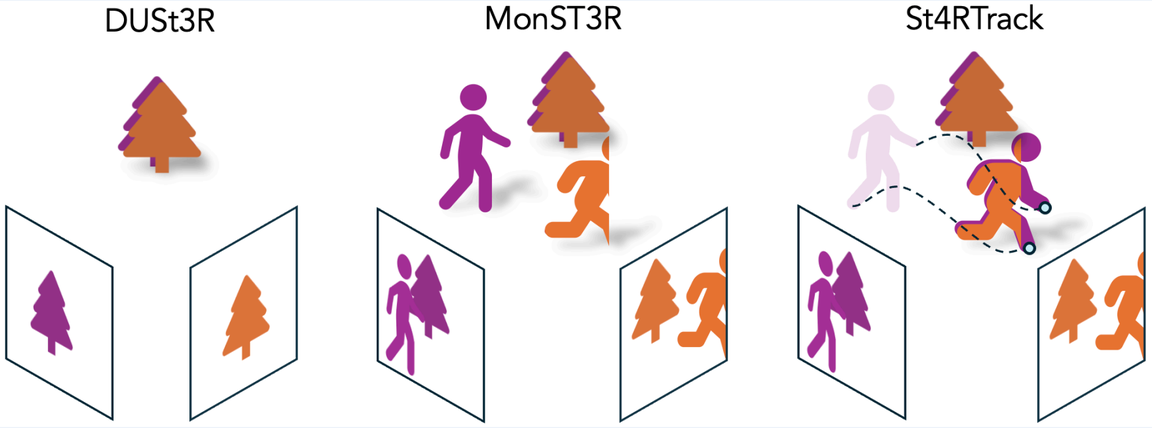
Geometry later in the video



Geometry at another time

Changing point sets do not automatically preserve point identity.

Scene flow



3D motion and changing image views require separate geometric bookkeeping.

$$\Delta \mathbf{X}_i = \mathbf{X}_i(t_b) - \mathbf{X}_i(t_a)$$

A 3D displacement for the same point, in one reference frame.

Optical flow instead describes displacement in an image.

Always specify the two times and the coordinate frame.

Feng et al., St4RTrack (2025) · Huang et al., PointWorld (2026)

Point trajectories



Persistent point trajectories describe motion through an observed scene.

$$\mathcal{T}_i = \{\mathbf{X}_i(t) : t \in \mathcal{Q}\}$$

Point i keeps its identity across time.

$\mathcal{Q}$ is the set of requested times.

The representation records where the same point goes.

Point identity



A point is selected in a source frame and followed through the scene.

Source pixel

Selects a physical surface point.

Point identity

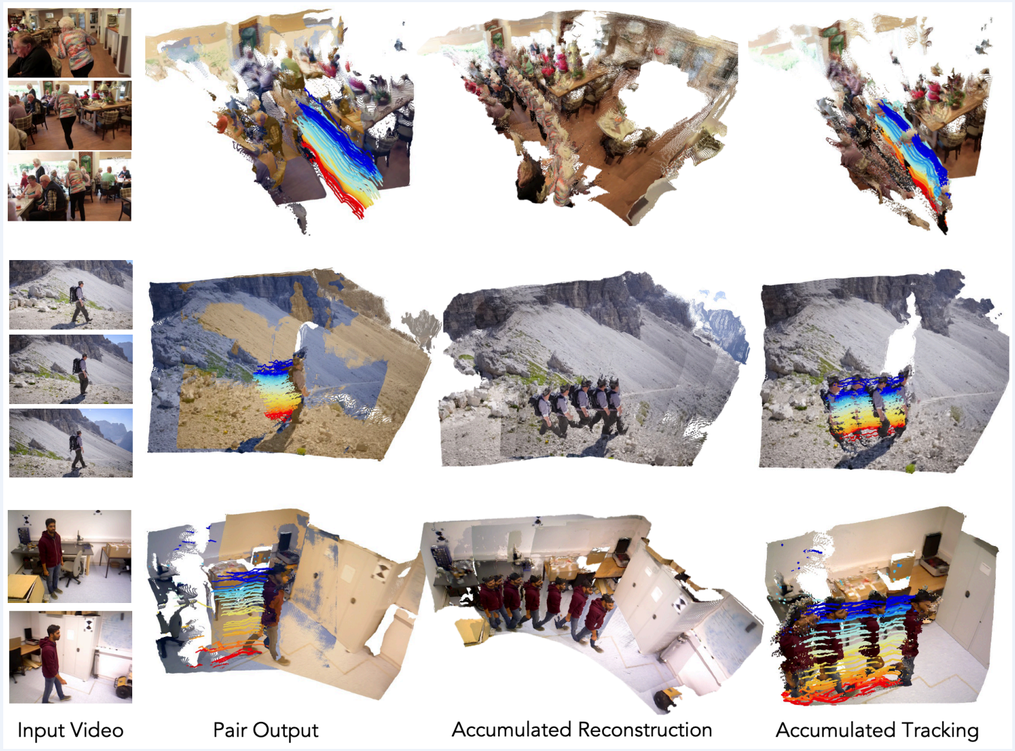
Remains associated with that point.

