DynOMo: Online Point Tracking by Dynamic Online Monocular Gaussian Reconstruction

Supplementary Material

In this supplementary material, we first provide additional details regarding our method in Sec. 6 and our implementation in Sec. 7. We further present 2D and 3D point tracking results on the iPhone dataset [10] in Sec. 8 and provide more insights on our design choices with additional ablation studies in Sec. 9, followed by discussions and visualizations of our failure cases in Sec. 10. Finally, we provide additional visualizations of our emergent trajectories and 2D point tracking. We will release our code upon acceptance.

6. Method Details

Densification Mask. In our approach we exploit the densification mask process from [14]. This means, we only add new Gaussians for unobserved regions based on a densification mask $M_D(p)$ computed by applying a threshold on pixel densities, *i.e.*,

$$M_D(p) = (\sum_{i \in H} T_i \alpha_i) < 0.5, \tag{11}$$

Physically-Based 3D Regularization We build on the physically-based priors from [24], *i.e.*, the rigidity $\mathcal{L}_{rigid}$, isometry $\mathcal{L}_{iso}$, and rotation $\mathcal{L}_{rot}$ losses. All losses are applied to every Gaussian directly and computed over its k nearest neighbors (kNN):

$$\mathcal{L}_x = \frac{1}{k|G|} \sum_{i \in |G|} \sum_{j \in kNN_i} w_{i,j} \mathcal{L}_{x,i,j} \tag{12}$$

where $x \in \{rigid, rot, iso\}$, $|G|$ is the number of Gaussians, and $w_{i,j}$ is the weight term depending on the Gaussians i and j. While [24] defines $w_{i,j}$ as the l_2 distance in 3D space, we utilize a semantic-based weighting as defined in Sec. 3.3 of the main paper.

The local short-term rigidity loss enforces close-by Gaussians to undergo a similar rigid transformations when observed from Gaussian i's coordinate system. This assumption holds if two Gaussians belong to the same object even for non-rigid objects since it is applied only in a local region. $\mathcal{L}_{rigid}$ is defined by:

$$\mathcal{L}_{rigid,i,j} = ||(\mu_{j,\tau-1} - \mu_{i,\tau-1}) - \Delta R_i (\mu_{j,\tau} - \mu_{i,\tau})||_2 \tag{13}$$

where ΔR_i is the relative rotation of Gaussian i from time step τ to $\tau - 1$ and is defined as $\Delta R_i = R_{i,\tau-1} R_{i,\tau}^{-1}$. The short-term rotation loss further enforces the rotation of close-by Guassians to undergo similar changes by:

$$\mathcal{L}_{rot,i,j} = ||\hat{q}_{j,\tau} \hat{q}_{j,\tau-1}^{-1} - \hat{q}_{i,\tau} \hat{q}_{i,\tau-1}^{-1}||_2 \tag{14}$$

where $\hat{q}$ is the normalized quaternion. While this constraint is implicitly enforced by the rigidity loss, we show that additionally adding rotation regularization leads to a slight performance increase (*cf*. Sec. 4.3 of the main paper). Finally, the long-term isometry loss additionally enforces Gaussians to keep the initial distance to their kNN Gaussians across time and is hence defined by:

$$\mathcal{L}_{iso,i,j} = ||\mu_{j,0} - \mu_{i,0}||_2 - ||\mu_{j,\tau} - \mu_{i,\tau}||_2 \,. \tag{15}$$

We compare the impact of individual loss terms in Sec. 4.3 of the main paper.

