Supplementary Material

In this supplementary material, we provide more details on our approach, experiment settings and further experimental results. We encourage readers to take a look at our open-sourced implementation upon publication for more detailed configuration. All experiments are performed on a desktop computer with an *AMD Ryzen Threadripper PRO 5955WX* CPU and *NVIDIA RTX4090* GPU.

1. Camera Projection

We use a pinhole camera model in all of our experiments due to the simplicity, but in theory we can also use more elaborate models e.g. with distortion.

$$\Pi\left(\mathbf{X}\right) = \left[f_x \frac{X}{Z} + c_x, \quad f_y \frac{Y}{Z} + c_y\right] \tag{8}$$

$$\Pi^{-1}\left(\mathbf{x},\ Z\right) = \left[Z\frac{u-c_x}{f_x}, \quad Z\frac{v-c_y}{f_y}, \quad Z, \quad 1\right]^T \tag{9}$$

2. Inference settings and hyperparameters

We run our system at resolution 320×432 on TUM-RGBD [44] and 360×640 on Replica [43].

2.1. Tracking

Tracking is configured by the *frontend* and *backend* parameters for graph building, optimization and our loop detector. Since the configuration system is quite complex, we will only brush across the critical parameter settings. We will release full configurations upon publication.

Frontend. We use a motion threshold of 3.0 for adding new keyframes. During scale optimization we use the objective

$$E = E + \alpha E_{reg} \tag{10}$$

with $\alpha = 0.001$. We found it important to keep keyframes longer in the frontend bundle adjustment window. This can be controlled with the *max age* variable in [47]. We increase this value from 25 to 30. We use a weight $\beta = 0.5$ on TUM-RGBD [44] and $\beta = 0.7$ on Replica [43] for measuring frame distance, see [47].

Backend. We run the backend every 8 frontend passes in our experiments. We build our global graph more conservatively by using a window of max. 150 frames, using up to 1500 edges on indoor scenes. A fully detailed configuration can be found on our open-sourced implementation for all experiments (and more).

Loop Detector. We compute visual features by using the EigenPlaces [1] ResNet50 network. We found qualitatively, that a feature threshold $\tau_f = 0.5$ works well in practice. We make the assumption that loop candidates are at least $\tau_t = 10$ frames apart. For our orientation threshold, we set $\tau_r = 15°$. This assumes that during a loop closure we have a very similar orientation, but due to drift a very distinct translation.

2.2. Rendering

We run our mapper every 20 frontend calls and optimize for 100 iterations at a small delay of 5 frames. We found that in practice, this can be arbitrarily tuned, i.e. we could also run more frequent with less training iterations. We anneal a 3D positional learning rate between $[1e-4, 1e-6]$ during our optimization, the other parameters are similar to [29]. During each iteration, we optimize newly added frames and add the 10 last frames and 20 random past frames similar to [29]. Since we run a test-time optimization, we are also prone to catastrophic forgetting. Using enough random frames ensures that this does not happen. Our system is not yet designed to handle large-scale scenes or unbounded scenes, where smarter strategies may be needed.

After filtering the tracking map with a covisibility check [47], we downsample the point cloud with factor 64 on Replica [43] and 16 on TUM-RGBD [44]. We use the same thresholds for densification across datasets. We made the experience that balancing these parameters can result in similar results as long as the total number of Gaussians is similar. We weight $L1$-error and $SSIM$ [51] in our appearance loss with $\lambda_2 = 0.2$. We balance depth and appearance supervision with $\lambda_1 = 0.9$ on TUM-RGBD [44] and 0.8 on Replica [43]. Balancing these two terms, can shift metrics slightly in favor of either appearance or geometry. Similar to [29] we encourage isotropic Gaussians with their scale regularization term. Our officially reported metrics are for dense depth supervision when a prior exists. For monocular video, we use the filtered tracking map as depth guidance.

Monte Carlo Markov Chain Gaussian Splatting. MCMC Gaussian Splatting [22] has additional hyperparameters for the noise level and the max. number of Gaussians in a scene. This poses an upper limit beyond which cannot be densified. We use $lr_{noise} = 1e4$ and use a slightly lower number of Gaussians from the runs with vanilla 3D Gaussian Splatting [21].

2D Gaussian Splatting 2D Gaussian Splatting has a slightly different objective function [15]. On top of the default rendering objective, we also have a normal consistency L_{normal} and depth distortion loss L_{dist}. We also found, that this representation has a different learning dynamic than 3D Gaussians. We therefore tuned the weighting to the best of our ability (without extensive parameter sweeps).

2.3. Feedback

In our feedback experiment, we perform backpropagation on the local pose graph of our rendering batch as is done in [29]. We used vanilla 3D Gaussian Splatting with the adaptive density control densification strategy [21]. As a sanity check, we only feedback the poses and/or disparity of rendered frames that have a decently similar disparity to the tracking map. The reason for this lies in the fact that our renderer has a small delay behind the leading tracker. If the rendering map is yet not covering enough pixels for some reason, we could potentially feedback a much sparser frame than we initially used during tracking. This could potentially disturb the update network [47]. We therefore check that the abs. rel. error between rendered disparity and tracking disparity is ≤ 0.2. If at least 50% of pixels satisfy this condition, then the frame is considered good.