Later pixel

Changes as the point and the camera move.

An image index and a persistent point ID serve different purposes.

Visibility and confidence



Observed correspondences can be interrupted by occlusion and view changes.

Position

Where the point is estimated to be.

Visibility

Whether the point is observable in this view.

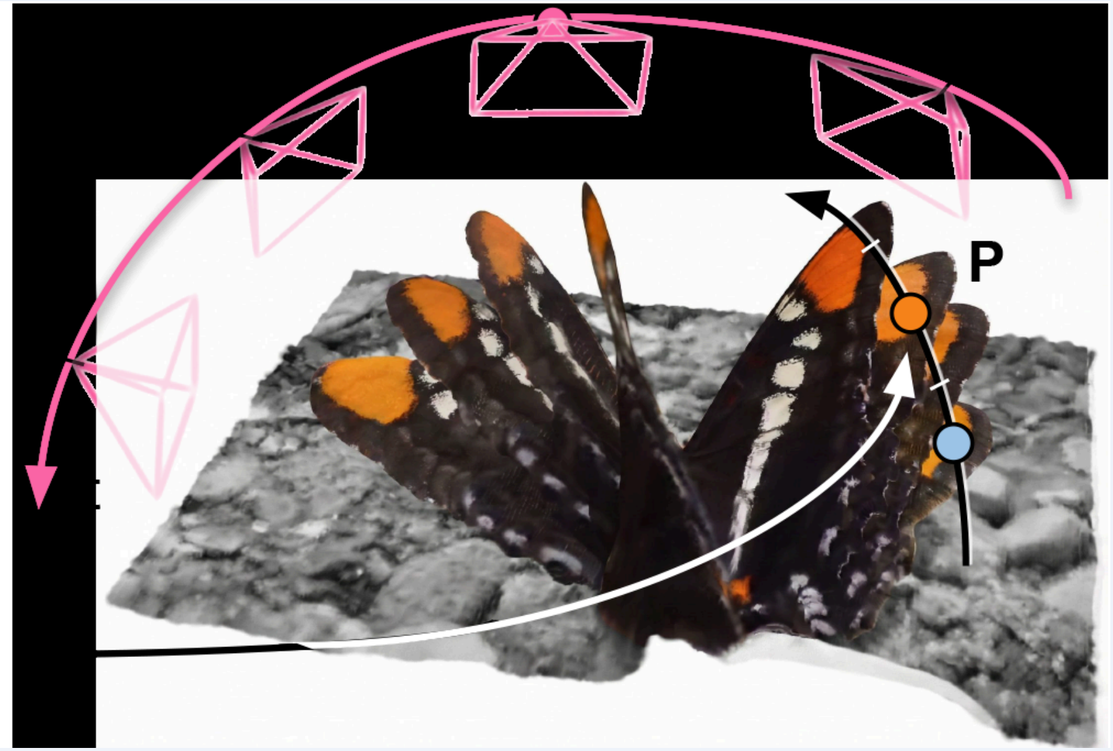
Confidence

How reliable the estimate is.

