Multi-View 3D Point Tracking

Frano Rajič[1] Haofei Xu[1] Marko Mihajlovic[1] Siyuan Li[1] Irem Demir[1]
Emircan Gündoğdu[1] Lei Ke[2] Sergey Prokudin[1,3] Marc Pollefeys[1,4] Siyu Tang[1]

[1]ETH Zürich [2]Carnegie Mellon University [3]Balgrist University Hospital [4]Microsoft

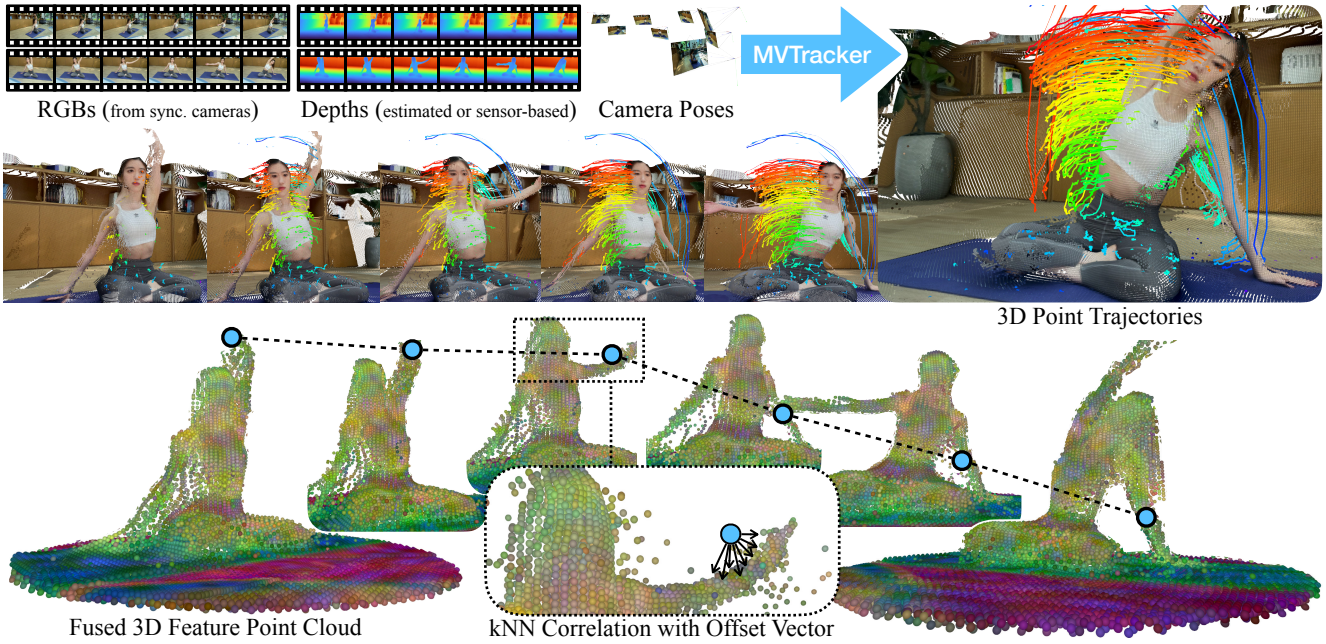

Figure 1. We introduce **MVTracker**, the first *data-driven multi-view 3D point tracker* for tracking arbitrary 3D points across multiple cameras. Our method fuses multi-view features into a unified 3D feature point cloud, within which it leverages kNN-based correlation to capture spatiotemporal relationships across views. A transformer then iteratively refines the point tracks, handling occlusions and adapting to varying camera setups without per-sequence optimization. This figure shows results on SelfCap [40] using DUSt3R-based [37] depth.

Abstract

We introduce the first data-driven multi-view 3D point tracker, designed to track arbitrary points in dynamic scenes using multiple camera views. Unlike existing monocular trackers, which struggle with depth ambiguities and occlusion, or prior multi-camera methods that require over 20 cameras and tedious per-sequence optimization, our feed-forward model directly predicts 3D correspondences using a practical number of cameras (e.g., four), enabling robust and accurate online tracking. Given known camera poses and either sensor-based or estimated multi-view depth, our tracker fuses multi-view features into a unified point cloud and applies k-nearest-neighbors correlation alongside a transformer-based update to reliably estimate long-range 3D correspondences, even under occlusion. We train on 5K synthetic multi-view Kubric sequences and evaluate on two real-world benchmarks—Panoptic Studio and DexYCB—achieving median trajectory errors of 3.1 cm and 2.0 cm, respectively. Our method generalizes well to diverse camera setups of 1–8 views with varying vantage points and video lengths of 24–150 frames. By releasing our tracker alongside training and evaluation datasets, we aim to set a new standard for multi-view 3D tracking research and provide a practical tool for real-world applications. Project page: https://ethz-vlg.github.io/mvtracker.

1. Introduction

Tracking arbitrary points in 3D [30] is a fundamental problem in computer vision, with numerous applications in dynamic scene reconstruction [14], robotics [10], and aug-

mented reality [25]. While remarkable progress has been made with the recent advancement of 2D point tracking methods [5, 7–9, 12, 16, 17, 45], they remain inherently limited in modeling 3D-consistent motion due to the fundamental ambiguity of the 3D-to-2D projection process. Thus, high-quality 3D point tracking remains a challenging task.

Scene flow approaches [26, 28, 30, 31] can estimate a dense 3D flow field for every 3D point, but they are usually limited to two consecutive video frames, whereas 3D point tracking operates on long sequences (*e.g.*, tens or even hundreds of video frames). Recently, several methods [18, 24, 38] have been proposed to tackle the 3D point tracking task from monocular videos. However, their performance remains far from satisfactory for real-world applications that demand high quality and robustness, due to the difficulty of estimating 3D from single viewpoints in challenging scenarios such as occlusion and complex motion.

To address these challenges, we adopt a multi-camera setup and develop the first feed-forward model that can efficiently and robustly predict long-range 3D point trajectories from multi-view videos. Unlike previous multi-camera methods [23, 30, 44] that require more than 20 cameras and tedious per-sequence optimization, our approach enables feed-forward 3D point tracking with a practical and flexible number of cameras, such as a set of four camera views with arbitrary viewpoints. Thus, our method strikes a promising balance between accuracy and practicality, making it particularly suitable for real-world applications.

Key to our model is a dynamic fused 3D feature point cloud, constructed by combining the unprojected per-view depth maps, which not only effectively aggregates multi-view information to a global scene representation, but also facilitates reliable and efficient correspondence search with the k-nearest-neighbors (kNN) operation. This is different from the triplane representation used in the previous state-of-the-art method SpatialTracker [38], which inevitably suffers from information loss during the triplane splatting process and is therefore less effective in processing varying numbers of input cameras. Moreover, our model performs reasonably well with different sources of depth input, whether from accurate depth sensors or noisy estimates from methods such as DUSt3R [37] and VGGT [33].

Using the fused dynamic point cloud, we retrieve local neighbors for each tracked point using a kNN search and compute multi-scale correlation features that capture both appearance similarity and 3D offset information. A spatiotemporal transformer then iteratively refines each point's 3D position and appearance over a sliding temporal window. Finally, the outputs from overlapping windows are merged to produce globally consistent 3D point trajectories.

