

# DynaGSLAM: Real-Time Gaussian-Splatting SLAM for Online Rendering, Tracking, Motion Predictions of Moving Objects in Dynamic Scenes

## Supplementary Material

### A. Experimental Setup

For the Bonn dataset, we use the raw depth sensors measurements. For TUM and OMD datasets, we use DepthAnythingV2 [75] to get smooth depth and recover the real scale with the original depth map because the raw depth sensor measurements come with large portion of invalid regions. In addition to the common metrics (PSNR, SSIM, LPIPS) used for mapping evaluation, we evaluate “DynaPSNR” as PSNR only for dynamic objects within 2D motion masks. We evaluate the Absolute Trajectory Error (ATE) of camera localization. The experiments are conducted on a desktop with a single NVIDIA 3090Ti (24GB). For low-level image processing, we stick to pre-trained models because 1. Jointly online per-frame optimizing the low-level 2D visual model is time and storage consuming, posing threats to real-time online purposes. 2. The disentanglement of the low-level 2D visual preprocessing model from 3D SLAM optimization enables convenient replacement of better 2D foundation models. Using pretrained semantic/optical/motion segmentation is a consensus as engineering-level image processing for SOTA dynamic SLAM even without GS rendering. SOTA and classical examples are shown as “**method {pretrained 2D optical/semantic/motion segmentation models they use}**”: DynaMoN [58] {DeepLabV3 [7] + DytanVO [60]}; DGSLAM [73] {OneFormer [19] + RAFT [71]}; DGSSLAM [73] {TrackAnything [8]}; Gassidy [72] {YOLO [56]}; PG-SLAM [32] {MaskRCNN [15]}; VDO-SLAM [79] {MaskRCNN [15] + PWC-Net [65]}; ClusterSLAM [18] {MaskRCNN}; DynaSLAM [3] {MaskRCNN}; DynaSLAM2 [4] {YOLACT [5]}; FlowFusion [80] {PWC-Net}. We use a combination of RAFT and a real-time online SAM2 [55] (Sec. B). Without our proposed novel online GS tracker and motion interpolation/extrapolation, simply segmenting motion masks cannot obtain high-quality dynamic GS rendering. Building a novel low-level 2D visual processing model is beyond the scope of this work.

### B. Dynamic GS Segmentation & Flow

While online 3D motion segmentation is challenging and slow today, we propose a novel Dynamic GS segmentation strategy (Fig. 3). Our GS is initialized from the point cloud of RGBD, so we estimate 2D pixel motion and align them to the GS motion. When our SLAM proceeds to current frame  $t$ , given two consecutive images  $C_{t-1}, C_t \in \mathbb{R}^{W \times H \times 3}$ , we first use a real-time optical flow model (RAFT [71]) to es-

	PSNR↑	SSIM↑	LPIPS↓	DynaPSNR↑
OMD (S4U)	30.6/31.0	95.1/95.7	15.1/15.1	34.6/31.1
TUM (xyz)	27.5/16.3	95.7/79.3	16.0/37.1	31.5/28.7
TUM (static)	26.9/12.8	96.1/77.7	14.0/37.8	30.3/27.0
TUM (rpy)	27.4/20.3	94.7/84.6	21.1/34.6	30.1/29.6
TUM (halfsphere)	27.2/19.4	94.5/83.1	20.0/35.4	30.7/31.6

Table 5. Ablation Study on the Impact of the Depth Quality. In each cell, the metric is “refined (DepthAnythingV2) depth/original sensor-depth”.

timate a flow image  $f_{t-1 \leftarrow t} \in \mathbb{R}^{W \times H \times 2}$ , where each pixel stores its own 2D velocity, and the velocities of moving pixels are distinct from the static pixels. We then, compute the gradient of  $f_{t-1 \leftarrow t}$ , which detect edges and close the shapes to get a coarse motion mask. By setting a click prompt at the object centroid, we use prompt-based model SAM2 [55] to segment 2D motion. Our strategy enables an automatic pipeline to segment moving pixels and filter out the static objects. Whereas the strategy handles well in general cases, it relies on robust optical flow.

Incorrect segmentation of static objects as dynamic does not deprecate GS quality since our dynamic GS management also handles static GS, it only introduces minor extra computation that could be ignored in practice. On the contrary, treating moving objects as static causes problems as classical static GS management cannot manage dynamic GS. However, our experiments show that our DynaGSLAM has tolerance to the low quality of 2D motion segmentations, which is further discussed in Sec 6 and Fig. 7.

