PANOWALKER: ITERATIVE PERSISTENT PANORAMIC SCENE CONSTRUCTION

Anonymous authors
Paper under double-blind review

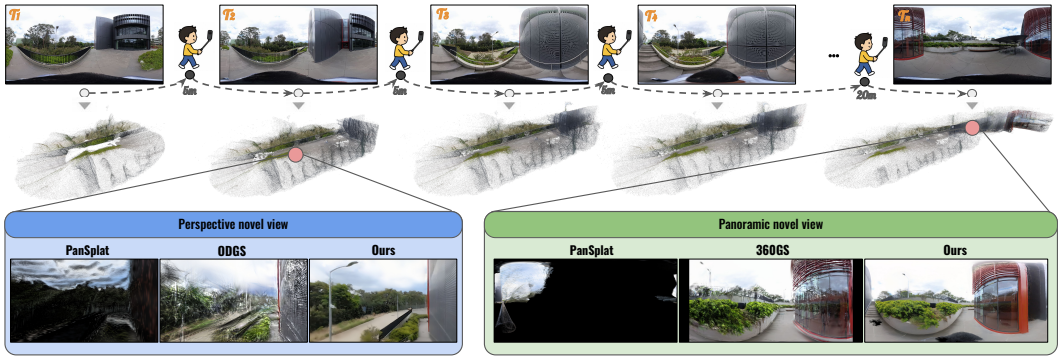

Figure 1: **PanoWalker progressively constructs a persistent panoramic scene.** Using supplied camera poses, we add pairwise Gaussian predictions to a common world frame as the walk progresses and render new views without per-scene optimization.

ABSTRACT

Real-world mobile capture is progressive: users explore an environment, acquire panoramas along a trajectory, and expect a reusable scene that grows with the captured history. Existing feed-forward methods reconstruct local inputs quickly, whereas per-scene optimization must repeatedly process the accumulated observations to provide successive scene updates. We present PanoWalker, a method for iterative panoramic 3D scene reconstruction without additional training or per-scene optimization. A frozen panoramic backbone predicts one local Gaussian set for each newly arrived pair. The supplied camera poses place every prediction in a shared world frame, where PanoWalker retains its source-pair identity and adds it to a persistent scene without re-predicting earlier pairs or globally optimizing the scene. At rendering time, each pair is rasterized independently; its camera distance and accumulated opacity determine its weight in the color blend, which is then composited using the source opacities. This resolves overlap at query time while leaving the stored scene unchanged. Among the compared methods, PanoWalker leads all reported metrics on HM3D, Replica, and 360Loc. It also achieves the lowest LPIPS on Insta360, where ODGS leads in WS-PSNR and SSIM. On two 360Loc walking segments, three updates from 10, 20, and 30 panoramas, each including its first preview, take 56.6 s in total on average; the final scenes reach 18.57 dB WS-PSNR. ODGS and 360GS refine their previous scenes under larger per-update time budgets, taking 2.12 and 2.15 minutes in total and reaching 14.42 and 13.67 dB, respectively. The code and model will be made publicly available.

1 INTRODUCTION

3D scene reconstruction (Kerbl et al., 2023; Lin et al., 2025b; Ye et al., 2024; Zhu et al., 2024; Wang et al., 2025) is a fundamental capability for embodied intelligence, robotics, digital twins, and immersive content creation. Recent advances in 3D Gaussian Splatting (3DGS) have made reconstruction fast and photorealistic, but most systems still treat a scene as a fixed batch: all

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

observations are supplied together, and one final representation is produced. Mobile capture is different. A user walks through an environment, discovers new regions over time, and may request a usable preview before capture is complete. The resulting representation should therefore grow without discarding the captured history or rebuilding the entire reconstruction.