3. Runtime and Memory Complexity

In this section, we provide more insights into the runtime and memory complexity of our system. Figure 4 already provides a detailed trade-off between rendering quality and inference speed on both Replica [43] and TUM-RGBD [44]. While these numbers will roughly transfer to other datasets (e.g. KITTI [12] or ScanNet [4]), they still depend on the input resolution, the amount of camera motion in the video and the bundle adjustment window size. Our monocular scale optimization has not yet been implemented as a compiled CUDA kernel, which usually subtracts roughly 2fps from the run-time compared to other modes.

The memory consumption is similar to [38]. We were able to reduce the memory footprint considerably compared to [47] by buffering images exclusively in int8 precision. Our framework can scale to long-term video and outdoor scenes like KITTI [12] with thousands of frames while still being in the limits of consumer GPU's. Memory complexity is a function that is hard to measure exactly by benchmarking specific scenes. We have the following memory complexities:

- Storing the update network [47] and loop detector [1] in float precision - $\mathcal{O}(1)$.
- Inference cost of running the networks at fixed precision and image resolution - $\mathcal{O}(HW)$
- A memory buffer growing with the number N of keyframes - $\mathcal{O}(N)$. The amount of motion in a scene will influence N compared to the length of the video stream.
- The Bundle Adjustment optimization scales quadratically w.r.t the number of camera poses in our optimization window - $\mathcal{O}(W^2)$. This can be restrictive when scenes have thousands of keyframes, i.e. we have a limit on how large our backend window can be and the number of factors we add to the graph. On indoor scenes, we usually perform a global optimization.
- Gaussian Splatting: Linear scaling w.r.t the number M of Gaussian primitives - $\mathcal{O}(M)$. We have to not only store the model parameters, but also gradients and optimizer states during optimization.

Components	Memory [MB]
Update network [47]	494
+ Video Buffer (empty)	1754
+ Frontend	8405
+ Backend	10239
+ Loop Detection [1]	10587
+ Mapping	13628

Table 8. **Memory Breakdown** on Replica [43] in *RGBD* mode. In this example, we use 135 000 Gaussian primitives, 360×640 input resolution and a buffer size of 128 keyframes. Note how storing the external sensor depths increases memory consumption compared to purely monocular.

Table 8 demonstrates a breakdown of the memory on *Replica/room0*.

More memory could be saved by low-precision inference and quantization of the neural networks and lowering the precision of the scene representation while sacrificing performance. However, we want to highlight that with advances in consumer graphics hardware (NVIDIA RTX 6000 Blackwell has 96GB RAM), the current generation allows us to render highly photorealistic large-scale scenes without introducing any new tricks. We encourage users to simply try out our framework.

4. Extended Evaluation

In this section, we want to provide more insights into how our system performs quantitatively and show more qualitative results. The reported rendering metrics for our comparison with related work are computed on the *keyframe* images based on the estimated poses, as is standard. However, this can give a warped view on the quality of a method. We want to highlight several key points:

- Every method has a *different keyframe management* or builds their graph based on different thresholds.
- *Not all metrics are reportedly available on all datasets.* We omitted an extensive evaluation of related work due to time constraints. Example: $L1$ metric is only readily available for Replica [43], however due to being a virtual dataset this metric is already quite saturated. The TUM-RGBD [44] benchmark is much more interesting.
- *Performance should be measured both on training and other frames!* Generalization of our test-time optimization is what normally counts, which is why we report results on non-training frames.
- The *difference between modes* only becomes apparent when considering both geometry and predicted images for both training and other frames.

Technique	#Gaussians	PSNR↑	LPIPS↓	L1↓	PSNR↑	LPIPS↓	L1↓
		KF			*Non-KF*		
TUM-RGBD							
Monocular	118 889	**26.84**	0.129	16.67	24.62	0.156	17.37
P-RGBD	119 100	26.53	0.131	8.50	24.81	0.155	8.38
RGBD	123 232	26.81	**0.110**	**4.26**	**24.89**	**0.144**	**4.63**
Replica							
Monocular	246 637	39.47	0.031	3.33	38.42	0.032	3.47
P-RGBD	248 175	38.33	0.032	3.72	38.34	0.033	3.83
RGBD	235 825	**39.66**	**0.028**	**0.55**	**38.87**	**0.029**	**0.61**

Table 9. **Full Rendering results**. We report our overall best results with MCMC densification [22] averaged over 5 runs on TUM-RGBD [44] and Replica [43] with refinement.

Supervision	#Gaussians	PSNR↑	LPIPS↓	L1↓	PSNR↑	LPIPS↓	L1↓
		KF			*Non-KF*		
TUM-RGBD							
dense	88 280	**25.98**	0.140	**8.2**	**24.48**	0.161	**8.2**
sparse	100 156	24.37	**0.129**	15.6	24.37	**0.155**	16.7
Replica							
dense	264 343	38.79	0.0361	3.55	**37.84**	0.0371	3.64
sparse	275 997	**38.95**	**0.0347**	**2.96**	37.82	**0.0361**	**3.11**

Table 10. **Sparse vs. dense supervision of vanilla 3D Gaussian Splatting [21] with monocular prior**. Geometry reconstruction depends heavily on the degree and quality of supervision. TUM-RGBD [44] does not have enough redundancy in frames for the filtered map to cover the scene. Replica [43] on the other hand will produce enough reliable covisible 3D points, such that the filtered tracking map provides strong supervision for each Gaussian. For this reason, we can achieve better results when using the sparser, filtered map on Replica. This closes the gap to related work [38]. We believe that with different priors and hyperparameters, we would achieve the same $L1$ error.