### C. Trick of Dynamic Mapping Results

A popular practice when evaluating mapping metrics (PSNR, SSIM, LPIPS) in previous works (like [23]) is to set all invalid pixels to be 0, where the invalid mask is defined by the invalid regions of the original depth maps. However, this practice is unfair since setting estimation and ground truth pixel values both to 0 significantly benefits all metrics. For a fair evaluation, we disregard the benefits from this practice by evaluating without mask, which is why our implementation results of SplatTAM [23] in Table 1 is worse than the results proposed in Table 3 of [72]. However, even without the boost of this practice, our mapping accuracy is significantly better than SOTA baselines.

### D. Ablation Studies

**Impact of depth quality.** While our model is not very sensitive to noisy 2D motion priors, it relies on good depth. While the three datasets used in our work are all real

Dist Threshold $\lambda_d$	0	0.01	0.1	0.5
PSNR $\uparrow$	30.63	29.15	26.15	23.95
DynaPSNR $\uparrow$	34.58	28.20	20.21	17.04
SSIM $\uparrow$	95.13	93.88	90.89	87.49
LPIPS $\downarrow$	15.13	17.51	22.88	27.50
Reuse Rate (%)	0	22.75	56.36	83.53
Dyna GS Num $\downarrow$	48.0k	42.8k	38.3k	25.4k

Table 6. **Ablation Study** on the distance threshold of the nearest neighbor in Dynamic GS management. Reuse Rate is the ratio of dynamic GS that are used for at least two consecutive frames. With higher distance threshold, the GS reuse rate is higher, and less new GS are initialized, yielding lower mapping quality but higher efficiency with lower number of dynamic GS.

datasets with the depth from sensor, TUM’s depth is unreliable. We achieved outstanding results on Bonn and OMD with their original depth (Table 1 and 5). However, the depth maps from TUM include large invalid regions, resulting point cloud in poor quality. Nonetheless, we use the original sensor-depth from OMD (Table 5) and Bonn (Table 1) to get outstanding mapping quality, which still proves the robustness of our DynaGSLAM with practical depth sensor resource. Moreover, although the PSNR with noisy depth is not ideal, its counterpart “DynaPSNR” is still competitive. The good mapping quality of the dynamic region regardless of the static scene further validates our proposed novel dynamic GS management.

**Impact of the distance threshold.** The distance threshold  $\lambda_d$  is the most important hyper-parameter for our dynamic GS management algorithm (Fig. 3(b) & Section 5.2). Table 6 shows the effect of distance threshold on mapping quality and computational efficiency. In our experiments, we chose a low distance threshold of  $\lambda_d = 0.05$  with a limit of 50k dynamic GS for the best mapping quality. In other applications, it may be beneficial to trade off the mapping quality for computational efficiency.

## E. Additional Results

**Localization Results.** Figure 8 visualizes the camera trajectory and ATE on OMD dataset (S4U). We directly adopt the world-centric graph optimization strategy from [46], which considers the moving objects. In contrast, ICP and ORBSLAM2 used in RTGSLAM do not distinguish between static and dynamic objects. The result validates the importance of static/dynamic separation.

**Dynamic Mapping results.** We show additional qualitative comparisons of the GS mapping quality between our DynaGSLAM with SOTA baseline GS-SLAM works (RTGSLAM[54], SplaTAM[23], GSSLAM [44], and GSLAM [77]). Fig. 10 is an extension of Fig. 4 on the Bonn Dataset. Fig. 11 is an extension of Fig. 1 on the TUM Dataset. Our DynaGSLAM significantly outperforms these baselines, especially around the moving object such as the