To train our model, we construct a synthetic multi-view dataset using Kubric [11] and simulate 5K sequences. We evaluate the model on two real-world datasets, DexYCB [3] and Panoptic Studio [15], for which we construct 3D point trajectories by leveraging ground-truth object and hand pose estimation labels in DexYCB and merging existing monocular trajectory labels [18] in Panoptic Studio. We conduct extensive experiments on these multi-view video datasets, which suggest that our model outperforms baseline methods by a significant margin. Our model also performs well with different camera setups, different numbers of cameras, and different sources of depth maps, indicating the robustness of our proposed approach. We release our source code and models alongside the training and evaluation datasets to facilitate future research on multi-view 3D point tracking.

2. Related Work

Scene Flow. Scene flow methods [26, 28, 30, 31] are usually designed to estimate the dense 3D motion between two consecutive video frames. Traditional methods [26, 30, 31] try to solve this task with an optimization framework, which is typically slow and requires many cameras (*e.g.*, 31 in [30]) to well-regularize the optimization process. Modern approaches [21, 28, 41] explore data-driven models to directly predict scene flow in a feed-forward manner. Despite recent progress, existing models are limited to two frames and are not able to track long-range correspondences in 3D.

2D Point Tracking. Recent years have witnessed significant progress in long-term 2D point tracking [5, 7–9, 12, 13, 16, 17, 34, 45]. For instance, CoTracker2 [16] leverages self-attention to aggregate spatial context from supporting tracks, while LocoTrack [5] extends traditional 2D correlation into bidirectional 4D correlation for more robust local matching. These methods typically estimate long-range point trajectories over an entire video, handling occlusions effectively. In this work, we investigate their extension to 3D, but with a sparse multi-view setup and known camera parameters, rather than monocular input. Moreover, unlike recent trackers [9, 17] that reduce the synthetic-to-real domain gap via self-supervision or pseudolabeling, our approach focuses on direct supervision from synthetic data.

3D Point Tracking. 3D point tracking has seen innovative contributions from recent methods [23, 24, 35, 38, 39, 42, 43]. SpatialTracker [38] generalizes point tracking into 3D by integrating depth information through a Triplane representation [2] and DELTA [24] introduces a coarse-to-fine approach to estimate dense trajectories across the entire image plane, as opposed to sparse point tracking. Both methods assume monocular input. The concurrent SpatialTrackerV2 [39] and TAPIP3D [43] are also monocular. In a different approach, Dynamic 3DGS [23] leverages 3D Gaussian reconstructions over time to track dense scene elements in 3D and perform accurate 2D and 3D point tracking, but requires a multi-camera setup of 27 cameras in Panoptic Studio [15] in addition to sparse depth from depth sen-

sors for point cloud initialization and segmentation masks for relevant objects. We also use multiple input views but assume fewer cameras (*e.g.*, four), which is more practical for real-world applications. Works such as [6, 27, 35] use monocular tracking priors and our experiments suggest that the multi-view extension of [35] is constrained by this. In contrast, our method directly learns a multi-view tracking prior that is neither Gaussian-based nor optimization-based. VideoDoodles [42] presents an interactive system that anchors hand-drawn animations to objects within a reconstructed 3D scene, harnessing optical flow and depth maps in a novel 3D point tracking algorithm. However, it is primarily designed for video editing rather than competitive performance on standard benchmarks such as TAPVid-2D [7], whereas we focus on 3D point tracking accuracy.

3. Method

Our goal is to perform online 3D point tracking across multiple views, given a set of synchronized RGB frames with known camera parameters. The input consists of video sequences captured from V different camera views, with each frame containing a set of query points that need to be tracked over time. We estimate temporally consistent 3D trajectories for these points while also predicting visibility, handling occlusions, and adapting to scene dynamics. Our method runs online at 7.2 FPS given RGB-D input. For RGB-only input, we rely on external depth estimation (*e.g.*, DUSt3R [37] runs at 0.17 FPS and VGGT [33] at 3.1 FPS). An overview of our method is provided in Fig. 2.

Unlike single-view trackers that compute feature correlation on a 2D grid [5, 7, 8, 12, 16, 17, 24] or a triplane [38], our method leverages a fused multi-view point cloud representation to establish k-nearest neighbors (kNN) feature correlations. While 2D-grid-based correlation can match irrelevant background pixels and triplane-based correlation inherently suffers from information loss, our kNN targets geometrically relevant 3D neighborhoods. This enables more robust tracking by integrating geometric consistency across views and learning motion priors directly from data.

3.1. Problem Formulation

Given a calibrated multi-view video sequence $\mathbf{I}_t^v \in \mathbb{R}^{H\times W\times 3}$ captured by V cameras over T frames, our goal is to track N query points in 3D space. We denote each of the N query points as $\mathbf{q}^n = (t_q^n,\, x_q^n,\, y_q^n,\, z_q^n) \in \mathbb{R}^4$, where t_q^n denotes the query frame and $(x_q^n,\, y_q^n,\, z_q^n)$ its initial 3D location in world coordinates. Besides images and query points, the camera intrinsics $\mathbf{K}_t^v \in \mathbb{R}^{3\times 3}$ and extrinsics $\mathbf{E}_t^v \in \mathbb{R}^{4\times 4}$ are given for each view v and frame t.

The goal is to predict the 3D trajectory $\{\mathbf{p}_t^n = (x_t^n,\, y_t^n,\, z_t^n) \in \mathbb{R}^3,\, t \geq t_q^n\}$, along with visibilities $\{v_t^n \in \{0, 1\}\}$ where $v_t^n = 1$ denotes that the point is visible in at least one camera view, and $v_t^n = 0$ indicates occlusion in all views. Note that the query location matches the ground-truth track location at time t_q^n, *i.e.* $\mathbf{p}_{t_q^n}^n = (x_q^n,\, y_q^n,\, z_q^n)$.

3.2. Point Cloud Encoding of Multi-view Videos

Feature Maps. For each input frame $\mathbf{I}_t^v$, we extract d-dimensional per-view feature maps $\Phi_t^v = \varphi(\mathbf{I}_t^v)$ using a convolutional backbone (same as in [16, 17, 38]). We use a stride factor of $k = 4$ to downscale the input resolution for computational efficiency, such that $\Phi_t^v \in \mathbb{R}^{\frac{H}{k}\times\frac{W}{k}\times d}$. We also compute the feature maps at $S = 4$ different scales, *i.e.*, $\Phi_t^{v,s} \in \mathbb{R}^{\frac{H}{k2^{s-1}}\times\frac{W}{k2^{s-1}}\times d}$, $s = 1, \ldots, S$. These are obtained by downscaling the base features (*e.g.*, via average pooling) and will later be used in our pyramid correlation.