Incremental registration and online Gaussian reconstruction have been explored (Cheng et al., 2025; Li et al., 2025). We study a complementary setting: extending one stored panoramic scene from predictions produced by a frozen pairwise backbone and placed in a common world frame using supplied camera poses. This setting exposes three coupled requirements. First, earlier predictions must remain available when new observations arrive. Second, independently predicted local surfaces may overlap or disagree even in a common world frame and therefore need a principled query-time composition rule. Third, evaluation must measure successive previews, rather than only the quality of a final scene reconstructed once.

Panoramic imagery is well suited to this sparse, casual capture regime. A perspective image observes only a limited field of view, so broad coverage requires dense viewpoints and strong overlap between neighboring images. In contrast, a panorama (Lin et al., 2025a; 2026) captures the full surrounding environment at one location and provides substantially richer spatial context per observation. Figure 1 illustrates the resulting walk-and-build setting: panoramas arrive along a trajectory while the stored scene expands into newly visited regions.

Existing panoramic reconstruction methods do not directly satisfy this setting. ODGS and 360GS (Lee et al., 2024; Bai et al., 2025) optimize one scene from a supplied set of panoramas; providing an updated scene after further capture requires additional optimization over the enlarged observation set. Feed-forward methods avoid per-scene optimization: PanSplat and PanoSplatt3R (Zhang et al., 2025; Ren et al., 2025) predict Gaussians from panoramic pairs, while PanoGRF (Chen et al., 2023) supports radiance-field rendering from two or more panoramas. Their predictions are fast, but turning a sequence of independent local predictions into a persistent, reusable scene requires both a representation that retains earlier predictions as new ones are added and a rendering rule that handles cross-pair overlap.

We therefore propose **PanoWalker**, a pipeline for iterative panoramic scene reconstruction without additional training or per-scene optimization. For every newly acquired panoramic pair, a frozen backbone predicts a local Gaussian set once. PanoWalker uses the supplied camera poses to place all predicted Gaussians in a shared world frame, stores them together as a Gaussian bank with their source-pair identities, and adds new predictions without revisiting earlier inputs or globally re-optimizing the scene. Accumulation retains every prediction. PanoWalker resolves overlap at rendering time. Jointly rasterizing all stored Gaussians would allow one pair's geometry to occlude another before their relative relevance to the query is considered. Instead, we preserve ordinary visibility within each source pair by rasterizing it independently. We then blend the resulting color layers using camera-distance weights multiplied by rendered coverage and normalized across sources. Coverage is the opacity accumulated from a pair's Gaussians at each pixel, and these coverages also determine the final composition of the blended colors. The stored bank remains unchanged, and the same representation can serve benchmark trajectory views and queries away from the capture path.

We evaluate PanoWalker through both conventional final-scene reconstruction and progressive walk-and-build updates. On the four-benchmark comparison in Table 1, PanoWalker leads all metrics on HM3D and Replica: it improves PSNR over the strongest baseline from 23.26 to 30.53 dB on HM3D and from 23.52 to 28.12 dB on Replica. PanoWalker also ranks first in all three 360Loc metrics and in Insta360 LPIPS, while ODGS retains the best Insta360 WS-PSNR and SSIM. Table 2 evaluates repeated scene updates after every five newly acquired pairs. PanoWalker delivers the 10-, 20-, and 30-panorama previews in 56.6 seconds in total and achieves 18.57 dB WS-PSNR, 0.663 SSIM, and 0.246 LPIPS after the final update. By comparison, ODGS/360GS runs with 30k iterations per update require 42.84–86.30 minutes, and the strongest reaches 16.42 dB WS-PSNR, 0.603 SSIM, and 0.347 LPIPS. When refining their previous scenes under larger per-update time budgets, ODGS and 360GS take 2.12–2.15 minutes in total and reach 14.42/13.67 dB WS-PSNR. Under these budgets, PanoWalker combines strong final quality with substantially faster updates.

In summary, our main contributions are as follows:

- **A walk-and-build formulation for panoramic reconstruction.** We formulate iterative panoramic reconstruction as a walk-and-build task in which registered observations arrive

along a trajectory and the system repeatedly delivers an updated scene before capture ends, reusing prior work as coverage grows.