Temporal Smoothness Regularization. We apply smoothness regularization to Gaussian features f_i, Gaussian colors c_i and means of the Gaussains of the background region, defined as:

$$\mathcal{L}_{sm,f} = \lambda_f \sum_{i \in |G|} ||f_{i,\tau-1} - f_{i,\tau}||_1 \tag{16}$$

$$\mathcal{L}_{sm,c} = \lambda_c \sum_{i \in |G|} ||c_{i,\tau-1} - c_{i,\tau}||_1 \tag{17}$$

$$\mathcal{L}_{sm,\mu_b} = \lambda_{\mu_b} \sum_{i \in |G_b|} ||\mu_{i,\tau-1} - \mu_{i,\tau}||_1 \tag{18}$$

where $|G|$ is the set of Gaussians, $|G_b|$ is the set of background Gaussians and the final smoothness regularization is given by $\mathcal{L}_{sm} = \mathcal{L}_{sm,f} + \mathcal{L}_{sm,c} + \mathcal{L}_{sm,\mu_b}$.

7. Implementation Details

Gaussian Initialization. We initialize Gaussian with means μ_i by lifting the every second pixel defined by its image coordinates to 3D given the depth map and the camera projection matrix. We initialize the rotation to the unit quaternion $q_i = (1, 0, 0, 0)$ and the scale to be dependent on the distance as $s_i = \frac{z}{0.5 * (f_x + f_y)} * (1, 1, 1)$ where f_x and f_y are the focal lengths in x and y direction of the camera. For each pixel position we also unproject RGB color values $c_i \in R^3$, the feature vectors $f_i \in R^{32}$ as well as the instance id $g_i \in R$. This means, we obtain each of the previous information per-pixel and then assign them to its corresponding the unprojected 3D Gaussain. Since we assume each G_i to be a particle in space, we intialize o_i to be biased towards being visible, *i.e.*, we initialize $o_i = \frac{e^{0.7}}{1+e^{0.7}}$.

Nearest Neighbourhood Selection. For our Gaussian forward propagation as well as for our semantics guided-weighting of the physically-based priors (see Sec. 3.2 and

Sec. 3.4 of the main paper) we require a k nearest neighbor (kNN) set for each Gaussian. We set $k = 20$. To determine $\mathcal{N}_i$, we first compute the 2kNN set based on l_2 distance in 3D space within all Gaussians belonging to the same instance. We then define $\mathcal{N}_i$ to be those Gaussians from the 2kNN with the smallest $s_{i,j}$, *i.e.*, the smallest cosine distance between the Gaussians' feature vectors. We utilize this kNN set for both, Eq. 3 and 7 of the main paper.

Gaussian Optimization. We optimize our dynamic 3DGS model using Adam [16] optimizer with a learning rate of 0.0016 for μ_i, 0.01 for q_i, 0.0005 for o_i, 0.001 for s_i, 0.001 for f_i, 0.0025 for c_i, 0.0001 for g_i, and 0.001 for camera pose. We set the reconstruction loss weights to $\lambda_I = 1.0$, $\lambda_F = 16$, $\lambda_D = 0.1$ and $\lambda_B = 3$. For the physically-based losses, we use the weights to $\lambda_{iso} = 16$, $\lambda_{rot} = 16$, and $\lambda_{rigid} = 128$. We set $\lambda_{sm} = 1$ and the weighting within the smoothness losses as $\lambda_f = 20$, $\lambda_c = 20$, $\lambda_{\mu_b} = 5$. For experiments on Panoptic Sports we run 500 iterations per time step while for all other experiments we run 200 iterations for Gaussian optimization as well as for the camera optimization.

Off-the-shelf Data Preparation. Given any input RGB image, we extract several other types of information from it using off-the-shelf models. We extract its depth image using the metric depth branch from DepthAnything [44], as well as visual feature maps using Dinov2 [29] with ViT small backbone followed by PCA to reduce the feature dimension from 384 to 32. For each image, we generate features for five quadratical, overlapping crops to obtain higher resolution feature maps. We further assume a sparse instance segmentation masks is provided. Those data are used either to initialize the scene or provide optimization supervisions.

Average jaccard (AJ) metrics. **AJ** combines both metrics by computing the fraction of

$$AJ = \sum_{h \in \{1,2,4,8,16\}} \frac{TP_h}{GT + FP_h} \tag{19}$$

where TP_h are the points that lies within the pixel distance threshold h and whose visibility was computed correctly, FP_h are all other predicted points and GT is the number of visible ground truth evaluation points.