Fused 3D Feature Point Cloud. We obtain multi-view-consistent depth maps $\mathbf{D}_t^v \in \mathbb{R}^{H\times W}$ for each view v and frame t using off-the-shelf depth estimation [23, 33, 37] (see Appendix A). Sensor-based depth (*e.g.*, Kinect) can also be used if available and preferred. Using the depth and camera parameters, we lift each valid pixel (u_x, u_y) into 3D:

$$\mathbf{x} \;=\; \mathbf{E}_t^{v^{-1}}\big(\mathbf{K}_t^{v^{-1}}\,(\,u_x, u_y, 1\,)^\top \cdot \mathbf{D}_t^v[\,u_y, u_x\,]\big). \qquad (1)$$

Invalid pixels are those that do not have a valid depth value due to missing Kinect depth values. For estimated depth, we retain all pixels and train for robustness (instead of thresholding by uncertainty predictions). Each of these lifted points $\mathbf{x}$ is associated with its corresponding feature from $\Phi_t^{v,s}$ and then fused across views into a single point cloud:

$$\mathcal{X}_t^s \;=\; \big\{\big(\mathbf{x},\, \Phi_t^{v,s}[\,u_y, u_x\,]\big) \;\big|\; v \in \{1, \ldots, V\},\; (u_x, u_y) \in \Omega_t^v\big\}, \qquad (2)$$

where Ω_t^v is the set of valid pixels in view v. This unified 3D representation facilitates the computation of spatial correlation across views, enabling multi-view-consistent tracking.

An alternative to our fused point cloud is to combine multi-view features via a single global *triplane* [38], projecting 3D points onto three orthogonal planes (*e.g.*, XY, YZ, ZX). However, this inevitably suffers from projection collisions: different surfaces map to the same planar coordinates, causing destructive feature averaging. It also requires choosing a fixed bounding region for the scene, which leads to wasted resolution or clipped content for large or uncentered scenes. By contrast, our point cloud preserves features directly in 3D, avoids collisions, and adapts naturally to different scenes, resulting in more robust multi-view tracking.

Track Features. We assign a d-dimensional time-dependent appearance feature $\mathbf{f}_t^n$ to each tracked point. These features are initialized by sampling from the fused 3D feature point cloud at the query frame and 3D location:

$$\big(\mathbf{x}^{n,*}, \phi^{n,*}\big) \;=\; \underset{(\mathbf{x},\,\phi)\,\in\,\mathcal{X}_{t_n^q}^1}{\operatorname{argmin}} \;\|\mathbf{x} - \mathbf{p}_{t_n^q}^n\|, \qquad (3)$$

for every track n. We use the sampled feature $\phi^{n,*}$ to initialize $\mathbf{f}_t^n$ for all $t \geq t_n^q$. These track features are subsequently

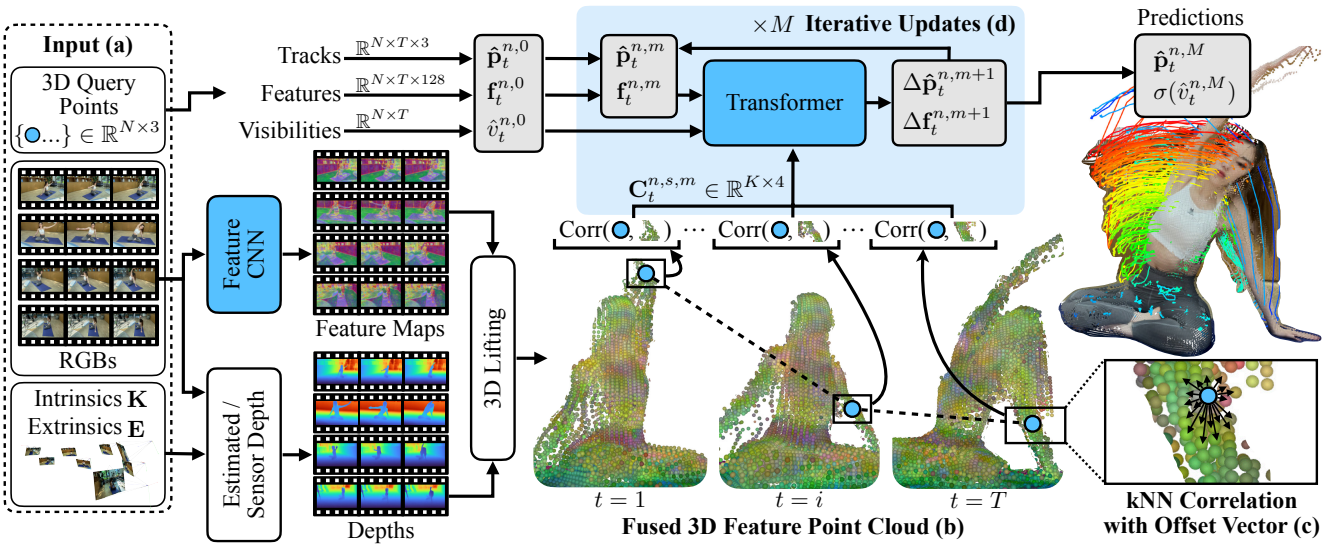

Figure 2. **MVTracker Pipeline**. (a) Given synchronized multi-view RGB videos with known intrinsics and extrinsics, we extract per-view feature maps using a CNN-based encoder. (b) We construct a point cloud from estimated or sensor-based depth maps, associating each point with learned feature embeddings (Sec. 3.2). (c) We compute directed kNN-based correlation within the point cloud, capturing spatiotemporal relationships across views (Sec. 3.3). (d) A transformer-based update module iteratively refines point trajectories using self-attention over multi-view feature correlations within a sliding temporal window (Sec. 3.4). (e) The model processes sequences in overlapping sliding windows, producing temporally consistent 3D point trajectories with occlusion-aware visibility predictions (Sec. 3.5). The blue blocks denote trainable neural models. The visualized sequence is from SelfCap [40] and uses DUSt3R [37] to estimate depth.

used when computing track-centric spatial correlation with nearest neighbors from the fused 3D feature point cloud (see Sec. 3.3), but also refined by the spatiotemporal transformer to capture appearance changes over time (see Sec. 3.4).

3.3. Multi-Scale Spatial Correlation

Instead of computing correlations separately for each view [5, 7, 8, 12, 16, 17, 24] or on an auxiliary triplane [38], we establish correspondences directly in the fused feature point cloud using a multi-scale kNN approach. For each query point, we retrieve its K nearest neighbors at different scales from $\mathcal{X}_t^s$ and compute local kNN feature correlations:

$$\mathbf{C}_t^{n,s} = \left\{ \langle \mathbf{f}_t^n, \phi_k \rangle \;\middle|\; (\mathbf{x}_k, \phi_k) \in \mathcal{N}_K(\hat{\mathbf{p}}_t^n, \mathcal{X}_t^s) \right\}, \quad (4)$$

where $\langle \cdot, \cdot \rangle$ is the dot product and $\mathcal{N}_K(\hat{\mathbf{p}}_t^n, \mathcal{X}_t^s)$ finds the k-nearest neighbors around the current location estimate $\hat{\mathbf{p}}_t^n$ in the fused 3D feature point cloud. These correlations capture spatial dependencies at multiple scales and are fed into our transformer-based tracking module, allowing it to search increasingly larger 3D neighborhoods. For reference, average neighbor distances at the four scales on Panoptic Studio (made-up of ~10m–wide scenes) are 12.5 ± 8.2, 22.4 ± 12.2, 42.7 ± 21.7, and 85.8 ± 44.0 cm. At the highest scale, this covers frame-to-frame motion of up to ~92 km/h at 30 FPS.