- **A method for persistent scene construction and source-aware rendering without additional training.** We present PanoWalker, which combines two complementary designs. First, it retains frozen pairwise Gaussian predictions in the supplied world frame and adds new predictions with their pair labels without revisiting earlier inputs or globally optimizing the accumulated scene. Second, it preserves within-pair visibility by rasterizing each source pair independently and combines their colors using camera distance and rendered coverage before final compositing.

- **Final-scene and progressive evaluation.** We evaluate PanoWalker on four benchmarks in the conventional final-scene setting and with a progressive protocol measuring update latency, reconstruction quality, quality at fixed earlier views, and memory use. PanoWalker performs better than existing model on these benchmarks. In the progressive task, three updates and their first previews take 56.6 seconds in total, with 18.57 dB WS-PSNR after the final update; ODGS/360GS require 42.84–86.30 minutes with 30k iterations per update.

2 Related Work

Sparse Perspective Novel View Synthesis. Sparse-view novel view synthesis has evolved along two complementary directions. Scene-optimized methods fit neural radiance fields (Jain et al., 2021; Deng et al., 2022; Niemeyer et al., 2022; Roessle et al., 2022) or Gaussian representations (Park et al., 2025; Chen et al., 2025a; Bao et al., 2025; Song et al., 2026) to sparse posed images, using priors or regularization to constrain reconstruction. Feed-forward approaches amortize reconstruction across scenes, constructing radiance-field representations (Chen et al., 2021; 2025b; Liu et al., 2022; Wang et al., 2021; Xu et al., 2024) or predicting 3D Gaussians (Charatan et al., 2024; Chen et al., 2024a; Li et al., 2024; Ye et al., 2025) from sparse inputs. At the same time, the limited field of view of perspective images constrains the observed surrounding environment from each input, making broader scene coverage difficult to obtain from only a small number of views. This naturally motivates panoramic inputs, where each observation captures substantially richer spatial context and provides a stronger basis for scene-level reconstruction.

Sparse Panoramic Novel View Synthesis. Panoramic methods include per-scene Gaussian optimization (Bai et al., 2025; Lee et al., 2024), generalizable spherical radiance fields (Chen et al., 2023), and feed-forward Gaussian prediction (Zhang et al., 2025; Chen et al., 2025c; Ren et al., 2025). 360-GeoGS (Yao et al., 2026) further targets geometric consistency in feed-forward panoramic Gaussian reconstruction. These methods differ in their input support: PanoGRF supports multiple reference panoramas, and Splatter-360 can also use multiple context views. Pairwise inference is therefore a particular operating setting rather than a universal architectural restriction. Complementary to novel-view synthesis, PanoVGGT (Guo et al., 2026) predicts camera poses, depth maps, and point clouds from one or multiple panoramas. Our focus is accumulating frozen pairwise Gaussian predictions into a persistent scene while retaining their source identities for query-dependent rendering.

Local-to-Global Gaussian Fusion. Local-to-global Gaussian reconstruction has been studied through learned aggregation and explicit registration. FreeSplat (Wang et al., 2024) fuses cross-view Gaussian features with a learned pixel-wise triplet fusion module. RegGS (Cheng et al., 2025) registers predictions from a pretrained NoPoSplat backbone using explicit similarity-transform optimization, followed by global scene refinement. StreamGS (Li et al., 2025) incrementally reconstructs unposed image streams using cross-frame refinement and feature aggregation. **PanoWalker** assumes known panoramic poses and accumulates predictions from frozen pairwise backbones. It retains every prediction and the identity of the pair that produced it, then blends images rendered from each pair separately, without training a new fusion module or globally optimizing the accumulated scene.

For irregularly sampled inputs, LLFF (Mildenhall et al., 2019) independently renders local multiplane images and combines their colors using camera-distance weights multiplied by accumulated alpha.