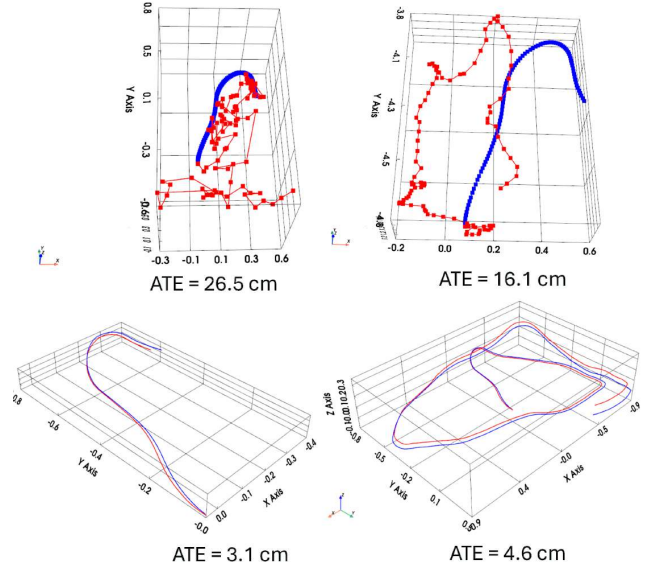


Figure 8. **Camera Tracking Results** on OMD dataset (S4U). Top Left: RTGSLAM [54]’s ICP, 90 frames. Top Right: RTGSLAM’s ICP refined ORBSLAM2 [47], 90 frames. Bottom Left: Ours, 90 frames. Bottom Right: Ours, 500 frames.



Figure 9. **Mapping result on OMD dataset.**

balloon and moving people. The failures of the baseline GS-SLAM works can be attributed to two aspects: 1. The past dynamic GS cannot be effectively deleted with static GS management, which become outlier GS in the background and contaminate static GS, such as the remnant red GS noises of RTGSLAM in Fig. 10, which belong to the red hoodie of the person in the past frames. 2. The new GS cannot be effectively added with static GS management, such as the missing left leg of GSLAM in 11 (row 1). Our novel proposed dynamic GS management algorithm overcomes all these limitations proves to be robust and accurate in the real dataset. We also include one more full mapping result as complementary to Fig. 2. We attach one more frame shot from the video sequence as shown in Fig. 9.

**Dynamic Motion Tracking & Prediction Results.** We show additional qualitative results of tracking & prediction in Fig. 12. In all of the three datasets, our DynaGSLAM shows the ability to synthesize unseen views by traversing the time dimension. As an extension of Fig. 5, we show the tracking (interpolation) and challenging prediction (extrapolation) over many missing frames. With the transparent white mask as the ground truth motion, we show that our



