Multi-View 3D Point Tracking

Supplementary Material

A. Depth Estimation Analysis

Robustness to depth quality. Our method is designed to be tolerant of moderate noise in estimated depth. As shown in Fig. A.1, tracking performance remains stable under additive Gaussian noise up to $\sigma = 2$ cm. This robustness stems from extensive training augmentations and from the transformer's ability to interpolate and re-identify points even when partial depth information is missing or corrupted.

Depth source comparisons. Tab. A.1 summarizes performance across different depth sources, including sensor depth, optimization-based reconstruction, and learned estimators (DUSt3R and VGGT). Our method achieves strong performance with noisy but plausible depth maps from learned estimators, particularly VGGT, but degrades significantly when depth estimation produces misaligned geometry or fails.

DUSt3R alignment and runtime. To align DUSt3R [37] reconstructions to our camera setup, we globally optimize the pairwise maps using its built-in alignment (following Sec. 3.4 in DUSt3R), while keeping our camera intrinsics and extrinsics fixed. This reconstruction step takes 300 iterations and runs at 0.17 FPS. VGGT offers a significantly faster alternative at 3.1 FPS.

Discussion. Sparse-view reconstruction remains an open problem. In our evaluations, DUSt3R was the most reliable method for producing usable depth maps, but even it fails on some scenes. As depth estimation continues to improve, particularly with recent advances in foundation models for geometry (e.g., MegaSAM [20], Easi3R [4]), MVTracker can directly benefit (with or without retraining). Fig. A.2 visualizes our predicted tracks under different depth sources, and Tab. A.1 quantifies performance across depth sources.

Table A.1. **AJ for different depth sources.** GT*: optimization-based depth. D: DUSt3R. V: VGGT. S: Sensor. ✗: depth estimation failed.

Method	Panoptic Studio			DexYCB			MV-Kubric		
	GT*	D	V	S	D	V	GT	D	V
DELTA	68.1	✗	1.8	54.6	36.8	21.1	57.4	25.7	1.9
SpaTracker	61.5	✗	10.6	60.9	58.3	33.2	65.5	61.7	3.9
Triplane Baseline	65.1	✗	29.7	68.0	57.5	61.8	74.7	**70.7**	46.7
MVTracker (ours)	**86.0**	✗	**47.7**	**82.7**	**71.6**	**68.0**	**81.4**	70.0	**48.7**

B. Inference Speed Comparison

We evaluate the runtime of various methods by measuring frames per second (FPS) on a representative test configuration. Tab. B.1 summarizes the results, excluding depth-estimation time. For *optimization-based* methods such as Shape of Motion [35] and Dynamic 3D Gaussians [23], the per-sequence runtime is significantly higher (*e.g.*, about 30 minutes for Shape of Motion and about 50 minutes for Dynamic 3DGS on a single Panoptic Studio [18] sequence). Overall, the optimization-based approaches are orders of magnitude slower, making them less practical for large-scale applications and unusable for real-time applications. In contrast, our feed-forward model runs online at a comparable speed to the triplane-based baseline while outperforming it in accuracy, highlighting the efficiency of our kNN correlation in 3D point clouds. Note that we correctly adjust the effective FPS by taking into account the window size for sliding-window-based methods.

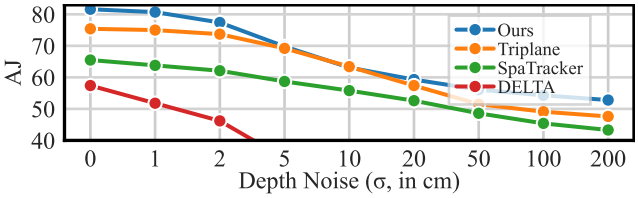

Figure A.1. **Robustness to depth noise $\mathcal{N}(0, \sigma^2)$ on MV-Kubric.**

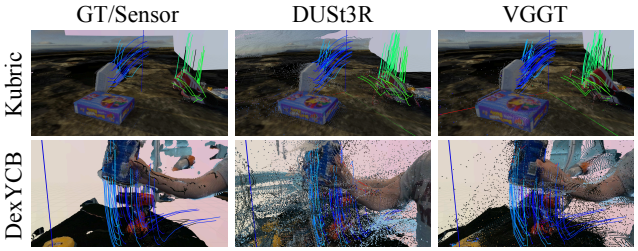

Figure A.2. **Different depth sources along our predicted tracks.**

Table B.1. **Runtime Comparison (FPS).** We measure frames per second for each method under a representative test setting, excluding depth-estimation overhead. Optimization-based methods are indicated with placeholders for their much lower throughput.