8. Comparison on iPhone Dataset

To further comprehensively understand `DynOMo`'s performance, we evaluate it for 2D and 3D point tracking on a third benchmark, the iPhone dataset [10] in Tab. 4.

Dataset. The iPhone dataset [10] contains moving camera, casual captures of real-world scenes recorded using an iPhone. This setting is highly aligned with potential use cases of mixed reality. Compared to Panoptic Sports [24] and TAPVID-Davis [4], the dataset provides rgb images, lidar depth, camera poses of the moving camera, as well as sparse 2D point correspondences across the entire video to evaluate 2D and 3D point tracking.

Metrics. We follow the same evaluation protocol as in [39] where the sparse correspondences are forward and backward tracked across the sequence. 2D point tracking is evaluated using the TAPVID-Davis metrics, *i.e.*, AJ, $\delta_{avg,2D}$ as well as OA. However, instead of using thresholds of $\{1, 2, 4, 8, 16\}$px, the authors use $\{4, 8, 16, 32, 64\}$px. For 3D point tracking, the authors evaluate the end point error (EPE) which is the average l_2 error in 3D reported in m over predicted trajectories. Additionally, they report the fraction of points within $5cm$ and $10cm$ to the ground truth trajectories, *i.e.*, $\delta_{.05,3D}$ and $\delta_{.10,3D}$.

Baselines. Following SOM [39], we use their aligned DepthAnything [44] depth maps. We downscale input images by a factor of 0.5, initialize Gaussians for every pixel position and adapt the visibility threshold accordingly to 0.1. We compare two variants of our method. For one variant, we use the refined camera poses from SOM and for the other one, we use our optimized camera poses. As shown in Tab. 4, we compare ourselves to a set of existing *offline* 2D/3D point trackers. Being highly relevant to our approach, SOM [39] also leverages 3D Gaussians as a dynamic scene representation and could be viewed as an offline version of `DynOMo`. However, the authors optimize their approach utilizing point trajectories extracted by TAPIR [5] as supervision signal and, hence, use correspondence-level supervisory signals similar to CoTracker, TAPIR. In contrast to them, our online tracker `DynOMo` shows emergent motion from online 3D Gaussian reconstruction.

Refined Poses from [39]. Despite the online character of our method and not requiring correspondence-level supervision, `DynOMo` achieves on-par performance with many existing approaches in 2D as well as in 3D. Interestingly, for $\delta_{avg,2D}$ and AJ we even outperform all previous methods. In 3D, we perform better than the purely view-reconstruction approaches, *i.e.*, those approaches that do not require correspondence-level supervision. Additionally, we perform on par with CoTracker[13]+DA[44] underlining `DynOMo`'s ability to generate emergent trajectories.

Optimized Poses. Even if we optimize for camera pose additionally, our EPE is highly competitive despite all other approaches despite them utilizing ground truth or refined camera poses. However, due to inaccuracies in our optimized camera poses we observe a drop in performance especially for the high precision metrics like $\delta_{.05,3D}$. Evaluating our accuracy of our optimized camera poses using ATE RMSE, *i.e.*, average translation error measures in root mean square error we obtain an ATE RMSE of $10.49cm$ mirroring the difficulty of optimizing camera poses for dynamic sequences.

Ablating Aligned Depth and Refined Pose [39]. We

Method	EPE ↓	$\delta_{.05,3D}$ ↑	$\delta_{.10,3D}$ ↑	AJ ↑	$\delta_{avg,2D}$ ↑	OA ↑
HyperNeRF [32]	0.182	28.4	45.8	10.1	19.3	52.0
DynIBaR [22]	0.252	11.4	24.6	5.4	8.7	37.7
Deformable-3D-GS [45]	0.151	33.4	55.3	14.0	20.9	63.9
CoTracker [13]+DA [44]	0.202	34.3	57.9	24.1	33.9	73.0
TAPIR [5]+DA [44]	0.114	38.1	63.2	27.8	41.5	67.4
SOM [39]	**0.082**	**43.0**	**73.3**	34.4	47.0	**86.6**
DynOMo	0.161	33.5	58.1	**35.9**	**58.0**	65.1
DynOMo optimized pose	0.205	20.7	46.0	33.7	54.3	63.9
Ablating Pose and Depth						
DynOMo original pose	0.171	32.1	55.1	35.7	56.7	66.4
DynOMo original lidar	0.198	32.9	53.0	33.2	54.5	65.0