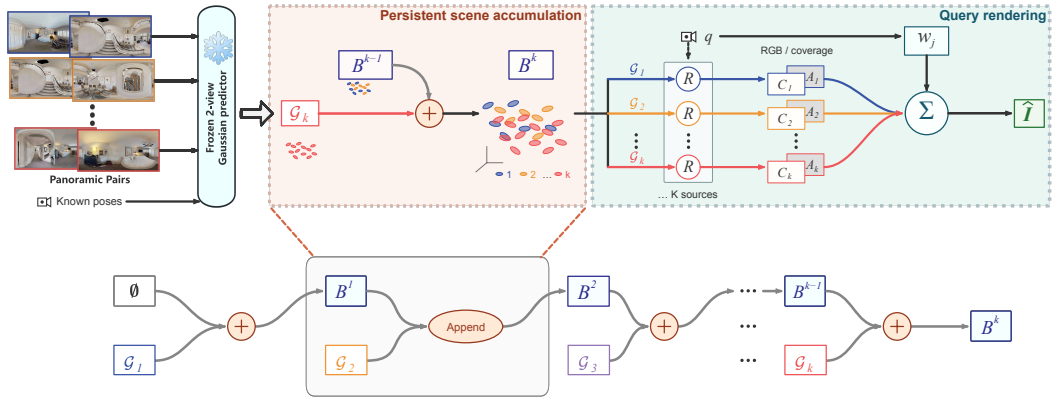

Figure 2: **PanoWalker overview.** Pairwise predictions extend a source-labelled Gaussian bank in a shared world frame. At each query, source layers are rendered independently and blended using camera distance and projected coverage.

3 METHOD

PanoWalker builds a persistent panoramic Gaussian scene from successively acquired image pairs with registered camera poses. A frozen backbone predicts each pair once; its Gaussians are added to a shared world representation together with their source identities. We call each pair's Gaussian prediction a source. For each requested camera view (query), the unified scene supplies rendered colors and coverage through its source components; coverage and camera-distance weights determine the source color weights used in the final rendering. This separates persistent scene construction from the resolution of overlapping predictions at rendering time. The method requires no per-scene optimization. The input camera poses are already expressed in a common world coordinate system.

3.1 PAIRWISE PANORAMIC GAUSSIAN PREDICTION

Let (I_k^a, I_k^b) be panoramic pair k, with camera-to-world poses (T_k^a, T_k^b). A frozen feed-forward backbone F_θ predicts a local Gaussian set in the common world coordinate system:

$$\mathcal{G}_k = F_\theta(I_k^a, I_k^b, T_k^a, T_k^b) = \{g_{ki}\}_{i=1}^{N_k}. \tag{1}$$

Each $g_{ki} = (\boldsymbol{\mu}_{ki}, \boldsymbol{\Sigma}_{ki}, \mathbf{c}_{ki}, \alpha_{ki})$ stores a center, covariance, color representation, and opacity. We also retain the two source-camera centers $\mathbf{a}_k^a$ and $\mathbf{a}_k^b$. Source identity k records which prediction produced a Gaussian; it is preserved throughout scene growth and rendering. The backbone parameters remain fixed, and the target image is not an input to prediction or rendering.

3.2 PERSISTENT SCENE ACCUMULATION

We process pairs in acquisition order and add every predicted Gaussian to the persistent bank, our stored collection of pair predictions. With $\oplus$ denoting concatenation of source-labelled Gaussian sets,

$$\mathcal{B}^{(0)} = \varnothing, \qquad \mathcal{B}^{(K)} = \bigoplus_{k=1}^{K} (\mathcal{G}_k, k). \tag{2}$$

The bank is a persistent world-coordinate scene whose source components remain available for subsequent queries as new observations arrive. All predicted Gaussian parameters and source identities are preserved: "unified" denotes a shared coordinate frame, not geometric deduplication. We write $\mathcal{B}_k$ for source k in the bank; it contains exactly the prediction $\mathcal{G}_k$.