An inferred position behind an occluder is different from a measured position.

Xiao et al., SpatialTrackerV2 (2025) · Feng et al., St4RTrack (2025)

Camera and world coordinates



The same point can be reported in different camera coordinate frames.

Camera coordinates change as the camera moves.
A fixed frame lets us compare scene motion.

$$\mathbf{X}_t^w = R_t^\top (\mathbf{X}_t^c - \mathbf{t}_t)$$

Undo the translation, then undo the rotation.

$R_t, \mathbf{t}_t$ still map world → camera.
Here we apply the inverse to recover world coordinates.

The first camera can define the reference; it does not guarantee absolute scale.

Lifting tracks into 3D

2D track

Pixel (u_t, v_t)

→

Depth

Optical-axis d_t

→

Camera frame

$\mathbf{X}_t^c$

→

Fixed frame

$\mathbf{X}_t^w$

$$\mathbf{X}_t^w = R_t^\top \left[d_t K_t^{-1} (u_t, v_t, 1)^\top - \mathbf{t}_t \right]$$

K_t: intrinsics. $R_t, \mathbf{t}_t$: world-to-camera transform; the equation uses its inverse.

Pixel + depth → camera point → point in a shared world frame.

Xiao et al., SpatialTrackerV2 (2025) · Zhang et al., D4RT (2026)

A point trajectory representation

Point identity	t_a	t_b	t_c
Point i	$\mathbf{X}_{i,a}$	$\mathbf{X}_{i,b}$	$\mathbf{X}_{i,c}$
Point j	$\mathbf{X}_{j,a}$	$\mathbf{X}_{j,b}$	$\mathbf{X}_{j,c}$
Point k	$\mathbf{X}_{k,a}$	$\mathbf{X}_{k,b}$	$\mathbf{X}_{k,c}$

$$X \in \mathbb{R}^{N \times T \times 3}$$

N queried points, T requested times, and three spatial coordinates. Visibility and confidence are stored separately.

Rigid, articulated and deformable motion

Rigid



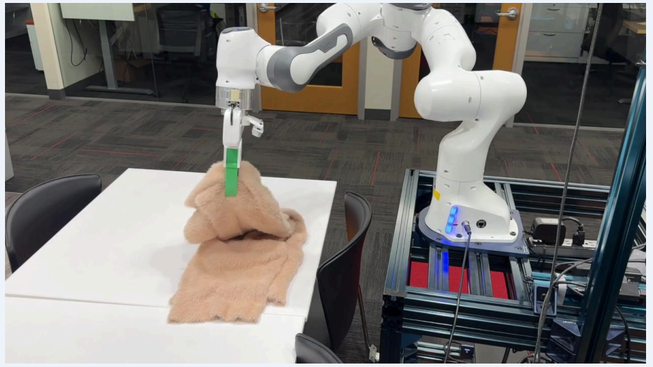
One body changes pose.

Articulated



Parts move through joints.

Deformable

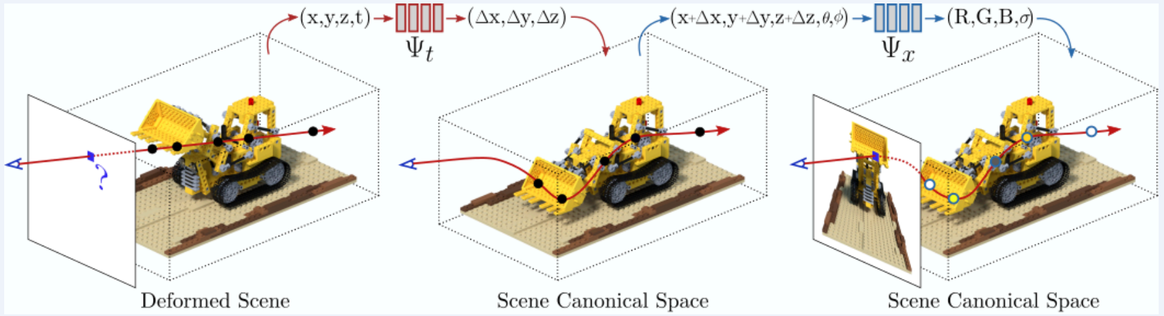


Different points change shape together.