Table 4. **Iphone Dataset:**. We compare the performance of DynOMo using the aligned DepthAnything [44] maps from [39] to other approaches on the Iphone dataset [10]. Note, prior approaches are all offline and mostly require correspondece-level supervisory signal for motion. We show DynOMo leads to emergent motion in 2D as well as in 3D. Additionally, we show that even with camera pose optimization our EPE is highly competitive compared to the other approaches that all use ground truth or refined camera pose information.

Method	AJ ↑	δ_{avg} ↑	OA ↑
DynOMo	45.8	63.1	81.1
Regularization Terms			
w fixing c_i, f_i, and μ_b	30.8	45.3	75.4
w temporal smoothness o_i, s_i, and g_i	38.0	51.7	76.1
w/o temporal smoothness and fixing	35.4	50.1	70.2
Additional Ablations			
isotropic Gaussians	42.7	59.8	79.2
$\mathcal{L}_{emb}$ w l_1 distance	42.1	59.4	79.1
Fixing camera pose	40.7	57.9	77.2

Table 5. **Additional Ablation of Single Part Importance:** In this table, we ablate additional design choices that have less impact on the final performance compared to the ones discussed in Sec. 4.3 of the main paper.

show that utilizing the original LiDAR singal to supervise DynOMo and the original poses provided by the iPhone dataset leads only to a slight performance decrease in 3D as well as in 2D mirroring DynOMo's robustness.

9. Additional Ablation Studies

In this section, we provide additional ablation studies in Tab. 5 to understand our method more comprehensively.

Temporal Smoothness Regularization Terms. We provide additional ablation studies on the temporal smoothness regularization. To recap, in our setting we apply temporal smoothness on f_i, c_i and μ_b while we fix s_i, o_i and g_i over time. In the main paper we show the performance drop of not applying temporal smoothness, *i.e.*, optimize f_i, c_i and μ_b at every time step without additional supervision.We also show the performance drop of not, fixing s_i, o_i and g_i, *i.e.*, also optimizing them at every time step. We now also provide experiments for fixing f_i, c_i and μ_b, applying temporal smoothness on s_i, o_i and g_i as well as not applying temporal smoothness nor fixing values at all (see section *Regularization Terms* in Tab. 5). We observe that all three lead to significant performance degradation: (i) fixing c_i, f_i, and μ_b leads to DynOMo not being able to adapt to, *e.g.*, temporal inconsistencies in image feature prediction or slight color changes due to viewpoint changes; (ii) temporal smoothness terms on o_i, s_i, and g_i can lead to, *e.g.*, Gaussians disappearing due to scale or opacity changes; (iii) not applying any regularization allows the Gaussians to change their attributes freely which allows them to "cheat" to adapt to the a new time frames supervisory signal.

Isotropic vs. Anisotropic Gaussians. Instead of using anisotropic Gaussian distributions, [14] chose to use isptropic Gaussian distributions for their use case of generating maps of static scenes. However, for non-rigid objects, this choice is less suited for two reasons: (i) composing non-rigid objects and motion with isotropic Gaussians, *i.e.*, small spheres, restricts the degrees of freedom to adapt the geometry and geometrical changes; (ii) the rotation of Gaussians actually does not matter for the reconstruction losses, hence, it adds noisy signals in supervision signal. This negatively impacts the rigidity loss, for which the rotation signal is of major importance.