We show our full evaluation metrics of the overall best results in Table 9. We can only see a clean progression from monocular to *RGBD* inputs on the challenging TUM-RGBD [44] benchmark. We want to highlight, that strict monocular methods can overfit the appearance of training frames very well independent of tracking accuracy or geometric accuracy. However, we can generalize better and achieve much more accurate geometry when using additional depth priors. The benefit of a monocular prior [57] seems to be much smaller on Replica [43]. We found out in Table 10, that depending on the depth supervision signal this result changes. We also suspect [38] to supervise with a filtered depth map for this reason. Figure 5 and 6 show qualitative examples on top to get a feeling for how good methods work. We specifically chose non-training frames, which might put us at a disadvantage. We can observe clear improvements on fine-structured details, such as the lamp or background.

Due to our dense map both in tracking and rendering we can achieve better reconstructions than related work. For monocular reconstructions, we specifically show our results with a depth prior [57], which achieves much more accurate geometric reconstruction and better photo-realism on non-training frames than the monocular counter-part. This holds true even for slightly worse $L1$ metrics on Replica, as can be seen in the qualitative images. Results on Replica are already so accurate, that slight scale differences across time can create slightly non-flat walls.

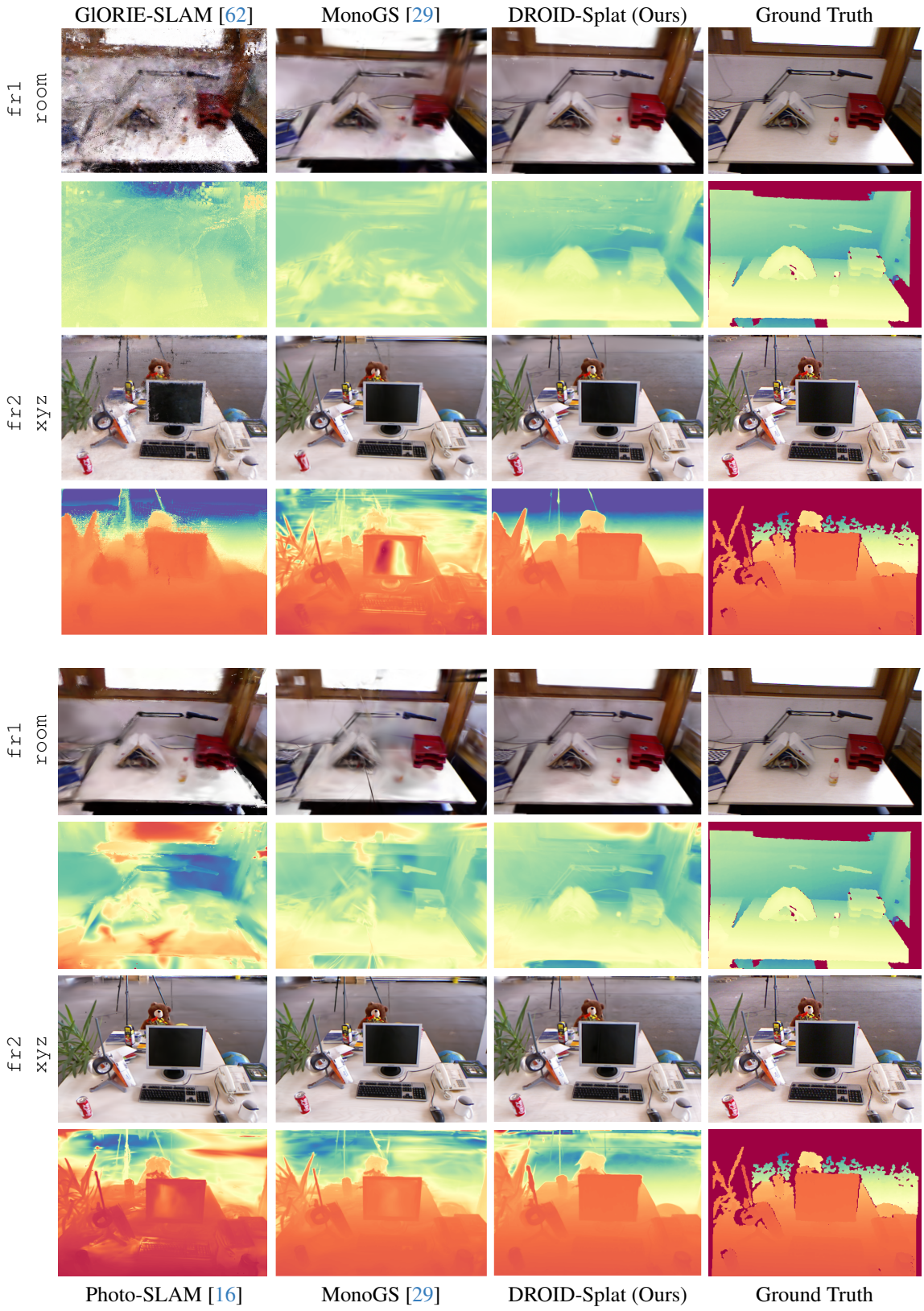

Figure 5. **More Rendering Results on TUM-RGBD [44]**. Top four rows are from *monocular* input, bottom from *RGBD*.