These describe motion structure; each can use several spatial representations.

Huang et al., PointWorld (2026) · Pfaff et al., MeshGraphNets (2021)

Canonical shapes and deformation



A deformation map connects a time-dependent scene to a reference configuration.

$$\mathbf{x}_t = \Phi_t(\mathbf{x}_{\text{ref}})$$

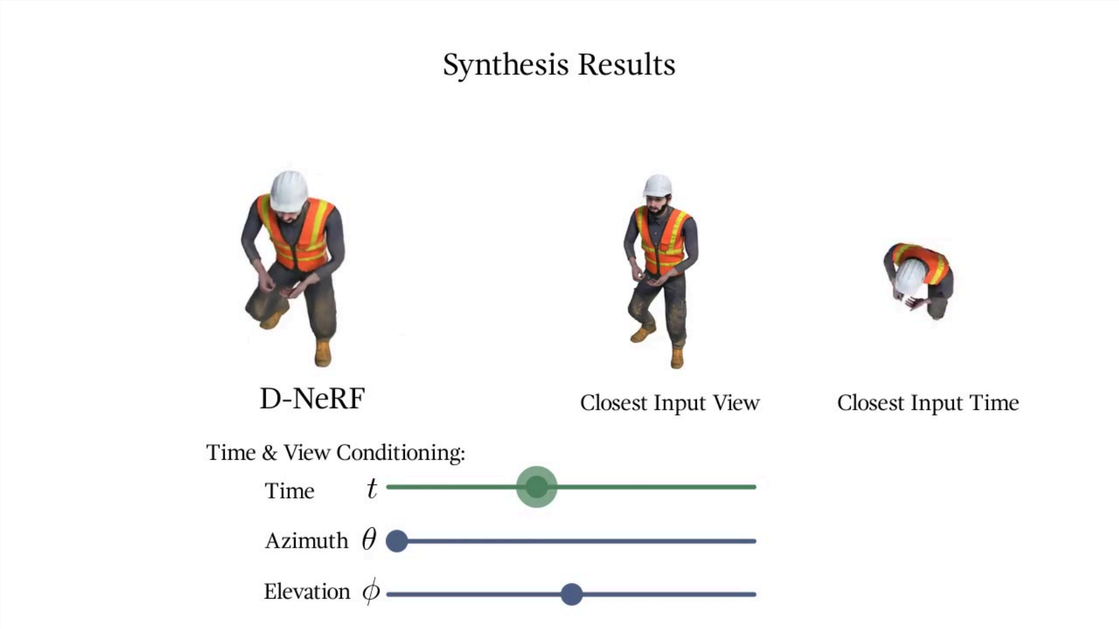
One reference shape or volume, plus a time-dependent mapping.

Some methods use the inverse direction: current space to canonical space.

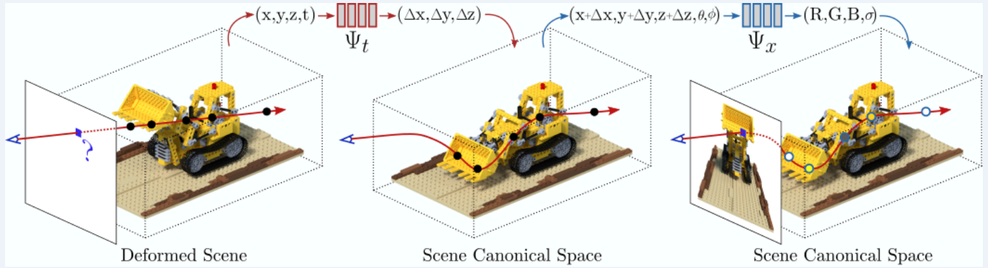
The mapping direction must match the query and the renderer.