Penalizing Outliers for the Feature Map Reconstruction. Our experiments show that utilizing rendered feature maps leads to significant performance improvement. Rendering feature maps can be seen as a stronger, less ambiguous supervisory signal than RGB colors that pulls the Gaussians to the correct location to match a given time steps observation. We found that utilizing a l_2 distance in computing the feature reconstruction loss penalizes outlier Gaussians in a stronger way (see "$\mathcal{L}_{emb}$ w l_1 distance" in Tab. 5).

Method	$MTE_{2D}\downarrow$	$S_{2D}\uparrow$	$\delta_{avg,2D}\uparrow$	2D %1	2D %8	2D %16	$MTE_{3D}\downarrow$	$S_{3D}\uparrow$	$\delta_{avg,3D}\uparrow$
DynOMo-DA [44]	**3.7**	83.7	59.0	19.6	80.3	88.7	70.7	25.4	0.7
DynOMo-D-3DGS-D	6.3	**85.7**	**61.8**	**25.3**	**83.6**	**90.3**	26.1	73.0	12.7

Table 6. **Impact of Different Depth Predictions:** We compare utilizing DepthAnyhthing [44] metric depth prediction with rendered depth predictions from [24] for the Panoptic Sports dataset [24]. Additionally to the main metrics, we also report the percentage of points within $1, 8$ and 16px distance. We observe that for 2D point tracking the depth prediction approach does not have a major impact on the overall performance. Meanwhile the performance drop is more significant for high precision metrics, *e.g.*, $2D1\%$. Additionally, for 3D point tracking, we observe a severe performance drop with respect to all metrics.

Method	$MTE_{2D}\downarrow$	$Survival_{2D}\uparrow$	$\delta_{avg,2D}\uparrow$	$MTE_{3D}\downarrow$	$Survival_{3D}\uparrow$	$\delta_{avg,3D}\uparrow$
α-composition [39]	7.2	81.9	55.0	32.1	65.3	8.7
Closest 3D Gaussian	9.4	79.6	35.7	26.1	73.0	12.7
DynOMo (Closest 2D Projection)	6.3	85.7	61.8	24.1	71.0	10.1

Table 7. **Choice of Gaussians in Trajectory Estimation:** We compare different approaches for choosing the Gaussian to track. We find that for 2D point tracking, our proposed choice of Gaussians based on the closest 2D Guassian projection achieves the best performance. The performance of α-compositioning lags behind since it chooses Gaussians per timestamp without enforcing those Gaussians to belong to the same point in 3D. When metric depth measurements are available, we choose the closest 3D Guassian directly in 3D, leading to improved 3D tracking performance.

Disentangled Camera Motion. [38] showed that it is not necessary to explicitly model camera motion, but it is sufficient to entangle camera and object motion. However, in Tab. 5 we demonstrate that not explicitly modeling camera motion, *i.e.*, fixing camera pose, leads to a significant performance drop for DynOMo. Additionally, this feature is important for real-world applications that require interaction with the real world.

Different Depth Priors In our online monocular setting, the depth information is directly used for the Gaussian mean initialization as well as for the depth reconstruction loss meanwhile also influences the physics-based loss functions. Therefore, we study the impact of the depth map on our tracking performance by comparing our DynOMo using the zero-shot metric depth predictions from DepthAnything [44] (DA) with using the depth predictions from [24] (D-3DGS-D). Additionally to the metrics from the main paper, we also report the percentage of points within $1, 8$ and 16px distance. As shown in Tab. 6, we only observe slight performance difference using different depth maps for 2D point tracking, since a correct 2D trajectory does not necessarily require a correct 3D trajectory. This performance difference is more observable in high precision metrics, *i.e.*, fraction of points within 1px distance. In contrast, for 3D point tracking, we observe a significant performance increase with better depth estimation, *i.e.*, D-3DGS-D. This suggests that our method directly benefits from future monocular depth estimation advancement.