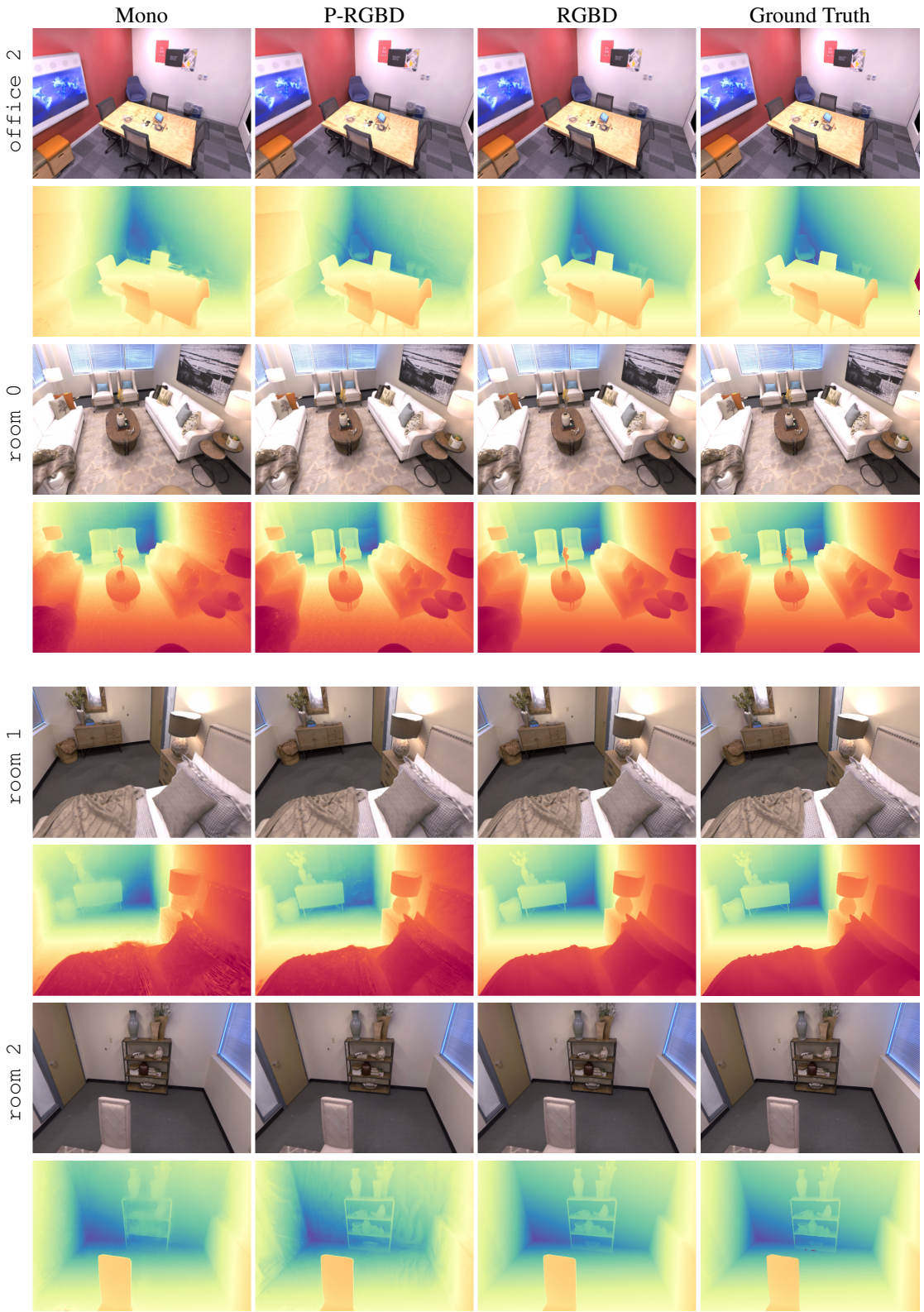

Figure 6. **Rendering Results on Replica [43].** We show non-training frames in multiple input modalities. Note how visually close the predictions are to the groundtruth.

Technique	# Gaussians	PSNR↑	LPIPS↓	L1↓	PSNR↑	LPIPS↓	L1↓
		KF			*Non-KF*		
no refinement							
2DGS [15]	173 309	20.71	0.31	10.2	19.84	0.33	10.3
3DGS [21]	111 878	23.26	0.23	9.1	22.46	0.25	9.2
+ MCMC [22]	113 060	**23.78**	**0.21**	**8.2**	**22.81**	**0.23**	**8.4**
with refinement							
2DGS [15]	131 576	22.87	0.21	8.8	21.73	0.23	8.7
3DGS [21]	88 280	25.98	0.14	**8.2**	24.47	0.16	**8.2**
+ MCMC [22]	119 100	**26.53**	**0.13**	8.5	**24.81**	**0.15**	8.4

Table 11. **Ablation Rendering Techniques**. We report results averaged over 5 runs on TUM-RGBD [44] in *P-RGBD* mode using [57] as a prior. We show a small progression with and without refinement. While 2D Gaussian Splatting [15] quickly produces smooth surfaces, this is not rewarded in the $L1$ error metric.