Unlike 2D correlation, where pixel offsets implicitly encode direction, our 3D point cloud representation requires explicit 3D offsets. Concretely, for each neighbor $(\mathbf{x}_k, \phi_k)$ we concatenate the local similarity $\langle \mathbf{f}_t^n, \phi_k \rangle$ with the offset vector $(\mathbf{x}_k - \hat{\mathbf{p}}_t^n)$ to encode direction and distance from $\hat{\mathbf{p}}_t^n$. This offset helps the model disambiguate different nearest neighbors and enables correlation-based matching in 3D.

3.4. Transformer-Based Iterative Tracking

Our transformer refines track estimates over a sliding window of T frames. For each track n at time t, we construct a token $G_t^n = (\eta(\hat{\mathbf{p}}_t^n - \hat{\mathbf{p}}_{t_n^q}^n), \mathbf{f}_t^n, \mathbf{C}_t^{n,s}, \hat{v}_t^n)$, where $\eta(\cdot)$ is a sinusoidal positional encoding. The tokens $\{G_t^n\}_{t,n}$ are passed to a transformer ψ, which applies temporal self-attention across time and cross-attention with a small set of learned virtual tracks to model spatial dependencies, following the design in CoTracker2 [17]. The transformer outputs residual position and feature updates $(\Delta\hat{\mathbf{p}}_t^n, \Delta\mathbf{f}_t^n) = \psi(G_t^n)$, which we apply iteratively for $m = 1, \ldots, M$:

$$\hat{\mathbf{p}}_t^{n,m+1} = \hat{\mathbf{p}}_t^{n,m} + \Delta\hat{\mathbf{p}}_t^{n,m+1}, \quad (5)$$

$$\mathbf{f}_t^{n,m+1} = \mathbf{f}_t^{n,m} + \Delta\mathbf{f}_t^{n,m+1}. \quad (6)$$

After each iteration m, we recompute $\mathbf{C}_t^{n,s,m}$ based on the refined position and feature. At the final iteration $m = M$, we estimate visibility as $\hat{v}_t^n = \sigma(W\,\mathbf{f}_t^{n,M})$, where $\sigma(\cdot)$ is the sigmoid function and W a learned projection matrix.

3.5. Windowed Inference and Unrolled Training

To process long videos, we adopt a windowed approach as in CoTracker [16]. Let T be the maximum window size; for a longer video of length $T' > T$, we divide it into

J overlapping windows of length T, each shifted by $T/2$ frames. Once the transformer completes M iterative updates for window j, its final track estimates initialize window $j+1$, allowing refined trajectories to propagate. Let $\hat{\mathbf{p}}_t^{n,m,j}$ be the predicted location of query n at time t after iteration m in window j, and $\hat{v}_t^{n,j}$ its visibility (predicted only at the final iteration). During training, we unroll these windowed updates so that each iteration learns to correct or refine predictions from previous windows and iterations.

3.6. Supervision

Training is supervised with ground-truth trajectories $\mathbf{p}_t^n$ and visibility labels v_t^n. The total loss consists of a position loss ($\mathcal{L}_{\text{xyz}}$) and a visibility loss ($\mathcal{L}_{\text{vis}}$), balanced by λ_{vis}:

$$\mathcal{L} = \mathcal{L}_{\text{xyz}} + \lambda_{\text{vis}}\mathcal{L}_{\text{vis}}. \tag{7}$$

In particular, the position loss is the weighted ℓ_1-norm error between the predicted and the ground-truth 3D positions:

$$\mathcal{L}_{\text{xyz}} = \sum_{j,m,n,t} \frac{\gamma^{M-m}}{JMNT} \left\| \hat{\mathbf{p}}_t^{n,m,j} - \mathbf{p}_t^{n,m,j} \right\|_1, \tag{8}$$

where j indexes the windows J, m the iterative updates M, n the trajectories N, t the number of frames T, and γ is a weighting factor used to penalize the later iterations of the iterative refinement more. The visibility loss addresses class imbalance by minimizing the balanced binary cross-entropy (B-BCE) between predicted and ground-truth labels:

$$\mathcal{L}_{\text{vis}} = \sum_{j,t,n} \frac{1}{JTN} \text{B-BCE}\big(\hat{v}_t^{n,j},\, v_t^{n,j}\big). \tag{9}$$

4. Experiments

Datasets. To train our model, we generate a synthetic multi-view dataset, MV-Kub, using Kubric [11], simulating 5K multi-view video sequences. Since no large-scale datasets exist for multi-view 3D point tracking, we rely on synthetic data for supervised training. For evaluation on the hand dexterity dataset DexYCB [3], we sample 10 scenes, leveraging ground-truth object and hand poses to generate track labels. We also construct a Panoptic Studio evaluation dataset by merging existing monocular labels from TAPVid-3D [18] for a total of 6 scenes. Evaluation query points are randomly sampled from both static and moving surfaces.

All results, unless otherwise stated, are reported assuming the availability of estimated depth [37] for DexYCB, ground-truth optimization-based depth [23] for Panoptic Studio, and simulated depth for MV-Kub. In Appendix C we ablate the impact of depth sources on the performance.

Evaluation Metrics. As no standard metrics exist for multi-view 3D point tracking, we extend those from monocular benchmarks [7], reporting four key metrics: (1) Median Trajectory Error (MTE) for visible points, quantifying spatial accuracy; (2) δ_{avg}, the average location accuracy over a set of threshold distances; (3) Occlusion Accuracy (OA), measuring the model's binary visibility prediction across views; and (4) Average Jaccard (AJ), which jointly evaluates occlusion and position accuracy. All metrics are computed per track, averaged within a scene, and finally aggregated across all dataset scenes. See Appendix F for details.

Training. We train our method on MV-Kub for 200K steps on 8 GH200 chips with 96GB GPU memory over 8 days, with a batch size of 8 and up to 2048 trajectories per sample. Our model is implemented in PyTorch (with Lightning Fabric for multi-node training) and optimized using AdamW [22]. The model operates with a latent feature dimension of $d = 128$ and is trained on MV-Kub sequences of up to 24 frames using a sliding window of 12 frames. We use six-head self-attention layers with a hidden size of 256. We apply a range of augmentations to improve generalization and robustness. See Appendix C.4 for further details.

Baselines. Since there currently exist no other feed-forward multi-view point trackers, we implement a triplane-based baseline by extending SpaTracker [38] to fuse multi-view features into a single global triplane, referred to as the "Triplane Baseline" in experiments. We also evaluate against two multi-view methods that require test-time optimization: Shape of Motion [35] and Dynamic 3DGS [23]; please refer to Appendix D for their implementation details.

We further evaluate against monocular 2D and 3D point trackers. For 2D point trackers (CoTracker2 [16], CoTracker3 [17], LocoTrack [5]), we run the models on each camera separately with the query points that are deemed to be best visible from that view (based on their projected depth), then lift the resulting 2D estimates to 3D using the known intrinsics and the same depth used by our model. We similarly run 3D trackers (DELTA [24], SpaTracker [38], SpaTrackerV2 [39], TAPIP3D [43]), but give depth as input to directly get 3D tracks. We finally merge per-camera predictions into the world space using known camera poses.

4.1. Method Comparisons

Tab. 1 compares MVTracker against several baselines on Panoptic Studio, DexYCB, and Multi-View Kubric. Our method consistently achieves higher location accuracy (δ_{avg}), better occlusion-aware tracking (AJ), and lower trajectory errors (MTE). This is visually apparent in Fig. 3.