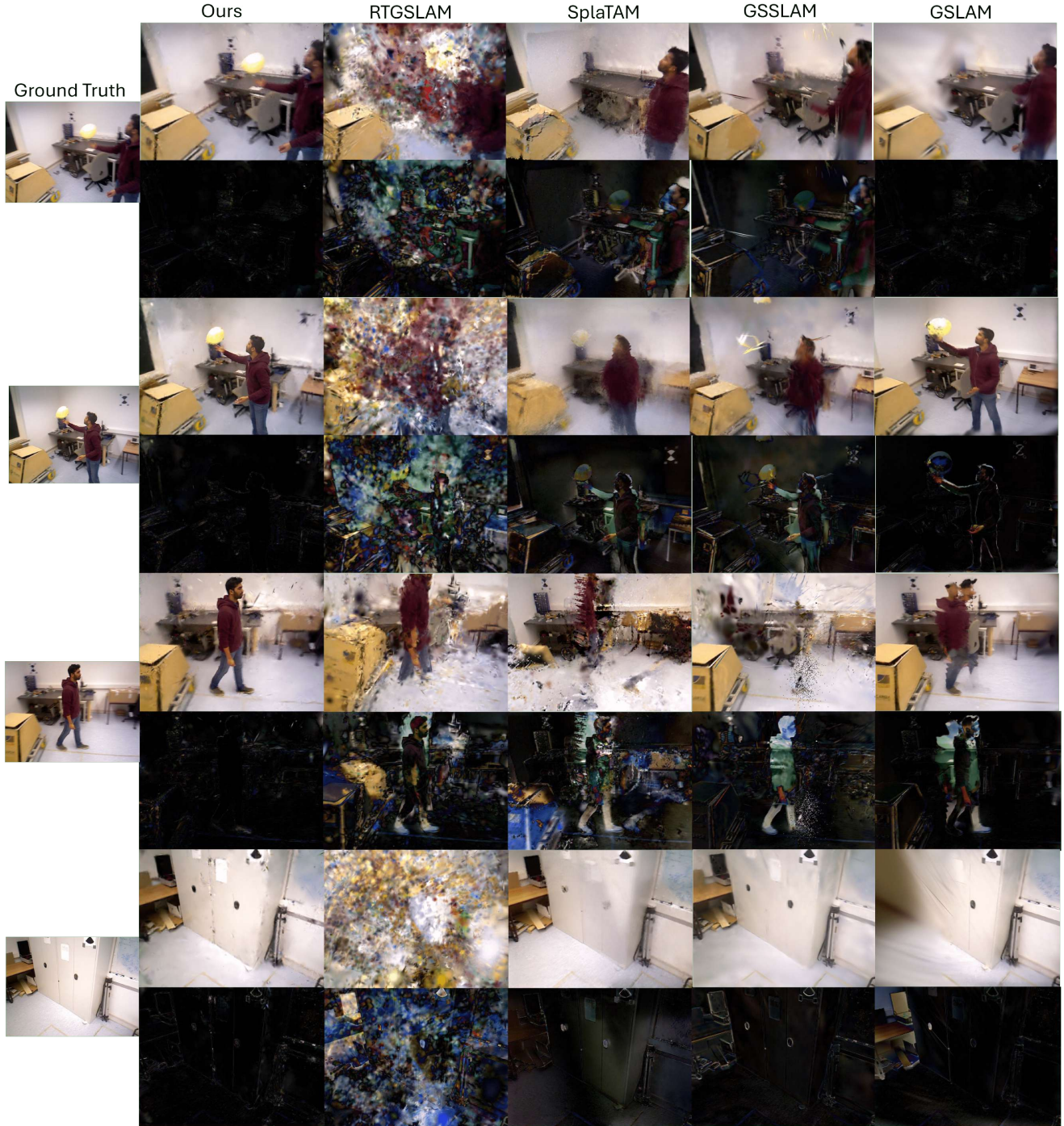


Figure 10. **GS Mapping Rendering Comparisons on the Bonn Dataset.** From top to bottom, the four scenes are: *balloon*, *balloon2*, *ps\_track*, *ps\_track2*. For each scene, the first row shows the RGB rendering results, the second row shows the absolute error between the rendered RGB to the ground truth. Our DynaGSLAM is obviously better than other SOTA GS-SLAM, especially at the moving entities such as the yellow balloon and the person.

motion model successfully brings GS to the desirable position, and the overlap of the dynamic entities (balloons and people) with the ground truth motion mask shows the qual-

ity of our proposed novel motion function. In contrast, the SOTA static GS-SLAM “RTGSLAM” [54] fails to correct the motion. Due to the lack of dynamic GS management,