Table 11 shows a detailed ablation of Rendering techniques. We did not combine 2D Gaussian Splatting with the improved densification strategy [22], however we expect this to gain a similar improvement. We did not succeed in achieving better reconstructions for 2D Gaussian Splatting on TUM-RGBD [44]. However, we observe a clear benefit of this representation similar to the results in the respective paper, see examples in Figure 7. We can quickly converge to flat surfaces, which helps to avoid many floaters in outdoor-scenarios. On the used indoor datasets, vanilla 3D Gaussians perform better.

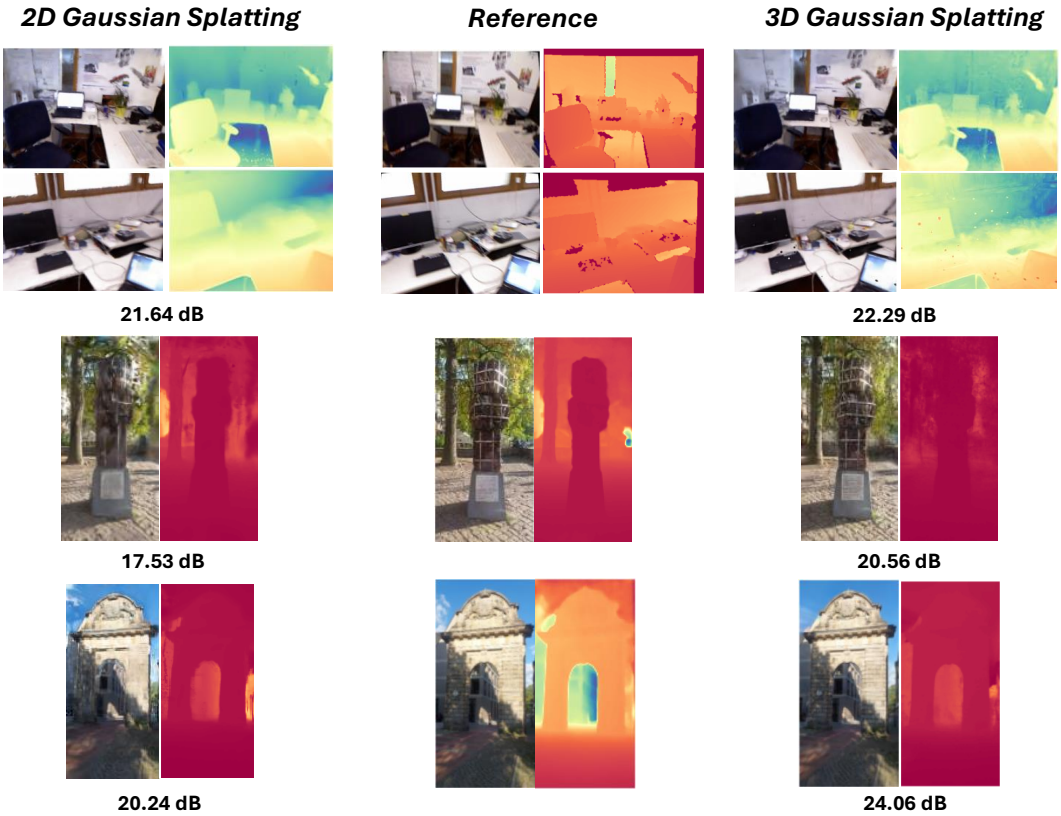

Figure 7. **Geometry vs. appearance**. We found, that 2D Gaussian Splatting [15] can quickly create smooth surfaces and does not accumulate many floaters in outdoor scenes. However, the rendering quality lacks behind 3D Gaussian Splatting [21] and as long as good supervision exists we can achieve better $L1$ metrics with 3D Gaussian Splatting.

5. Monocular Depth Prediction

Monocular depth prediction is a longstanding task with very impressive in-the-wild results of recent SotA models [2, 14, 55, 57]. We show some qualitative comparisons between selected models in Figure 8. Due to training on massive datasets, current single-image depth predictions can recover fine-structured details. Nonetheless, the accuracy of rel. depth on a single frame is not the only thing that matters for SLAM. We want to highlight:

- *The rel. depth error on a single image should be minimal.* This is obvious, however most recent models are only evaluated on specific benchmarks such as e.g. KITTI [12] or NYU [41]. Even though model predictions can look qualitatively very different, their abs. rel. error does not seem to be that different on untypical depth prediction benchmarks.
- *Temporal consistency matters a lot.* Even though we optimize scale $\mathbf{s}_i$ and shift $\mathbf{o}_i$ parameters to match our perceived optical flow, models result in differently consistent integrated maps. It is still very beneficial to have high temporal scale consistency in a depth model.