Choice of Gaussians in Trajectory Estimation. We validate our way of choosing Gaussians for trajectory estimation (as introduced Sec 3.5 of the main paper) by comparing to other ways of choosing Gaussians as introduced by other works. Inspired by [38], SOM[39] computes the 3D trajectory of a corresponding pixel p at time τ by taking the set of Gaussians $H(p)$ into account that intersect with pixel p at τ_s:

$$X_p^{\tau_s \rightarrow \tau} = \sum_{i \in H(p)} T_i \alpha_i \mu_i^\tau \tag{20}$$

where μ_i^τ is the mean of G_i at time τ and $X_p^{\tau_s \rightarrow \tau}$ is the 3D location at time τ corresponding to the trajectory starting from p at τ_s. For 2D point tracking, the authors project $X_p^{\tau_s \rightarrow \tau}$ to the image plane:

$$X_{p,2D}^{\tau_s \rightarrow \tau} = W^\tau X_p^{\tau_s \rightarrow \tau} \tag{21}$$

where W_{τ_s} is the viewing transformation at τ. We compare DynOMo to the above explained approach from [39] denoted as α-composition. Additionally, assuming 3D ground-truth (GT) trajectories are available, instead of choosing a 3D Gaussian for a query pixel based on its closest 2D projection, we directly choose the 3D Gaussian closest to the GT 3D Gaussian for tracking. As shown in Tab. 7, α-composition leads to worse performance in general, as the 3D Gaussians are selected per timestamp and thus can belong to different points in 3D. While choosing the Gaussian based on the 3D means performs slightly better for evaluation in 3D tracking, it's performance in 2D is worse. Furthermore, 3D GT trajectory information is not generally available, *e.g.*, for TAPVID-Davis. We therefore stick to our proposed choice of Gaussian as in Sec 3.5.

Figure 4. **Failure Cases of DynOMo:** DynOMo struggles (i) to track points and add new Gaussians in sequences with extreme occlusions; (ii) to track the camera position as well as the Gaussians in sequences with extreme camera and object motion as well as low background texture; (iii) to track points when extreme acceleration changes occur; (iv) to track Gaussians and camera positions when solely little background is observed but extreme camera motion occurs.

10. Failure Cases

In this section, we discuss and visualize several failure cases of our method in Fig. 4 as well as point out potential solutions. We hope those insights spark future research along this direction to address those limitations.

Extreme Camera Motion. If camera poses are unknown, we estimate the camera poses. However, for extreme camera motion as well as background with little texture, DynOMo struggles to reconstruct the motion appropriately which is also a common failure case in SLAM methods.

Extreme Occlusions. In cases with sudden extreme occlusions, DynOMo struggles to reconstruct the camera pose. Additionally, due to our online monocular character, DynOMo struggles to track points over long-term occlusions. Utilizing a stronger constant velocity assumption or exploiting fine-grained instance id's can potentially help to recover after occlusions.