On Panoptic Studio, our tracker obtains an AJ of **86.0** and a δ_{avg} of **94.7**, substantially outperforming single-view methods such as SpaTracker (61.5 AJ) and CoTracker3 (74.5 AJ). We observe similar trends on DexYCB, where our method reaches an AJ of **71.6** with a median trajectory error of only **2.0** cm, compared to the lower performance of LocoTrack, DELTA, and CoTracker3. This is because 2D point trackers are not robust to lifting their 2D trajectories into 3D estimates based on estimated depth maps. Finally, on the Multi-View Kubric validation set, our model attains an AJ of **81.4** and an exceptionally low MTE of **0.7** cm.

Table 1. **Quantitative evaluation of multi-view 3D point tracking performance.** We report results on Panoptic Studio [18], DexYCB [3], and Multi-View Kubric [11]. Our method outperforms existing approaches across all datasets, achieving the highest accuracy (δ_{avg}) and occlusion-aware tracking performance (AJ), while maintaining lower median trajectory error (MTE). Notably, our model surpasses the triplane-based baseline, indicating the effectiveness of kNN-directed correlation for multi-view tracking. We report MTE in centimeters. F denotes that the method is feed-forward, data-driven, and online. M denotes that the method supports fusing multi-view inputs. We use estimated depth [37] for DexYCB, ground-truth optimization-based depth [23] for Panoptic Studio, and simulated depth for Kubric.

Method	Train	F	M	Panoptic Studio [18]				DexYCB [3] (DUSt3R depth)				Multi-View Kubric [11]			
				AJ ↑	δ_{avg} ↑	OA ↑	MTE ↓	AJ ↑	δ_{avg} ↑	OA ↑	MTE ↓	AJ ↑	δ_{avg} ↑	OA ↑	MTE ↓
Dynamic 3DGS [23]	✗	✗	✓	66.5	92.4	74.1	3.9	45.7	57.1	81.3	11.3	30.4	48.6	68.3	11.2
Shape of Motion [35]	✗	✗	✓	72.6	89.2	82.1	4.8	36.2	53.6	63.3	8.0	57.8	63.4	86.2	5.3
SceneTracker [32]	LSFO	✓	✗	–	85.1	–	6.3	–	61.1	–	5.7	–	56.9	–	7.8
LocoTrack [5]	Kub	✓	✗	65.8	79.4	79.6	11.8	27.8	38.6	77.0	22.8	52.5	58.4	76.5	12.9
DELTA [24]	Kub	✓	✗	68.1	86.3	79.2	5.9	36.8	51.6	61.0	18.3	57.4	68.4	77.9	9.6
CoTracker2 [16]	Kub	✓	✗	69.5	83.2	80.7	10.4	28.8	40.5	76.2	20.8	54.6	60.3	79.8	10.4
CoTracker3 [17]	Kub+15k	✓	✗	74.5	84.5	86.4	8.6	29.4	40.3	78.6	22.0	55.1	60.6	77.7	11.9
SpaTracker [38]	Kub	✓	✗	55.5	70.8	81.7	9.9	48.1	62.6	73.9	5.5	43.6	55.1	76.3	5.1
SpaTracker [38]	MV-Kub	✓	✗	61.5	82.3	75.5	7.3	58.3	72.0	80.2	5.9	65.5	77.6	83.1	2.2
SpaTrackerV2 [39]	See [39]	✓	✗	75.3	85.1	91.2	6.9	35.5	45.1	91.3	9.8	58.6	69.3	86.3	3.9
TAPIP3D [43]	Kub	✓	✗	84.3	93.8	91.4	**3.1**	38.8	50.0	90.1	8.2	72.4	86.5	85.4	1.3
Triplane Baseline	MV-Kub	✓	✓	65.1	81.8	82.8	7.2	57.5	71.0	81.3	4.3	74.7	85.2	90.4	1.2
MVTracker (ours)	MV-Kub	✓	✓	**86.0**	**94.7**	**92.3**	**3.1**	**71.6**	**80.6**	**91.3**	**2.0**	**81.4**	**90.0**	**93.7**	**0.7**

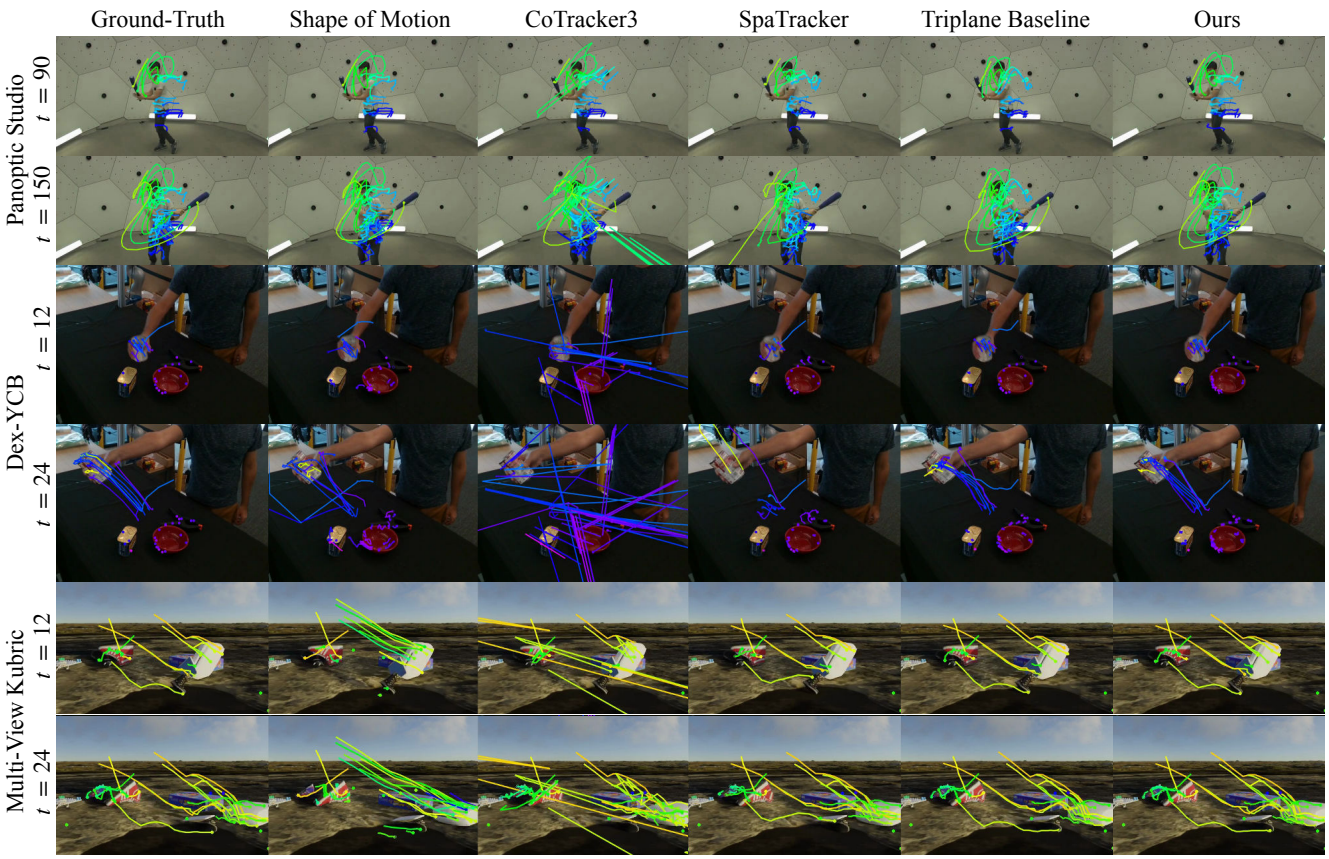

Figure 3. **Qualitative comparison of multi-view 3D point tracking.** We visualize results from a camera viewpoint *not* used during inference (and not near the input cameras). Each pair of rows corresponds to two time steps from the same dataset. The leftmost column shows ground-truth 3D trajectories, while remaining columns depict predictions from different methods. Points predicted as occluded are indicated by empty circles. Compared to baselines, our method more accurately maintains correspondences across views and handles occlusions. Please see the supp. video and project page for additional visualizations. These examples correspond to the results in Tab. 1.