Recent diffusion models [14, 18] can leverage *billion-*

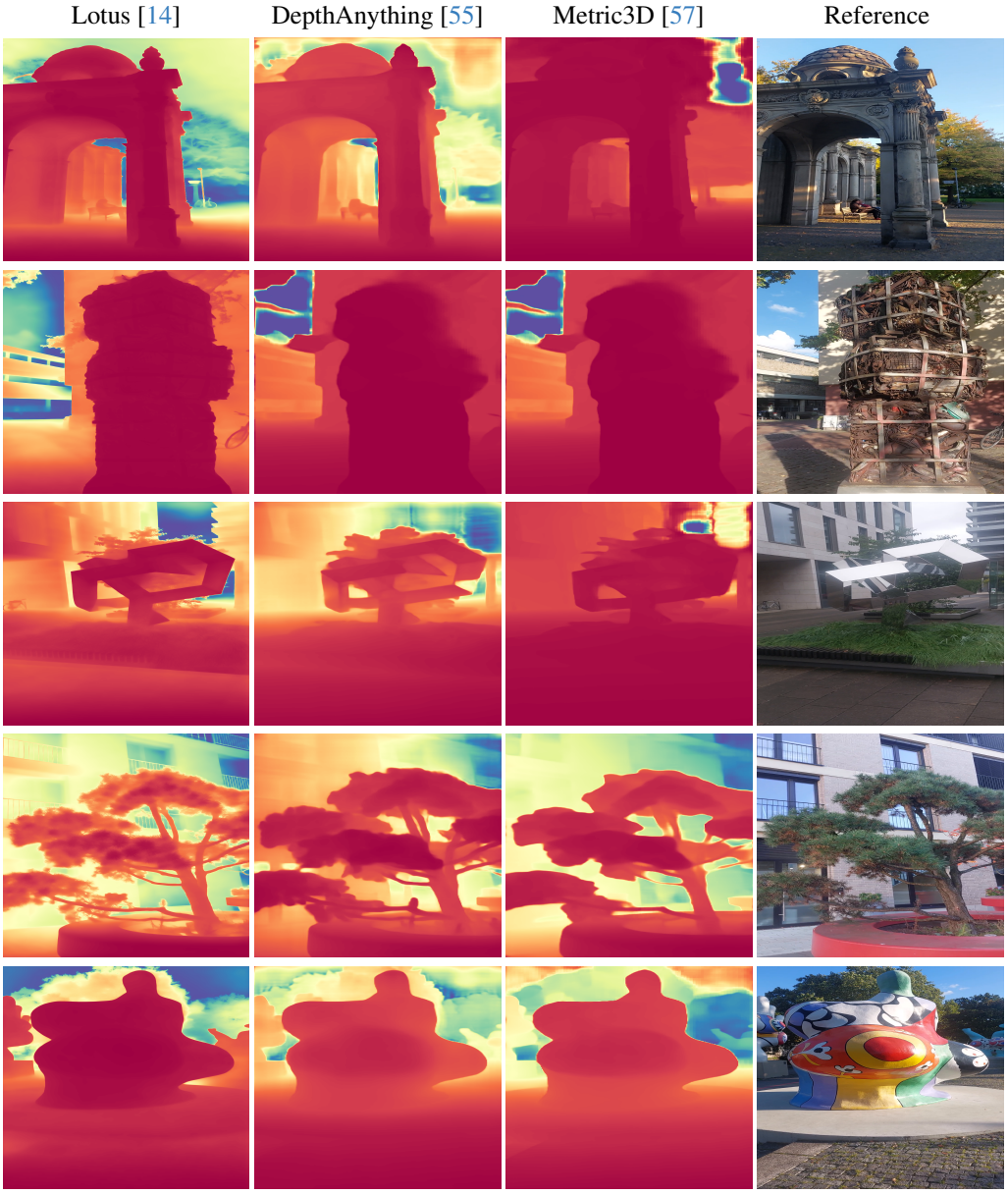

Figure 8. **Monocular depth prediction in-the-wild**. Models exhibit very clear differences w.r.t captured details and scale consistency on self-captured video. Problems can arise in particular for reflective surfaces or paintings.

Prior	ATE RMSE↓	PSNR↑	LPIPS↓	L1↓	ATE RMSE↓	PSNR↑	LPIPS↓	L1↓
	KF				*Non-KF*			
TUM-RGBD								
Metric3D [57]	1.93	**23.27**	**0.226**	**0.091**	1.83	**22.48**	**0.242**	**0.089**
ZoeDepth [2]	1.97	23.21	0.233	0.132	1.87	22.34	0.249	0.136
DepthAnything [55]	**1.91**	23.24	0.229	0.098	**1.79**	22.43	0.246	0.099
Lotus [14]	2.45	22.84	0.256	0.297	2.39	21.84	0.273	0.313
Replica								
Metric3D [57]	0.269	32.92	0.134	**0.037**	0.268	32.62	0.134	**0.038**
ZoeDepth [2]	**0.266**	**33.24**	0.123	0.088	**0.265**	**32.89**	0.123	0.091
DepthAnything [55]	0.268	33.06	0.131	0.063	0.268	32.73	0.131	0.066
Lotus [14]	0.275	32.23	**0.116**	0.295	0.278	31.72	**0.118**	0.318

Table 12. **Ablation Prior Depth on Replica [43] and TUM-RGBD [44]**. Recent SotA depth prediction networks [2, 14, 55, 57] have different qualities for SLAM. Good temporal consistency allows accurate geometry reconstruction. However, rendering quality and tracking does not necessarily correlate with it. Results are after online mapping without any refinement using vanilla 3D Gaussian Splatting [21] and averaged over 5 runs.

scale text-to-image pretraining to achieve strong depth prediction results with little finetuning. As can be seen in Figure 8, the qualitative difference and recovered fine-structured details compared to models trained only on *million-scale* depth prediction datasets seems obvious. However, diffusion models exhibit strong scale differences across a video. This seems to create a lot of floaters, in part enhanced due to the high-frequency details. We did not see an improvement for SLAM by integrating these models for this reason. Table 12 shows the performance of our system with vanilla 3D Gaussian Splatting [21]. We observe that *Metric3D* [57] consistently optimizes the best geometry. However, other metrics are not always consistent.