Method	**FPS**
Dynamic 3DGS [23]	*(optimization-based)*
Shape of Motion [35]	*(optimization-based)*
SpaTracker [38]	1.4
DELTA [24]	1.5
SceneTracker [32]	3.5
CoTracker2 [16]	5.7
Triplane Baseline	5.8
MVTracker (ours)	7.2
CoTracker3 [17]	18.4

Table C.1. **Impact of Different Depth Sources** on DexYCB. We report results using either DUSt3R-estimated [37] or sensor-based depth for all methods. Sensor depth generally improves performance across the board, underscoring the importance of high-fidelity depth. Notably, our multi-view tracker benefits the most, indicating that our fused 3D point cloud representation effectively capitalizes on better depth quality.

Method	AJ↑	δ_{avg}↑	OA↑	MTE↓
DexYCB [3] (DUSt3R depth)				
LocoTrack [5]	27.8	38.6	77.0	22.8
DELTA [24]	36.8	51.6	61.0	18.3
CoTracker2 [16]	28.8	40.5	76.2	20.8
CoTracker3 [17]	29.4	40.3	78.6	22.0
SpaTracker [38]	58.3	72.0	80.2	5.9
SpaTrackerV2 [39]	35.5	45.1	91.3	9.8
TAPIP3D [43]	38.8	50.0	90.1	8.2
Triplane Baseline	57.5	71.0	81.3	4.3
MVTracker (ours)	**71.6**	**80.6**	**91.3**	**2.0**
DexYCB [3] (sensor depth)				
LocoTrack [5]	55.1	64.6	76.7	27.5
DELTA [24]	54.6	67.8	72.6	42.6
CoTracker2 [16]	56.4	67.4	76.4	26.4
CoTracker3 [17]	57.3	68.5	76.7	23.4
SpaTracker [38]	60.9	75.0	82.5	4.0
SpaTrackerV2 [39]	69.2	77.5	91.3	4.2
TAPIP3D [43]	77.2	88.9	88.0	1.3
Triplane Baseline	68.0	79.5	85.9	2.6
MVTracker (ours)	**82.7**	**91.0**	**91.6**	**0.8**

C. Ablation Study

C.1. Impact of Different Depth Sources

Table C.1 reports the performance of our model and several baselines on DexYCB when using two different depth sources: DUSt3R [37] (estimated) and sensor-based depth. We observe that sensor-based depth leads to improved accuracy for all methods, highlighting the importance of depth quality. Notably, our multi-view tracker benefits substantially from sensor depth, surpassing single-view methods and the triplane baseline by a larger margin. This suggests that our fused 3D point cloud representation is robust to a wide range of depth sources, yet can further leverage higher-fidelity depth to achieve stronger 3D point tracking.

C.2. Varying the Number of Input Views

Tab. C.2 shows detailed per-dataset results for a varied number of input views. Our tracker consistently improves as more views are added.

Table C.2. **Varying the Number of Input Views.** Evaluation of AJ under a varied number of input views.