Table 2. **Point Correlation Components.** "No offset" omits the offset vector entirely, "Offset + location" concatenates both the offset and the neighbor's world-space coordinates, and "Offset only" encodes just the relative direction. Including only the offset vector (third row) yields the best overall performance. These experiments were trained for 25% of the total steps and on 8 GPUs.

	Multi-View Kubric [11]			
	AJ ↑	δ_{avg} ↑	OA ↑	MTE ↓
No offset	21.3	45.3	40.6	15.6
Offset + location	48.7	59.6	68.5	6.8
Offset only	**53.6**	**64.9**	**73.4**	**4.3**

Compared to optimization-based methods, which require per-sequence training, our approach is fully feed-forward and runs at 7.2 FPS. While Shape of Motion achieves reasonable accuracy, its iterative optimization makes it impractical for large-scale or real-time applications. Dynamic 3DGS further requires dense camera setups, limiting its applicability to real-world scenarios with fewer cameras.

Impact of the kNN-based correlation. Our method's strong performance is largely due to its kNN-based correlation mechanism. Replacing it with a world-aligned triplane correlation leads to a significant drop in performance (see Tab. 1). Triplane-based methods compress multi-view features onto fixed 2D planes, causing destructive feature collisions when different 3D points from the same or different views project to the same grid cell. This leads to information loss that cannot easily be disentangled, especially as the number of views increases. Moreover, triplanes require placing the 2D planes in a fixed scene-aligned coordinate frame with a pre-defined scale and extent, which is difficult to generalize across diverse camera setups and scene scales. In contrast, our kNN-based correlation operates directly in 3D world space and dynamically selects relevant neighbors, avoiding these issues and enabling more robust tracking.

4.2. Ablations

We further evaluate the impact of the point correlation components, the varying number of input views, and different camera setups, as well as training augmentation strategies.

Point Correlation Components. Tab. 2 analyzes the impact of different components in our correlation module. Unlike 2D correlation, where relative direction is implicitly encoded in the pixel grid, kNN in 3D retrieves neighbors from all directions, making explicit offset information crucial. We compare three variants: (i) no offset vector, (ii) offset vector plus explicit neighbor location, and (iii) offset vector only. We train each variant for 25% of total steps on 8 GPUs. The results suggest that (iii) "offset only" achieves the best performance across metrics, indicating that encoding only the relative offset, without concatenating absolute neighbor locations, yields most effective correspondences.

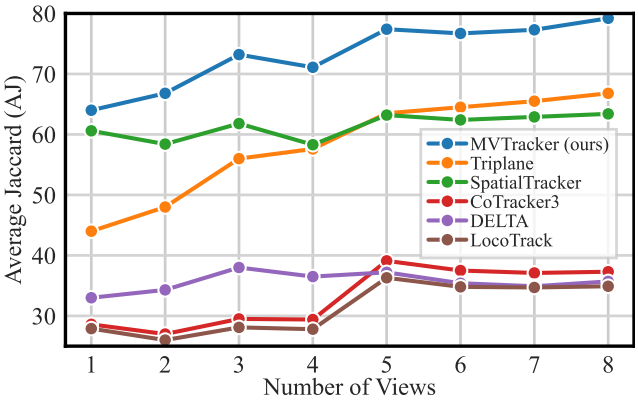

Figure 4. **Effect of the Number of Input Views** on DexYCB [3] (DUSt3R-based depth). MVTracker (blue) consistently improves with more views, reaching an AJ of 79.2 with eight views, indicating the benefit of multi-view information. SpatialTracker (green) and Triplane (orange) show moderate improvements but plateau earlier, while single-view methods such as CoTracker3, DELTA, and LocoTrack (red, purple, and brown) exhibit limited gains.

Effect of the Number of Input Views. Fig. 4 shows how performance changes with the number of input views on DexYCB. Our tracker consistently improves as more views are added, achieving an AJ of 64.0 with a single view, 71.1 with four views, and 79.2 with eight views. This trend highlights the importance of multi-view information in reducing depth ambiguities and improving spatial consistency.

Compared to baseline methods, our approach benefits the most from additional views, thus suggesting better scalability. While SpatialTracker and Triplane show moderate improvements with more views, their performance plateaus earlier, indicating difficulty in fully leveraging additional multi-view information. In contrast, single-view methods like CoTracker3, DELTA, and LocoTrack show limited gains, reinforcing their inability to reliably reconstruct 3D correspondences. Similar trends are observed on Panoptic Studio as well as Multi-View Kubric (see Appendix C.2).

Impact of Camera Setups. Tab. 3 shows the performance (measured in AJ) under different camera configurations on Panoptic Studio and DexYCB. For Panoptic Studio, we selected sets of four cameras from the available 27, ensuring they were either positioned opposite each other (Setup A) or placed nearby with varying baseline sizes (Setups B and C). For DexYCB, we sequentially chose four out of the available eight cameras: views 1–4 for setup A, views 3–6 for setup B, and views 5–8 for setup C. Note that on Multi-View Kubric the camera configuration is randomized (with cameras generally oriented toward the scene center), so separate evaluations are unnecessary; the main table (Tab. 1) already reflects a random sampling of camera placements. Our method consistently achieves high AJ across all tested

Table 3. **Impact of Camera Setups.** Evaluation of AJ under varied camera setups. For Panoptic Studio [18], we select 4 out of the available 27 views as opposing views (setup A) or nearby views (setups B and C). For DexYCB [3], we form 4-view sets out of the available 8 views: (A) views 1–4, (B) views 3–6, (C) views 5–8.

Method	PStudio [18]			DexYCB [3]		
	A	B	C	A	B	C
Dynamic 3DGS [23]	66.5	50.8	56.6	45.7	–	–
Shape of Motion [35]	72.6	64.3	66.8	36.2	–	–
LocoTrack [5]	65.8	57.9	63.7	27.8	40.9	42.9
DELTA [24]	68.1	61.1	65.9	36.5	43.3	47.6
CoTracker2 [16]	69.5	62.3	66.4	28.8	42.0	44.4
CoTracker3 [17]	74.5	66.3	70.9	29.4	43.8	46.3
SpaTracker [38]	61.5	54.8	57.8	58.3	57.9	63.8
Triplane Baseline	65.1	59.9	63.5	57.5	62.0	66.3
MVTracker (ours)	**86.0**	**75.7**	**83.2**	**71.0**	**71.2**	**78.3**

setups, indicating robustness to varying camera positions.