6. Loop Closure

Table 13 shows a comparison for long-term monocular tracking. Our SLAM system is competitive with the State-of-the-Art. We want to highlight, how a scale optimization of the monocular priors improves tracking significantly on indoor scenes, but can increase drift on long outdoor scenes. The trajectory from the DROID-SLAM network diverges on scene *01* like ORB-SLAM v3. We believe this issue could be fixed by using more autonomous driving data with mostly forward translation in the training.

	00	01	02	03	04	05	06	07	08	09	10	Avg
ORB-SLAM3 [3]	6.77	X	30.50	**1.04**	0.93	5.54	16.61	9.70	60.69	7.89	8.65	-
LDSO [11]	9.32	11.68	31.98	2.85	1.22	5.1	13.55	2.96	129.02	21.64	17.36	**22.42**
DROID-VO [47]	98.43	84.2	108.8	2.58	0.93	59.27	64.4	24.2	64.55	71.8	16.91	54.19
DPVO [48]	113.21	12.69	123.4	2.09	**0.68**	58.96	54.78	19.26	115.9	75.1	13.63	53.61
DROID-SLAM [47]	92.1	344.6	-	2.38	1.00	118.5	62.47	21.78	161.6	-	118.7	-
DPV-SLAM [25]	112.8	**11.50**	123.53	2.50	0.81	57.80	54.86	18.77	110.49	76.66	13.65	53.03
DPV-SLAM++ [25]	8.30	11.86	39.64	2.50	0.78	5.74	11.60	**1.52**	110.9	76.70	13.70	25.76
Ours mono	**6.44**	X (16.81)	35.06	5.45	0.73	26.89 (5.33)	15.21	3.01	64.72	9.31	17.21	
Ours PRGBD	10.52	X	114.04	6.10	5.19	17.55 (5.27)	**3.96**	7.73	44.14	29.06	10.75	-
Ours PRGBD*	8.09	X	**13.69**	2.11	3.23	**4.41** (3.76)	8.51	13.41	**13.37**	**4.68**	**3.15**	-

Table 13. **Ablation Long-term Tracking**. We compare monocular methods on KITTI [12], using *ATE*[m]. All our methods use the classic loop closure with PGO. We noticed, that DROID-SLAM [47] cannot use the sparse lidar in *RGBD*-mode without densification. Related work metrics are from [25]. In general, we notice that as long as the scene contains a loop closure, we can reduce drift reliably. Scenes *01*, *03* and *04* are mostly forward-translation trajectories. We believe, that the DROID-SLAM network cannot handle these sequences as well as [48]. We further noticed that these short sequences depend a lot on the internal pose interpolation (DPVO uses a damped linear interpolation, while the naive DROID implementation simply uses the previous pose). Scene *05* is special, since it involves a loop correction across multiple past segments. We can achieve much better results here without fixing the PGO for our prior optimization (results in brackets). On some scenes the scale optimization can enhance the drift in the system, in contrast a fixed prior can reduce it significantly as long as it is consistent enough over the video (PRGBD* uses a fixed prior).

7. How important is camera calibration really?

In this section we want to show some qualitative examples of in-the-wild footage with unknown intrinsics. As stated in the main paper, we perform a two-stage reconstruction:

1. Run the system without scale-optimization and optimize the camera intrinsics θ.
2. Use the now calibrated camera to run in *P-RGBD* mode and additionally optimize $\mathbf{s}_i$ and $\mathbf{o}_i$

Since we need an initial estimate of the intrinsics, we assume a heuristic where for a pinhole camera

$$\begin{aligned} fx = fy = (H + W)/2 \\ cx = W/2 \quad cy = H/2 \quad . \end{aligned} \tag{11}$$

The benefit of camera calibration was quantitatively shown in [13]. We report qualitative results on self-recorded scenes and show the robustness when initializing from a heuristic. It can be seen in Figure 9, that both intrinsics calibration and scale optimization are beneficial for in-the-wild reconstruction. With wrong intrinsics, we observe distorted odometry and structure. With scale optimization, we can generate globally consistent maps. All together forms a good basis for rendering.

8. Failure Cases

Due to the challenging unbounded outdoor setting on uncalibrated cameras, we quickly observed common limitations of our framework. We notice that even though monocular depth prediction networks allow highly detailed single-frame predictions, their usage on in-the-wild video is limited. Scale inconsistencies and inaccurate predictions make us accumulate floaters over time. We therefore have to use the following: We limit depth supervision to consistent 3D points using the covisibility check [47] and pixels with confidence $\sigma_i \geq 0.1$. This removes the sky and many floaters, but can also underconstraint the scene.