Method	Number of views							
	1	2	3	4	5	6	7	8
Panoptic Studio [18]								
LocoTrack [5]	54.7	56.4	61.4	65.8	65.8	67.3	66.4	66.8
DELTA [24]	60.9	61.8	63.7	68.1	66.7	67.4	67.6	66.3
CoTracker2 [16]	59.5	60.7	64.4	69.5	68.8	69.3	68.4	69.9
CoTracker3 [17]	61.9	65.0	68.5	74.5	74.0	75.2	75.1	76.0
SpaTracker [38]	51.2	52.1	57.5	61.5	60.9	61.4	62.6	63.4
SpaTrackerV2 [39]	57.4	62.7	66.8	72.4	71.8	72.7	72.7	72.9
TAPIP3D [43]	**68.8**	**72.9**	78.5	84.3	84.6	85.6	86.0	86.8
Triplane Baseline	61.9	60.9	61.6	65.0	64.0	65.1	66.8	65.6
MVTracker (ours)	66.4	72.2	**79.0**	**86.0**	**87.4**	**88.8**	**89.1**	**89.9**
DexYCB [3] (DUSt3R depth)								
LocoTrack [5]	27.9	26.0	28.1	27.8	36.3	34.8	34.7	34.9
DELTA [24]	33.0	34.3	38.0	36.5	37.2	35.4	34.9	35.7
CoTracker2 [16]	29.8	26.4	29.2	28.8	37.8	36.2	36.0	36.0
CoTracker3 [17]	28.6	27.0	29.5	29.4	39.1	37.5	37.1	37.3
SpaTracker [38]	60.6	58.4	61.8	58.3	63.2	62.4	62.9	63.4
SpaTrackerV2 [39]	39.8	39.5	36.5	35.5	41.1	37.1	37.0	37.7
TAPIP3D [43]	36.6	35.6	40.5	38.8	57.7	54.2	55.2	56.4
Triplane Baseline	44.0	48.0	56.0	57.6	63.5	64.5	65.5	66.8
MVTracker (ours)	**64.0**	**66.8**	**73.2**	**71.1**	**77.4**	**76.7**	**77.3**	**79.2**
Multi-View Kubric [11]								
LocoTrack [5]	57.3	56.1	53.8	52.5	52.3	52.1	51.6	51.8
DELTA [24]	65.9	61.9	58.9	57.4	56.3	56.0	54.8	54.1
CoTracker2 [16]	61.4	59.1	56.1	54.6	53.8	53.7	53.1	52.8
CoTracker3 [17]	60.1	58.7	56.5	55.1	54.8	54.6	54.2	54.3
SpaTracker [38]	72.8	70.3	67.4	65.5	64.8	64.5	63.1	62.7
SpaTrackerV2 [39]	54.0	57.6	57.6	58.6	58.3	59.1	58.8	59.7
TAPIP3D [43]	76.4	75.4	72.8	72.5	71.7	71.4	70.9	70.6
Triplane Baseline	71.7	73.4	74.1	74.7	75.0	75.2	75.2	75.7
MVTracker (ours)	**81.7**	**81.2**	**81.4**	**81.4**	**81.9**	**82.1**	**82.3**	**82.5**

C.3. 2D Tracking Accuracy

Tab. C.3 compares the performance of 2D point tracking methods when their outputs are projected into 2D using standard metrics [7]. Although 2D methods do not rely on estimated depth quality, they generally achieve lower location accuracies compared to our multi-view approach, which fuses multi-view data to enhance performance. On DexYCB, we only report results using sensor depth maps to provide a more informative comparison in 2D point tracking. For results with depth estimated by DUSt3R, where single-view methods benefit from not having to handle noisy or incomplete depth maps, see Tab. C.4. Note that such depth noise or incompleteness affects our 2D tracking more than that of SpaTracker, due to SpaTracker's XY-aligned feature plane compared to our fused 3D point cloud.

Table C.3. **2D Tracking Evaluation.** Evaluation of projected 2D tracking performance using standard 2D metrics [7]. Multi-view outputs are projected onto four views and the metrics are averaged across these views. Note that we can only report location accuracies since our multi-view methods report any-view visibility (not per-view).