Ablation of Training Augmentation. In Appendix C.4, we examine how training augmentations impact performance, focusing on (i) the number of views used during training (ranging from 1 to 8) and (ii) the depth map source (ground truth vs. off-the-shelf estimation). Our results show that training with a variable number of views significantly improves robustness to different camera setups, ensuring better generalization across datasets. Additionally, incorporating both accurate and estimated depth maps helps mitigate domain shifts and enhances adaptability to real-world depth estimation errors. This is particularly noticeable on DexYCB, where DUSt3R depth quality varies significantly.

Overall, these ablations suggest that MVTracker remains robust across different camera configurations, benefits from diverse training conditions, and performs best with kNN-based offset encoding and carefully designed augmentations. Please refer to Appendix C for more ablation experiments and additional discussions. The inference speed measurements in Appendix B show that MVTracker runs at 7.2 FPS when provided with sensor depth, underscoring its suitability for near-real-time and large-scale applications.

5. Discussion and Future Work

While our method effectively tracks 3D points across multiple views, it has limitations. The key challenge is its reliance on the quality of sparse-view depth estimation. If no sensor depth is available, the approach assumes a reasonably accurate estimated depth. However, in sparse camera setups, such estimation can be unreliable or fail entirely, making tracking infeasible. While our learned motion priors help mitigate moderate noise or incompleteness, a more principled solution would involve jointly estimating depth and tracking for potential mutual refinement, or developing a foundation model for 4D reconstruction with integrated tracking capabilities. We view this as a critical future direction for the community. Appendix A provides additional analysis on the impact of depth quality, robustness to noise, comparisons across different estimators, and failure cases.

Another limitation is that our study focuses on tracking within bounded regions where camera overlap is sufficient. Extending to unbounded or outdoor environments presents additional challenges due to limited training data availability, scene scale variation, and less constrained viewpoints.

A related issue is scene normalization: our model is trained on a fixed dataset with randomized but similar scene scales and layouts. To bridge distribution gaps at test time, we currently apply manually or heuristically determined similarity transforms. While this works well on our benchmarks and a few additional qualitative datasets, more principled approaches are needed to support arbitrary new scenes.

Finally, long-term 3D point tracking faces the broader challenge of scarce large-scale real-world training data. Unlike optimization-based approaches, data-driven point trackers require diverse training distributions to generalize. However, most existing datasets are synthetic, limiting robustness. Recent community efforts [9, 17] show promise in leveraging self-supervised learning from real-world videos, which could be instrumental for improving generalization.

6. Conclusion

We introduce MVTracker, the first *data-driven multi-view 3D point tracker*, which combines kNN-based correlation in a fused 3D point cloud with a spatiotemporal transformer to track arbitrary points across multiple views. Our method outperforms single-view, multi-view, and optimization-based baselines, suggesting strong generalization across diverse camera configurations and occluded environments.

We hope that our work enables real-world use cases in robotics [29], provides a better data-driven multi-view prior for optimization-based reconstruction methods [19, 23, 35], and acts as a point tracking head in the upcoming age of scalable end-to-end 4D reconstruction and tracking models.

Acknowledgements

We are grateful for insightful discussions with Yiming Wang, Zador Pataki, Luigi Piccinelli, and Paul-Edouard Sarlin. We thank Ignacio Rocco and Skanda Koppula for their helpful clarifications regarding TAPVid-3D. This study was conducted within the national "Proficiency" research project (No. PFFS-21-19) funded by the Swiss Innovation Agency Innosuisse in 2021 as one of 15 flagship initiatives. This work was additionally supported as part of the Swiss AI initiative by a grant from the Swiss National Supercomputing Centre (CSCS) under project IDs A03 and A136 on Alps, enabling large-scale training and evaluation.

References

[1] Shariq Farooq Bhat, Reiner Birkl, Diana Wofk, Peter Wonka, and Matthias Müller. ZoeDepth: Zero-shot transfer by combining relative and metric depth. *arXiv preprint arXiv:2302.12288*, 2023. 5

[2] Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J Guibas, Jonathan Tremblay, Sameh Khamis, et al. Efficient geometry-aware 3d generative adversarial networks. In *CVPR*, 2022. 2

[3] Yu-Wei Chao, Wei Yang, Yu Xiang, Pavlo Molchanov, Ankur Handa, Jonathan Tremblay, Yashraj S. Narang, Karl Van Wyk, Umar Iqbal, Stan Birchfield, Jan Kautz, and Dieter Fox. DexYCB: A benchmark for capturing hand grasping of objects. In *CVPR*, 2021. 2, 5, 6, 7, 8, 3

[4] Xingyu Chen, Yue Chen, Yuliang Xiu, Andreas Geiger, and Anpei Chen. Easi3R: Estimating disentangled motion from dust3r without training. In *ICCV*, 2025. 1

[5] Seokju Cho, Jiahui Huang, Jisu Nam, Honggyu An, Seungryong Kim, and Joon-Young Lee. Local all-pair correspondence for point tracking. In *ECCV*, 2024. 2, 3, 4, 5, 6, 8

[6] Wen-Hsuan Chu, Lei Ke, and Katerina Fragkiadaki. DreamScene4D: Dynamic multi-object scene generation from monocular videos. In *NeurIPS*, 2024. 3

[7] Carl Doersch, Ankush Gupta, Larisa Markeeva, Adria Recasens, Lucas Smaira, Yusuf Aytar, Joao Carreira, Andrew Zisserman, and Yi Yang. TAP-Vid: A benchmark for tracking any point in a video. In *NeurIPS*, 2022. 2, 3, 4, 5

[8] Carl Doersch, Yi Yang, Mel Vecerik, Dilara Gokay, Ankush Gupta, Yusuf Aytar, Joao Carreira, and Andrew Zisserman. Tapir: Tracking any point with per-frame initialization and temporal refinement. *ICCV*, 2023. 3, 4

[9] Carl Doersch, Pauline Luc, Yi Yang, Dilara Gokay, Skanda Koppula, Ankush Gupta, Joseph Heyward, Ignacio Rocco, Ross Goroshin, João Carreira, et al. BootsTAP: Bootstrapped training for tracking-any-point. In *ACCV*, 2024. 2, 8

[10] Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, Junbo Wang, Haoyi Zhu, and Cewu Lu. RH20T: A robotic dataset for learning diverse skills in one-shot. In *RSS 2023 Workshop on Learning for Task and Motion Planning*, 2023. 1

[11] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, et al. Kubric: A scalable dataset generator. In *CVPR*, 2022. 2, 5, 6, 7, 3

[12] Adam W Harley, Zhaoyuan Fang, and Katerina Fragkiadaki. Particle video revisited: Tracking through occlusions using point trajectories. In *ECCV*, 2022. 2, 3, 4

[13] Adam W. Harley, Yang You, Xinglong Sun, Yang Zheng, Nikhil Raghuraman, Yunqi Gu, Sheldon Liang, Wen-Hsuan Chu, Achal Dave, Pavel Tokmakov, Suya You, Rares Ambrus, Katerina Fragkiadaki, and Leonidas J. Guibas. AllTracker: Efficient dense point tracking at high resolution. In *ICCV*, 2025. 2

[14] Linyi Jin, Richard Tucker, Zhengqi Li, David Fouhey, Noah Snavely, and Aleksander Holynski. Stereo4D: Learning how things move in 3d from internet stereo videos. In *CVPR*, 2025. 1