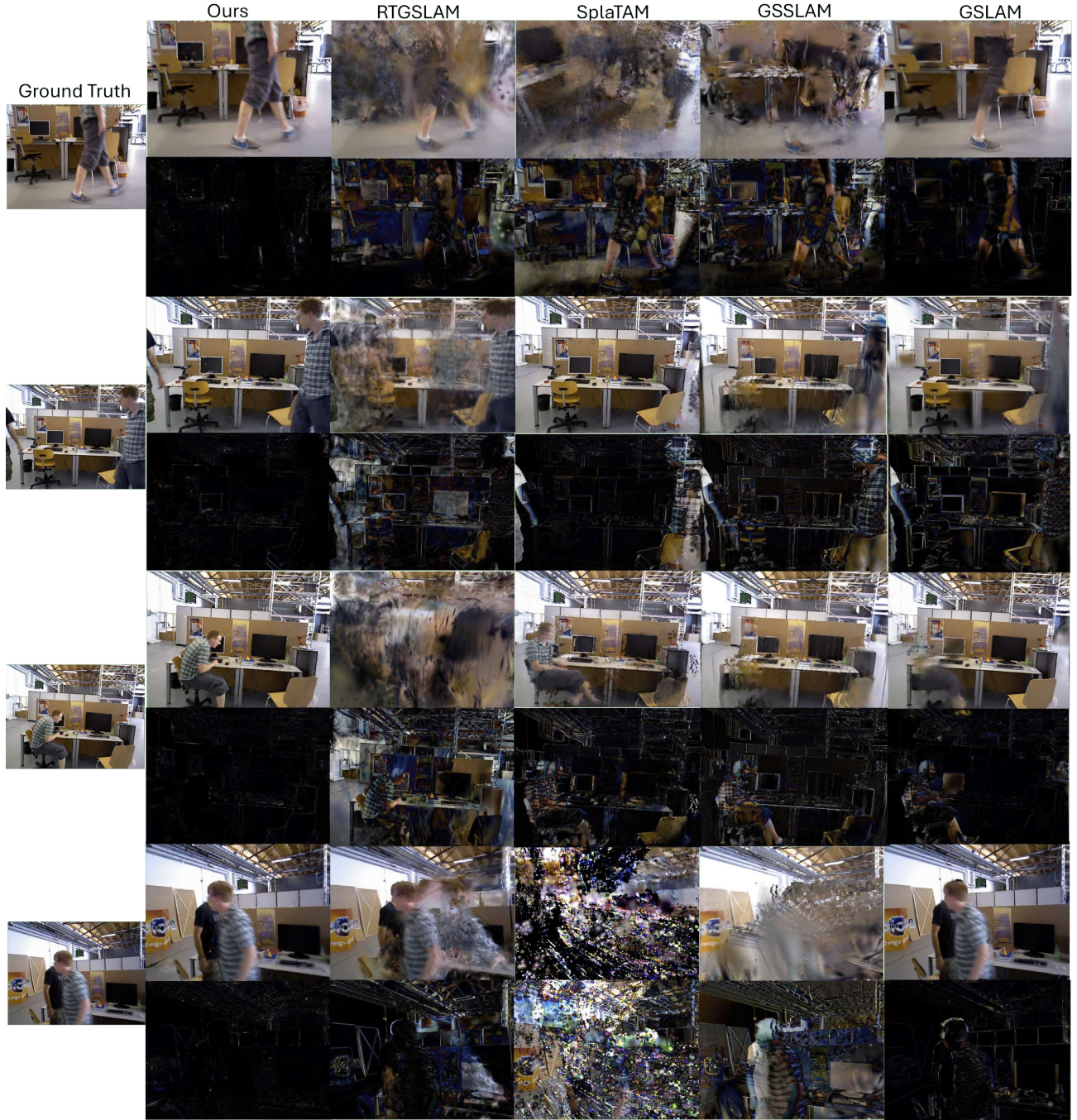


Figure 11. **GS Mapping Rendering Comparisons on the TUM Dataset.** From top to bottom, the four scenes are: *fr3\_walking\_xyz*, *fr3\_walking\_static*, *fr3\_walking\_static*, *fr3\_walking\_halfsphere*. For each scene, the first row shows the RGB rendering results, the second row shows the absolute error between the rendered RGB to the ground truth. Our DynaGSLAM is obviously better than other SOTA GS-SLAM, especially at the moving people.

their background static GS are also contaminated by moving GS. Our DynaGSLAM generates some minor artifacts under the extrapolation of long “Motion Horizon”, which is mainly because we use an extremely low number of GS

for real-time efficiency, so that individual GS can adjust the position and shape to cover more space whereas diminish their photometric textures, this issue can be moderated by trading-off the number and efficiency of GS.