Method	$\delta^{2D}_{avg}\uparrow$	$\delta^{2D}_{<1}\uparrow$	$\delta^{2D}_{<2}\uparrow$	$\delta^{2D}_{<4}\uparrow$	$\delta^{2D}_{<8}\uparrow$	$\delta^{2D}_{<16}\uparrow$
Panoptic Studio [18]						
LocoTrack [5]	66.7	29.3	51.4	72.6	86.5	93.7
DELTA [24]	70.0	**34.8**	56.5	76.5	87.6	94.8
CoTracker2 [16]	64.3	28.0	49.3	69.7	82.9	91.4
CoTracker3 [17]	70.9	33.1	**56.8**	78.2	90.5	96.2
SpaTracker [38]	66.4	29.3	51.8	71.8	85.7	93.3
Triplane Baseline	51.9	8.5	24.1	52.0	80.3	94.5
MVTracker (ours)	**70.5**	26.6	52.9	**79.5**	**94.6**	**98.9**
DexYCB [3] (sensor depth)						
LocoTrack [5]	83.7	74.1	78.4	84.0	89.4	92.4
DELTA [24]	84.6	75.7	79.9	85.2	89.4	92.8
CoTracker2 [16]	83.7	73.7	78.9	84.5	89.3	92.1
CoTracker3 [17]	84.1	74.8	79.0	84.5	89.4	92.9
SpaTracker [38]	85.8	**75.9**	80.7	86.1	91.2	94.9
Triplane Baseline	75.9	52.2	70.4	78.9	85.9	92.0
MVTracker (ours)	**87.5**	73.3	**81.2**	**88.3**	**95.5**	**99.2**
Multi-View Kubric [11]						
LocoTrack [5]	75.3	53.6	66.8	78.0	86.1	91.9
DELTA [24]	78.4	61.9	72.0	80.3	86.5	91.2
CoTracker2 [16]	72.6	51.8	63.7	74.2	83.1	90.2
CoTracker3 [17]	74.8	54.2	66.3	76.8	85.4	91.5
SpaTracker [38]	75.8	51.4	65.8	79.3	88.5	94.0
Triplane Baseline	80.6	51.3	72.4	87.3	94.5	97.7
MVTracker (ours)	**86.8**	**61.2**	**81.7**	**94.0**	**97.9**	**99.2**

Table C.4. Evaluation of projected 2D tracking performance using standard 2D metrics [7] on DexYCB with estimated depths. Single-view methods only require RGB for tracking and do not depend on the quality of the depth for achieving high 2D point tracking performance. However, the multi-view methods face challenges when dealing with the noisy depth estimates on DexYCB.

Method	$\delta^{2D}_{avg}\uparrow$	$\delta^{2D}_{<1}\uparrow$	$\delta^{2D}_{<2}\uparrow$	$\delta^{2D}_{<4}\uparrow$	$\delta^{2D}_{<8}\uparrow$	$\delta^{2D}_{<16}\uparrow$
DexYCB [3] (DUSt3R depth)						
LocoTrack [5]	85.4	75.6	80.0	85.8	91.4	94.4
DELTA [24]	85.6	**77.0**	81.2	86.0	90.2	93.7
CoTracker2 [16]	86.2	75.7	81.3	87.1	92.1	95.0
CoTracker3 [17]	**86.7**	**77.0**	**81.5**	**87.2**	**92.1**	**95.7**
SpaTracker [38]	86.0	76.4	80.8	86.2	91.7	95.1
Triplane Baseline	64.3	31.8	53.6	71.4	79.3	85.4
MVTracker (ours)	71.3	42.1	59.8	76.0	85.2	93.5

Table C.5. **Training Augmentation** of the variable number of input views ranging from 1 to 8 (V) and varying depth sources between ground-truth and off-the-shelf depth estimation (D) during training on tracking performance.

V	D	AJ ↑	δ_{avg} ↑	OA ↑	MTE ↓
Panoptic Studio [18]					
✓		80.4	**94.4**	87.8	**3.1**
	✓	**80.7**	93.3	**88.7**	3.5
✓	✓	**80.7**	94.3	88.2	3.2
DexYCB [3]					
✓		49.9	72.4	70.5	3.6
	✓	62.3	77.1	82.1	3.2
✓	✓	**65.2**	**79.5**	**82.9**	**2.3**
Multi-View Kubric [11]					
✓		80.2	**91.1**	91.1	**0.7**
	✓	77.2	88.0	91.0	0.8
✓	✓	**80.4**	90.5	**92.1**	**0.7**

Table C.6. **Training Augmentation** with evaluation on different numbers of views. We investigate the impact of using a variable number of views ranging from 1 to 8 (V) and varying depth sources between ground-truth and off-the-shelf depth estimation (D) during training on tracking performance. Results are reported in AJ.