[15] Hanbyul Joo, Hao Liu, Lei Tan, Lin Gui, Bart Nabbe, Iain Matthews, Takeo Kanade, Shohei Nobuhara, and Yaser Sheikh. Panoptic studio: A massively multiview system for social motion capture. In *ICCV*, 2015. 2

[16] Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. CoTracker: It is better to track together. In *ECCV*, 2023. 2, 3, 4, 5, 6, 8, 1

[17] Nikita Karaev, Iurii Makarov, Jianyuan Wang, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. CoTracker3: Simpler and better point tracking by pseudo-labelling real videos. In *ICCV*, 2025. 2, 3, 4, 5, 6, 8, 1

[18] Skanda Koppula, Ignacio Rocco, Yi Yang, Joe Heyward, João Carreira, Andrew Zisserman, Gabriel Brostow, and Carl Doersch. TAPVid-3D: A benchmark for tracking any point in 3d. In *NeurIPS*, 2024. 2, 5, 6, 8, 1, 3, 4

[19] Jiahui Lei, Yijia Weng, Adam Harley, Leonidas Guibas, and Kostas Daniilidis. MoSca: Dynamic gaussian fusion from casual videos via 4d motion scaffolds. In *CVPR*, 2024. 8

[20] Zhengqi Li, Richard Tucker, Forrester Cole, Qianqian Wang, Linyi Jin, Vickie Ye, Angjoo Kanazawa, Aleksander Holynski, and Noah Snavely. MegaSaM: Accurate, fast and robust structure and motion from casual dynamic videos. In *CVPR*, 2024. 1, 5

[21] Haisong Liu, Tao Lu, Yihui Xu, Jia Liu, Wenjie Li, and Lijun Chen. CamLiFlow: bidirectional camera-lidar fusion for joint optical flow and scene flow estimation. In *CVPR*, 2022. 2

[22] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 5

[23] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3D Gaussians: Tracking by persistent dynamic view synthesis. In *3DV*, 2024. 2, 3, 5, 6, 8, 1, 4

[24] Tuan Duc Ngo, Peiye Zhuang, Chuang Gan, Evangelos Kalogerakis, Sergey Tulyakov, Hsin-Ying Lee, and Chaoyang Wang. DELTA: Dense efficient long-range 3d tracking for any video. In *ICLR*, 2024. 2, 3, 4, 5, 6, 8, 1

[25] Xiaqing Pan, Nicholas Charron, Yongqian Yang, Scott Peters, Thomas Whelan, Chen Kong, Omkar Parkhi, Richard Newcombe, and Yuheng Carl Ren. Aria digital twin: A new benchmark dataset for egocentric 3d machine perception. In *ICCV*, 2023. 2

[26] Christian Richardt, Hyeongwoo Kim, Levi Valgaerts, and Christian Theobalt. Dense wide-baseline scene flow from two handheld video cameras. In *3DV*, 2016. 2

[27] Jenny Seidenschwarz, Qunjie Zhou, Bardienus Duisterhof, Deva Ramanan, and Laura Leal-Taixé. DynOMo: Online point tracking by dynamic online monocular gaussian reconstruction. In *3DV*, 2025. 3

[28] Zachary Teed and Jia Deng. RAFT-3D: Scene flow using rigid-motion embeddings. In *CVPR*, 2021. 2

[29] Mel Vecerik, Carl Doersch, Yi Yang, Todor Davchev, Yusuf Aytar, Guangyao Zhou, Raia Hadsell, Lourdes Agapito, and

Jon Scholz. RoboTAP: Tracking arbitrary points for few-shot visual imitation. In *ICRA*, 2024. 8

[30] Sundar Vedula, Simon Baker, Peter Rander, Robert Collins, and Takeo Kanade. Three-dimensional scene flow. In *ICCV*, 1999. 1, 2

[31] Christoph Vogel, Konrad Schindler, and Stefan Roth. 3d scene flow estimation with a rigid motion prior. In *ICCV*, 2011. 2

[32] Bo Wang, Jian Li, Yang Yu, Li Liu, Zhenping Sun, and Dewen Hu. SceneTracker: Long-term scene flow estimation network. *IEEE TPAMI*, 2025. 6, 1

[33] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. VGGT: Visual geometry grounded transformer. In *CVPR*, 2025. 2, 3

[34] Qianqian Wang, Yen-Yu Chang, Ruojin Cai, Zhengqi Li, Bharath Hariharan, Aleksander Holynski, and Noah Snavely. Tracking everything everywhere all at once. In *ICCV*, 2023. 2

[35] Qianqian Wang, Vickie Ye, Hang Gao, Weijia Zeng, Jake Austin, Zhengqi Li, and Angjoo Kanazawa. Shape of Motion: 4d reconstruction from a single video. In *ICCV*, 2025. 2, 3, 5, 6, 8, 1, 4

[36] Ruicheng Wang, Sicheng Xu, Cassie Dai, Jianfeng Xiang, Yu Deng, Xin Tong, and Jiaolong Yang. MoGe: Unlocking accurate monocular geometry estimation for open-domain images with optimal training supervision. In *CVPR*, 2025. 5

[37] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. DUSt3R: Geometric 3d vision made easy. In *CVPR*, 2024. 1, 2, 3, 4, 5, 6

[38] Yuxi Xiao, Qianqian Wang, Shangzhan Zhang, Nan Xue, Sida Peng, Yujun Shen, and Xiaowei Zhou. SpatialTracker: Tracking any 2d pixels in 3d space. In *CVPR*, 2024. 2, 3, 4, 5, 6, 8, 1

[39] Yuxi Xiao, Jianyuan Wang, Nan Xue, Nikita Karaev, Yuri Makarov, Bingyi Kang, Xing Zhu, Hujun Bao, Yujun Shen, and Xiaowei Zhou. SpatialTrackerV2: 3d point tracking made easy. In *ICCV*, 2025. 2, 5, 6

[40] Zhen Xu, Yinghao Xu, Zhiyuan Yu, Sida Peng, Jiaming Sun, Hujun Bao, and Xiaowei Zhou. Representing long volumetric video with temporal gaussian hierarchy. *ACM TOG*, 2024. 1, 4

[41] Gengshan Yang and Deva Ramanan. Upgrading optical flow to 3d scene flow through optical expansion. In *CVPR*, 2020. 2

[42] Emilie Yu, Kevin Blackburn-Matzen, Cuong Nguyen, Oliver Wang, Rubaiat Habib Kazi, and Adrien Bousseau. VideoDoodles: Hand-drawn animations on videos with scene-aware canvases. *ACM TOG*, 2023. 2, 3

[43] Bowei Zhang, Lei Ke, Adam W Harley, and Katerina Fragkiadaki. TAPIP3D: Tracking any point in persistent 3d geometry. In *ICCV*, 2025. 2, 5, 6

[44] Chengwei Zheng, Lixin Xue, Juan Zarate, and Jie Song. GSTAR: Gaussian surface tracking and reconstruction. In *CVPR*, 2025. 2

[45] Yang Zheng, Adam W. Harley, Bokui Shen, Gordon Wetzstein, and Leonidas J. Guibas. Pointodyssey: A large-scale synthetic dataset for long-term point tracking. In *ICCV*, 2023. 2