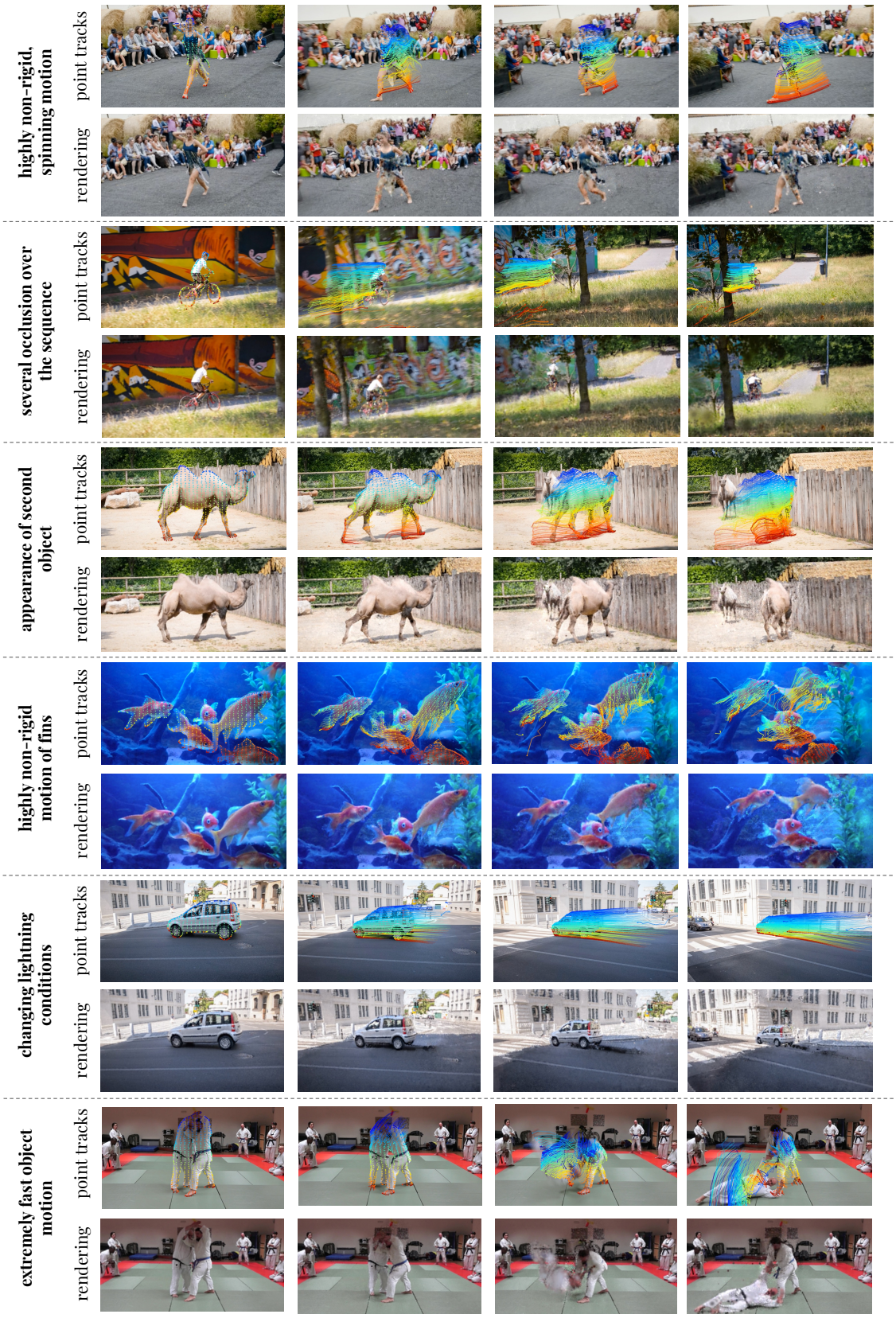

Figure 5. **Visualizations on TAPVID-Davis:** We visualize renderings as well as point tracks on challenging scenes on TAPVID-Davis. DynOMo is able to generate emerging trajectories despite facing non-rigid motion, occlusions, appearance of new objects, or fast motion.