V	D	Number of views 1	2	3	4	5	6	7	8
Panoptic Studio [18]									
✓		**69.3**	72.0	76.9	80.4	81.4	**84.4**	84.8	85.1
	✓	67.0	71.3	**77.2**	**80.7**	**81.9**	84.2	**84.9**	**85.8**
✓	✓	68.1	**73.1**	76.9	**80.7**	81.5	83.4	84.4	84.3
DexYCB [3] (DUSt3R depth)									
✓		37.6	40.0	48.1	49.9	65.6	67.4	69.6	73.4
	✓	51.6	54.1	60.7	62.3	68.8	69.9	71.8	74.8
✓	✓	**56.3**	**58.9**	**64.9**	**65.2**	**74.0**	**75.4**	**76.3**	**78.9**
Multi-View Kubric [11]									
✓		**80.7**	**79.9**	**80.1**	80.2	80.9	81.4	**81.8**	81.9
	✓	75.9	76.3	76.6	77.2	77.2	78.0	78.1	78.4
✓	✓	80.4	79.6	**80.0**	**80.4**	**81.1**	**81.6**	**81.8**	**82.0**

C.4. Training Augmentation

Tab. C.5 and C.6 evaluate the impact of training augmentation strategies. We consider two factors: (V) employing a variable number of views during training (ranging from 1 to 8) and (D) varying the source of depth maps (ground-truth versus off-the-shelf depth estimation). The results reveal that combining both variable view and depth augmentations leads to the best performance, especially as measured on DexYCB.

We apply a range of augmentations to improve gener-

Table D.1. **Dynamic 3D Gaussians.** Novel view synthesis scores for Dynamic 3DGS [23] under varying numbers of cameras on Panoptic Studio [18].

	Number of Cameras			
	27	17	9	4
PSNR	28.5	23.7	17.9	12.6
SSIM	0.91	0.83	0.74	0.52

alization and robustness. These include photometric augmentations such as color jitter, Gaussian blur, and occlusion simulation via erasing and region replacement. Spatial augmentations involve random cropping, padding, flipping, and scaling. Depth perturbations include noise, rescaling, and occlusion-based erasure. Scene-level augmentations transform the world coordinate system through random rotation, translation, and scaling. Camera intrinsics and extrinsics are perturbed to simulate realistic variations. Additionally, we vary the number of tracks per sample, randomly sample the number of input views (between 1 and 6), and use either ground-truth depths or depths from an estimation method to improve model adaptability.

D. Baselines

Dynamic 3D Gaussians. We use the original training code provided by the authors of the Dynamic 3D Gaussians [23]. The training process utilizes camera intrinsics, extrinsics, segmentation masks, images, and an initial point cloud. This initial point cloud is constructed from depth maps corresponding to selected viewpoints at the initial timestep. To differentiate between static and dynamic points within the point cloud, we leverage the provided segmentation masks.

For tracking, we identify the most influential Gaussian for a given point at time t based on the influence computation described in the Dynamic 3DGS paper. To establish a trajectory, we then track the motion of this Gaussian across all time steps. The visibility of each point at time t is determined by comparing the depth of the Gaussian with the corresponding depth value of the depth map rendered from each camera viewpoint. Trajectories that show sudden discontinuities or have a dominant Gaussian influence near zero for the query point are classified as static to ensure consistency in motion.

To evaluate the model's performance in novel view synthesis, we conducted experiments on Panoptic Studio. We trained different models using a varying number of cameras. Then, we render images using these trained models from four test cameras. Our results in Tab. D.1 show that with 27 input views, we achieve reconstruction quality comparable to that reported in the original paper. However, as the number of views decreases, reconstruction quality reduces.

Shape of Motion. We adapt the monocular training pipeline of the original Shape of Motion [35] to a multi-view setting. The method requires segmentation masks for moving objects, depth maps, long-range 2D tracks, and camera poses. In our multi-view adaptation, we rely on the available depth maps, segmentation masks, and camera poses, while using TAPIR [8] to obtain long-range 2D tracks, as in the original approach.

For both static and dynamic part initialization, we use multi-view depth maps and segmentation masks to construct an initial 3D representation that captures the overall scene structure more effectively than the monocular approach. We follow the original paper's strategy for choosing the canonical frame and optimizing motion bases. Additionally, we adapt the monocular loss function to a multi-view formulation, computing the original loss function separately for each view and averaging the results across all views. This adjustment ensures that the optimization effectively integrates multi-view information.

E. Evaluation on TAPVid-2D

In Tab. E.1, we evaluate MVTracker on the TAPVid-2D benchmark by projecting our predicted 3D trajectories onto image views. As our method requires depth input, we report results using various monocular depth estimators. However, tracking performance is significantly degraded in this benchmark, which is in part attributable to TAPVid-2D containing outdoor unbounded scenes, our scene normalization not aligning scenes well to the training distribution, and monocular depth estimation frequently failing or flickering.