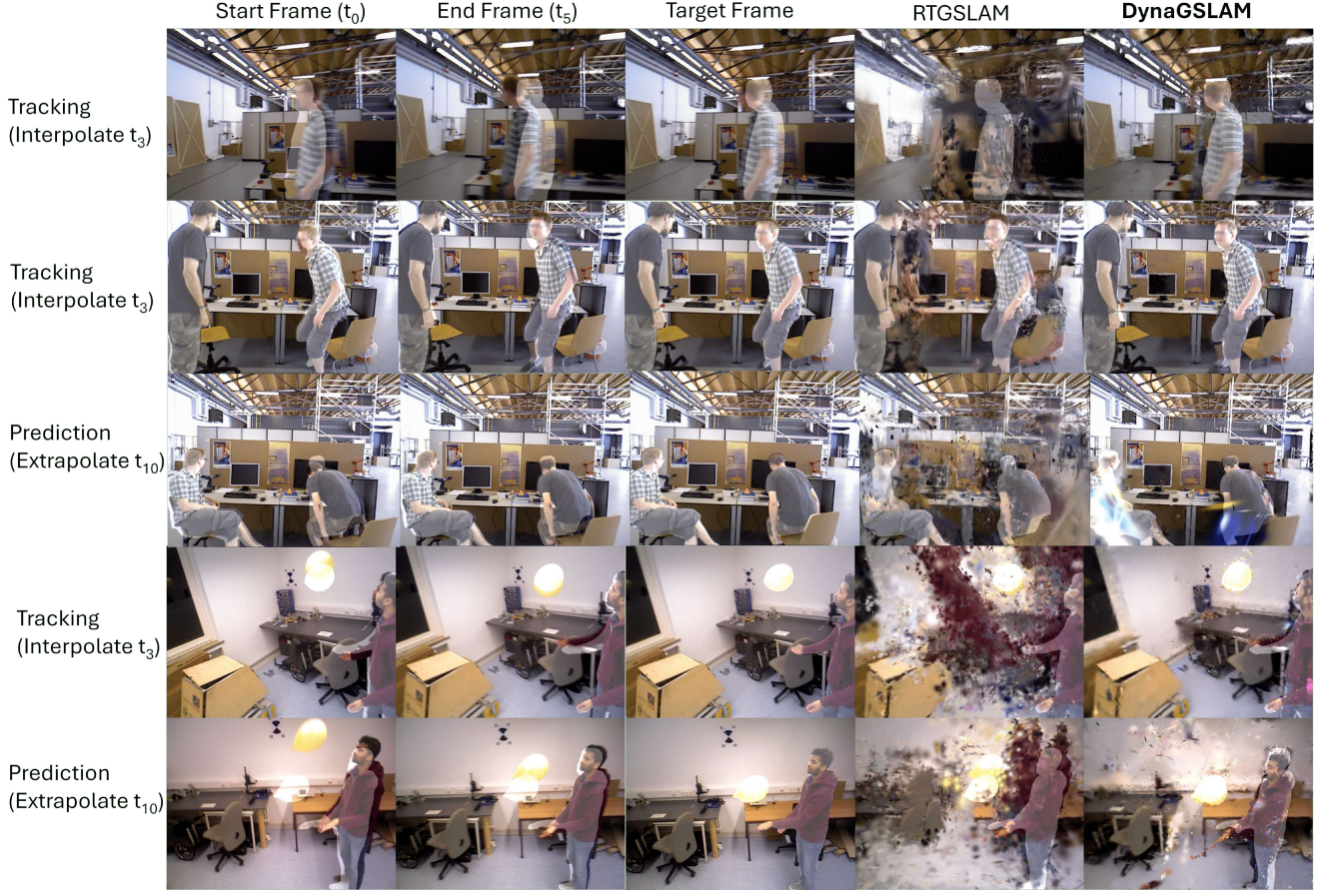


Figure 12. **Tracking & Prediction results on OMD, TUM and Bonn datasets.** This figure is an extension of Fig. 5 showing the tracking and prediction quality of our DynaGSLAM with our proposed novel GS motion function. Please check the annotation explanation in Fig. 5.

Methods	SplaTAM [23]	RTGSLAM [54]	Ours
Mapping (ms/frame)	1027	555	347
Localization (ms/frame)	5179	59	95
GS number	310K	520K	22K
Memory (GB)	0.82	1.7	2.6

Table 7. **Comparison of Inference Speed & Memory** on TUM fr3\_walking\_xyz with a single 3090Ti GPU.

**Online Speed & Memory.** Computational speed and memory usage are important for online SLAM. We compare online speed and memory usage on TUM “fr3\_wk\_xyz” sequence. The results are shown in Table 7. For SplaTAM [23], we follow their official configurations for TUM with 200 iters/frame for mapping optimization. For both RTGSLAM [54] and our DynaGSLAM, we follow RTGSLAM’s official configuration with 50 iters/frame for mapping optimization. SplaTAM and RTGSLAM only up-

date active GS to reduce the memory usage, but their static GS management fails on dynamic scenes yielding much larger number of GS than ours. Thanks to our dynamic GS management strategy, we achieve better mapping with much fewer GS. Our mapping runs at  $\sim 347$  ms/frame, including the online SAM2 [55] segmentation ( $\sim 36$ ms) and RAFT [71] optical flow ( $\sim 55$ ms). We chose SAM2 since its stability has been widely validated [68–70]. Our memory usage is higher than baselines mainly due to online SAM2 ( $\sim 1100$ mb) and RAFT ( $\sim 980$ mb), but it is acceptable given the benefits of dynamic rendering over baselines. We adapt DynoSAM for localization [46] (details in *Supp*) that costs  $\sim 450$  ms/frame. Overall, the fast computation guarantees real-time operation with efficient memory usage, while ensuring accurate mapping and localization.