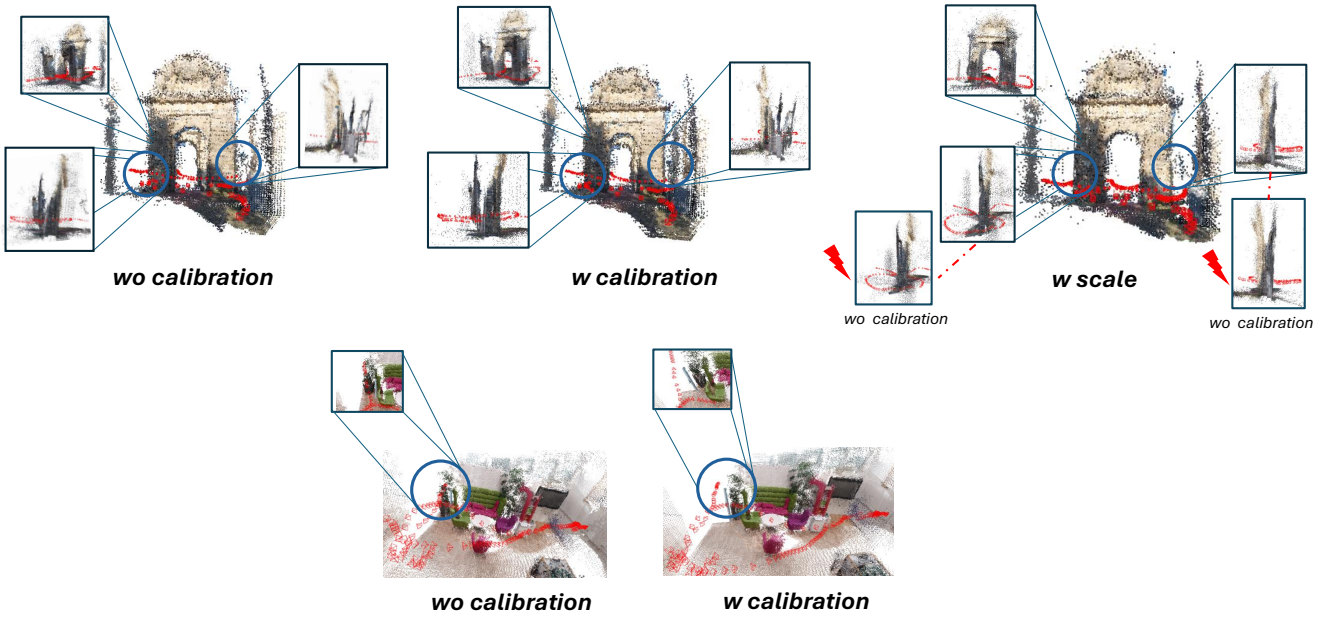

Figure 9. **Camera calibration and prior integration matter.** Distortion effects and artifacts both on the map and camera odometry can be observed without calibration. Using our strategy, we can get rid of distortions. The scale-optimized prior integration allows accurate structure reconstruction. Outdoor scenes require all together due to scale inconsistencies of common depth prediction models.

Figure 10. **Results on hand-captured cellphone videos.** In-the-wild outdoor scenes pose different challenges than benchmarks. *Left*: 3D Gaussian Splatting. *Right*: 2D Gaussian Splatting. While 2DGS is more resistant to floaters due to its surface optimization, it struggles with rendering quality. Both methods cannot deal well with strong lighting changes and reflections without extensions.

9. What did not work?

We tried the following things unsuccessfully:

- Multi-View Gaussian Splatting [7] backprojects crops of 2D appearance error into 3D by using the camera ray. We can then perform an intersection test to carve out a 3D volume across multiple views. This test identified new Gaussians, that cause a high 2D error, but were not identified in the original densification strategy [21]. However, we did not manage to improve densification this way within our framework.
- [45] uses a regularization term to battle catastrophic forgetting. We did not succeed on improving our metrics this way. We further tried to simply scale the gradients of optimized Gaussians by the number of times its frame has been already optimized by the renderer.
- Sparse GS [53] uses a *softmax* for rendering depths. We can identify floaters on outdoor scenes by analyzing the modality of the depth distribution. Since we compute an integrated absolute depth and supervise with priors, we were not able to converge quickly to the correct values due to the used *logarithm* function. Since Sparse GS was created with rel. depth supervision, we did not pursue this further.

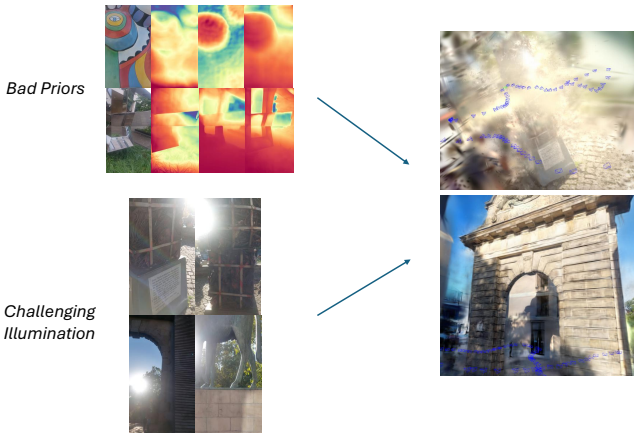

Figure 11. **Common failure cases**. Since we are heavily dependent on depth priors on in-the-wild video, our method can fail when priors get unreliable. Similarly, if the geometry supervision is not good enough, we accumulate floaters on outdoor scenes. Challenging lighting conditions can enhance this effect, since our model will overfit the scene and create additional Gaussians for modeling lighting effects (see Gaussians surrounding object).