F. Evaluation Metrics

There is no established set of metrics for multi-view point tracking in 3D. Thus, we adopt four metrics from prior monocular point-tracking benchmarks [7], extending them to our multi-view 3D setting. These metrics measure the quality of the predicted 3D trajectories (δ_{avg}, MTE), the ability to predict whether a point is visible in any of the views (OA), or both simultaneously (AJ). We compute all metrics per-trajectory before averaging across all tracks in a scene, and then across all scenes in a dataset.

Let N be the number of tracked points, T the number of frames, and $\hat{p}_t^i$ (resp. p_t^i) the predicted (resp. ground-truth) 3D location of track i at time t. Similarly, let $\hat{v}_t^i \in \{0, 1\}$ and $v_t^i \in \{0, 1\}$ be the predicted and ground-truth visibility flags.

Median Trajectory Error (MTE). For each track i, the median trajectory error measures the median distance between predicted and ground-truth locations over timesteps

Table E.1. **TAPVid-2D Evaluation.** Evaluation results in 2D point tracking on TAPVid-2D [7]. Since our method requires depth as input, we report results for several monocular depth estimators. However, performance is far from satisfactory for most trajectories, which is in part attributable to out-of-distribution outdoor scenes with many far-away points, scenes not being normalized well to the training scale, failures in depth estimation, flickering in the estimated video depth, etc. We report both the original TAPVid-2D metrics as well as the metrics used elsewhere in this paper; the difference is that our metrics take the average of per-trajectory metrics, whereas TAPVid-2D computes each metric over all points, treating all trajectories as one trajectory. All metrics are in pixel scale (*e.g.*, $\delta^{2D}_{<4}$ is the within-a-radius-of-4-pixels location accuracy, MTE is in pixels, etc.). Best and second-best results per metric are **bold** and underlined, respectively.

Method	Depth	AJ^{2D}	$AJ^{2D}_{<1}$	$AJ^{2D}_{<2}$	$AJ^{2D}_{<4}$	$AJ^{2D}_{<8}$	$AJ^{2D}_{<16}$	δ^{2D}_{avg}	$\delta^{2D}_{<1}$	$\delta^{2D}_{<2}$	$\delta^{2D}_{<4}$	$\delta^{2D}_{<8}$	$\delta^{2D}_{<16}$	OA	MTE
		TAPVid-2D Metrics													
MVTracker (ours)	ZoeDepth [1]	9.5	1.6	3.9	8.5	14.3	19.4	20.4	3.0	7.5	16.7	29.7	45.3	46.5	–
MVTracker (ours)	MoGe [36]	13.1	2.0	4.4	9.5	19.5	30.0	25.0	3.4	8.0	18.4	37.5	57.6	54.6	–
MVTracker (ours)	MegaSAM [20]	31.6	5.7	14.4	31.3	47.4	59.1	46.0	11.0	25.3	47.8	66.8	79.1	78.6	–
CoTracker3 [17]	none required	**64.1**	**28.2**	**51.3**	**72.2**	**82.4**	**86.4**	**77.0**	**42.1**	**67.1**	**85.3**	**93.3**	**97.0**	**91.0**	–
		Our Metrics (where numbers are computed per track and then averaged)													
MVTracker (ours)	ZoeDepth [1]	10.3	1.8	4.4	9.5	15.6	20.4	21.8	3.3	8.4	18.3	31.9	47.3	46.1	59.1
MVTracker (ours)	MoGe [36]	14.6	2.4	5.4	11.6	22.0	31.6	26.7	3.9	9.2	20.5	40.4	59.7	55.1	86.0
MVTracker (ours)	MegaSAM [20]	35.0	7.4	17.4	35.7	51.9	62.6	46.4	12.1	26.2	48.1	66.7	78.9	79.8	14.0
CoTracker3 [17]	none required	**66.7**	**32.6**	**56.2**	**74.9**	**83.3**	**86.4**	**77.2**	**43.2**	**67.6**	**85.3**	**93.1**	**96.7**	**91.7**	**2.9**

where the track is visible:

$$MTE_i \;=\; \text{median}\Big\{\, \|\hat{p}^i_t - p^i_t\|_2 \;\big|\; t = 1, \ldots, T,\; v^i_t = 1 \Big\}. \tag{F.1}$$