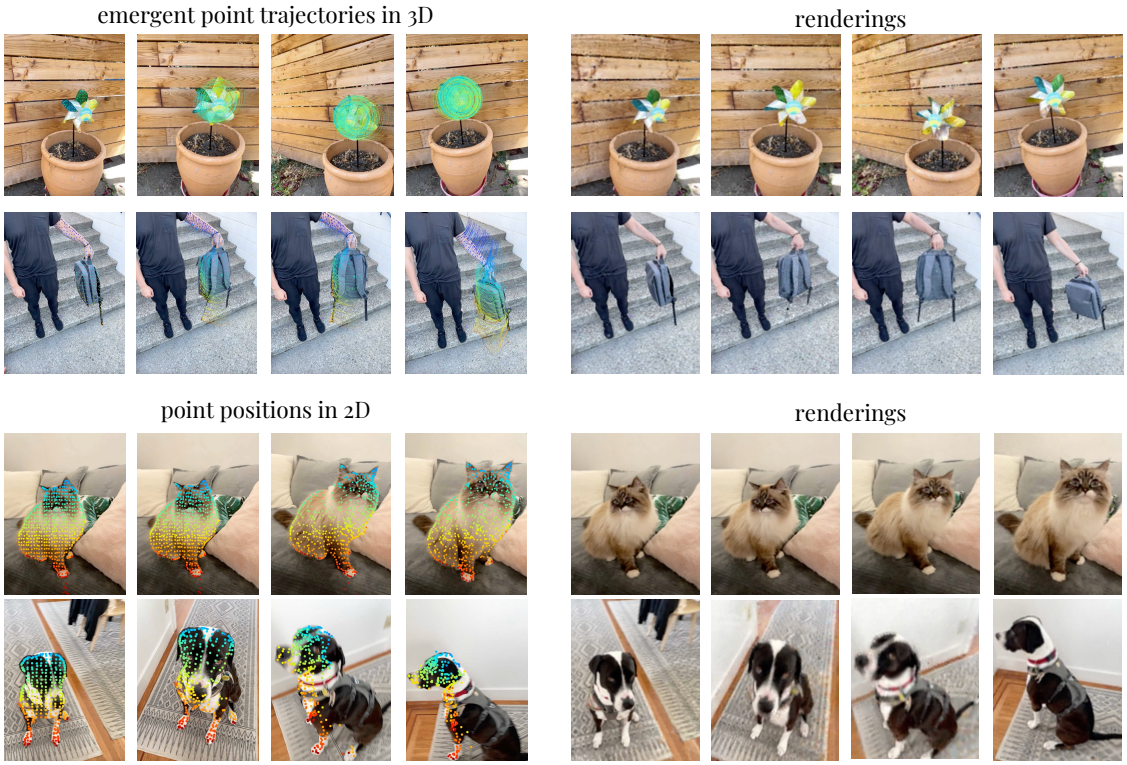

Figure 6. **Visualizations on iPhone Dataset:** We visualize renderings as well as point trajectories and point positions on casual captures from the iPhone dataset. DynOMo is able to generate trajectories for challenging spinning motions and is able to track points in 2D.

Extreme Object Acceleration. For extreme object acceleration, DynOMo struggles to model the sudden change of motion. This is due to the fact that we forward propagate points to the next frame and our constant velocity assumption breaks for sudden extreme velocity changes. Future research could explore utilizing motion prediction models for forward propagation to resolve this issue.

Previously Observed Regions. When adding new Gaussians based on the densification concept, it can happen that a new object enters the scene in front of a previously observed concept. In such cases, no new Gaussians will be added for the newly appearing concept. Towards this end, [14] also exploits depth errors to add Gaussians. However, this requires highly accurate depth prediction, which we do not have access to in, *e.g.*, the TAPVIS-Davis sequences. We assume the advances in depth prediction models will help in resolving this challenge.

Using Depth As Supervisory Signal. Since we are operating in a monocular and online setting, depth predictions are the most important 3D information for DynOMo. However, noisy depth maps can lead to degrading performance when using them directly as supervisory signal, *i.e.*, using a rendered depth loss. Hence, we only exploit it moderately for supervision. Additionally, we add Gaussians by lifting pixel positions to the 3D space given using depth maps. However, since those depth maps are not precisely consistent over time, the Gaussians may be added in a highly incorrect location in the 3D space from which it is highly challenging to recover. As also suggested by our ablation in Tab. 6, the recent advances in depth prediction models will support in solving this issue.

Spinning and Turning Objects. Similarly to adding Gaussians for newly appearing objects, adding Gaussians for a newly appearing side spinning objects is highly challenging since previously existing yet now occluded Gaussians can be re-used to ”cheat”. Exploiting recent approaches in zero-shot mesh prediction could help to populate the whole mesh, *i.e.*, the whole surface of a spinning object with Gaussians.

11. Additional Visualizations

Finally, we present additional visualizations on TAPVID-Davis in Fig. 5 as well as on the iPhone dataset Fig. 6. For the TAPVID-Davis, we visualize emergent trajectories for scenes with occlusions, appearing objects, highly non-rigid motions, changing lightning conditions as well as extremely fast object motion. On the iPhone dataset we show that DynOMo is able to generate emergent trajectories as well as to track points in a robust manner in 2D.