Pumarola et al., D-NeRF (2021) · Wu et al., 4D Gaussian Splatting (2024)

Dynamic NeRF



D-NeRF reconstructs and renders a dynamic scene across time and viewpoint.

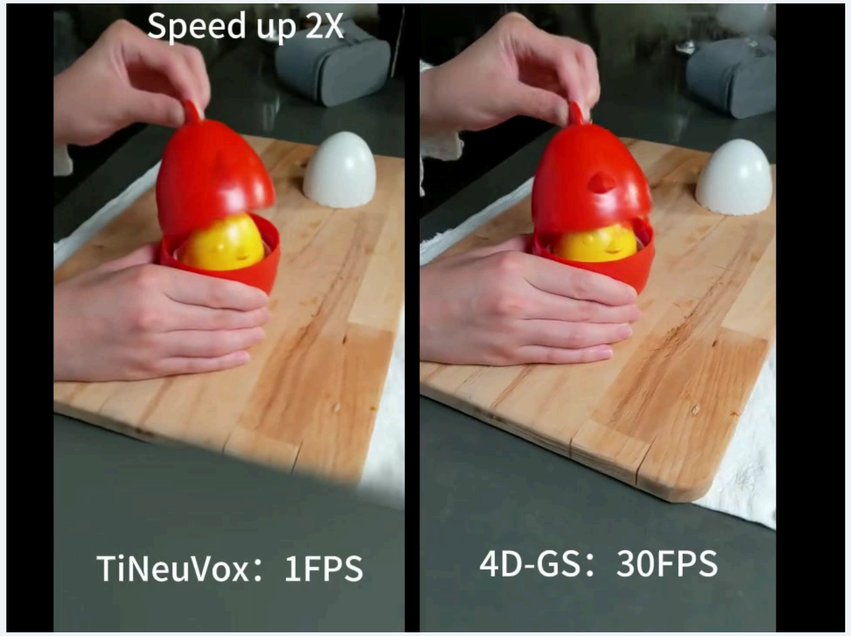


Time-dependent deformation and a canonical radiance field.

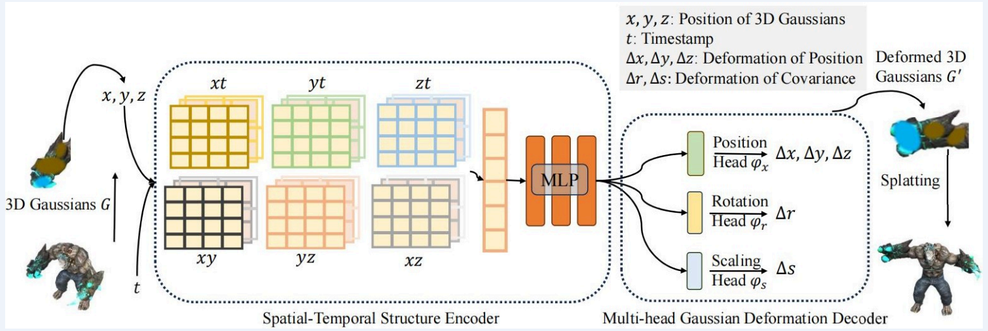
Time becomes a query variable in the scene representation.

Pumarola et al., D-NeRF (2021)

Dynamic Gaussian representations



Dynamic Gaussian rendering at different views and times.



A canonical Gaussian representation with learned time-dependent deformation.

A dynamic rendering representation can share structure across time.

Wu et al., 4D Gaussian Splatting (2024) · Kerbl et al., 3D Gaussian Splatting (2023)

Tracking representations

Representation	What stays indexed?	What changes over time?
Pointmap	Image pixels in a particular view	Their reconstructed 3D positions
Point trajectory	A selected physical point	Its position in a reference frame
Voxel / dynamic field	Spatial locations or a reference domain	Attributes or deformation
Dynamic mesh / splats	Representation elements	Positions and other parameters

Different methods can predict the same event through different objects.

Feng et al., St4RTrack (2025) · Xiao et al., SpatialTrackerV2 (2025) · Wu et al., 4D Gaussian Splatting (2024)