The overall MTE is the mean across all N tracks:

$$MTE \;=\; \tfrac{1}{N} \sum_{i=1}^{N} MTE_i. \tag{F.2}$$

Occlusion Accuracy (OA). Occlusion accuracy quantifies how well the model predicts visibility:

$$\text{OA}_i \;= \tfrac{1}{T} \sum_{t=1}^{T} \mathbb{1}\big(\hat{v}^i_t \;=\; v^i_t\big), \tag{F.3}$$

$$\text{OA} \;= \tfrac{1}{N} \sum_{i=1}^{N} \text{OA}_i. \tag{F.4}$$

Average Location Accuracy (δ_{avg}). We evaluate the fraction of points within each of H distance thresholds $\{x_1, \ldots, x_H\}$ in centimeters. For a single threshold x, define the following:

$$\delta^i_x \;=\; \tfrac{1}{\sum_{t=1}^{T} v^i_t} \sum_{t=1}^{T} \mathbb{1}\Big(v^i_t = 1\Big)\mathbb{1}\Big(\|\hat{p}^i_t - p^i_t\|_2 < x\Big). \tag{F.5}$$

We then average over all tracks and thresholds:

$$\delta_x \;=\; \tfrac{1}{N} \sum_{i=1}^{N} \delta^i_x, \quad \delta_{\text{avg}} \;=\; \tfrac{1}{H} \sum_{h=1}^{H} \delta_{x_h}. \tag{F.6}$$

Average Jaccard (AJ). This metric jointly assesses spatial and occlusion accuracy at each threshold x. Define $\alpha^i_{t,x} = \mathbb{1}\big(\|\hat{p}^i_t - p^i_t\|_2 < x\big)$. Then, for track i:

$$AJ^i_x \;=\; \frac{\sum_{t=1}^{T} v^i_t \hat{v}^i_t \alpha^i_{t,x}}{\sum_{t=1}^{T} \big(v^i_t + (1 - v^i_t)\hat{v}^i_t + v^i_t \hat{v}^i_t (1 - \alpha^i_{t,x})\big)}. \tag{F.7}$$

Averaging over N tracks yields

$$AJ_x \;=\; \tfrac{1}{N} \sum_{i=1}^{N} AJ^i_x, \quad AJ \;=\; \tfrac{1}{H} \sum_{h=1}^{H} AJ_{x_h}. \tag{F.8}$$

F.1. Evaluation Details

For MV-Kubric and DexYCB, we sample 512 points uniformly from object surfaces and report metrics averaged across static and dynamic points. Panoptic Studio often lacks labeled static objects, so we sample 512 points and compute metrics over all of them. We use four views in the main evaluation tables: for DexYCB, these are the first four calibrated cameras; for Panoptic Studio, we select distant views; and for MV-Kubric, views are randomly sampled per scene. The sensitivity to the selected views is analyzed in Tab. 3. Jaccard and location accuracy thresholds are set to 1/2/5/10/20 cm for DexYCB, 0.65/1.3/2.6/5.2/10.4 simulation-scale-adjusted centimeters for MV-Kubric (1 Kubric unit is about 13 cm before simulation-scale adjustment), and 5/10/20/40 cm for Panoptic Studio.

Panoptic Studio trajectories were generated by merging per-view monocular labels from TAPVid-3D [7], filtered from Dynamic 3DGS [23] 27-view predictions. These la-

bels are often noisy and erroneous, and for example, exhibit point drift (*e.g.*, labels jumping from an elbow to a finger), which is why we use higher evaluation thresholds on this benchmark and recommend interpreting results as well as using the benchmark with caution. MV-Kubric labels are derived from simulated ground-truth trajectories, while DexYCB tracks are extracted from fitted object meshes provided with the dataset. For DexYCB, we sample points on annotated surfaces and track their position using barycentric coordinates of the deforming mesh triangles. Visibility labels on DexYCB can still be noisy due to imperfect Kinect depth maps and segmentation boundaries. The scripts used to process all datasets and generate labels are publicly available in our codebase.