When ΔK new pairs arrive, the update is

$$\mathcal{B}^{(K+\Delta K)} = \mathcal{B}^{(K)} \oplus \bigoplus_{k=K+1}^{K+\Delta K} (\mathcal{G}_k, k). \tag{3}$$

Only the new pairs pass through the backbone. Earlier predictions remain available for every subsequent query, including views far from the latest capture position. Thus each update extends one stored scene rather than replacing it with a small set of recently predicted views. We assume registered poses in a fixed coordinate frame; this formulation does not perform loop closure or revise earlier poses.

3.3 Query-Conditioned Composition of the Unified Gaussian Scene

At query time, we compose the source components of the persistent representation in Eq. 2 without modifying the stored Gaussians. Independently predicted surfaces can disagree in depth. Rasterizing their concatenation jointly lets one source's geometry occlude another source before their relative contributions are resolved. We instead retain ordinary Gaussian visibility within each pair, rasterize each pair independently, and combine its color and coverage with the other source layers. Each source layer consists of the color and coverage images rendered from one source. All stored source pairs are processed for each query.

Query-dependent source weights. For a query camera centered at $\mathbf{q}$, we measure distance to the closer input camera of each pair and use normalized inverse-distance weights:

$$\begin{aligned} d_k(\mathbf{q}) &= \min\big(\|\mathbf{q}-\mathbf{a}_k^a\|,\ \|\mathbf{q}-\mathbf{a}_k^b\|\big)\,, \\ w_k(\mathbf{q}) &= \frac{(d_k(\mathbf{q})+\epsilon_d)^{-4}}{\sum_{j=1}^{K}(d_j(\mathbf{q})+\epsilon_d)^{-4}}, \qquad \epsilon_d = 10^{-6}. \end{aligned} \tag{4}$$

The weight w_k favors pairs captured closer to the query camera. It acts on rendered layers; it does not change the stored Gaussian parameters or select a fixed number of source pairs.

Independent pair rasterization. For rendering only, we map each Gaussian opacity to optical depth and apply a fixed gain:

$$\begin{aligned} \bar{\alpha}_{ki} &= \mathrm{clip}(\alpha_{ki}, 10^{-6}, 1-10^{-6}), \\ \tau_{ki} &= -\log(1-\bar{\alpha}_{ki}), \\ \alpha_{ki}^{\mathrm{sat}} &= 1-\exp[-\min(40,\ s\tau_{ki})]\,, \qquad s = 7.5. \end{aligned} \tag{5}$$

The gain s is fixed as the scene grows, including scenes containing the first 5, 10, and 15 acquired pairs. Each pair is rasterized on a black background with these temporary opacities, giving a premultiplied color image $C_k(\mathbf{x})$ whose colors already include opacity weights. A second pass with a unit-color channel gives coverage $A_k(\mathbf{x}) \in [0,1]$, the accumulated opacity of that source at each pixel. Coverage depends on each source's geometry, projected Gaussian footprints, opacity, and within-source visibility. Neither the camera-distance weight w_k nor coverage A_k estimates geometric correctness. Pair geometry and stored opacities are unchanged.

Blending the source layers. At each pixel, let $p_k(\mathbf{x}) = \mathbf{1}[A_k(\mathbf{x}) > \eta]$, with $\eta = 10^{-3}$. We combine premultiplied colors and normalize by their weighted coverage:

$$b(\mathbf{x}) = \frac{\sum_{k=1}^{K} w_k p_k(\mathbf{x}) C_k(\mathbf{x})}{\max\Big(\sum_{k=1}^{K} w_k p_k(\mathbf{x}) A_k(\mathbf{x}), \epsilon_A\Big)}, \qquad \epsilon_A = 10^{-8}. \tag{6}$$

For the unpremultiplied source color $c_k(\mathbf{x}) = C_k(\mathbf{x})/\max(A_k(\mathbf{x}), \epsilon_A)$, Eq. 6 can be written as $b(\mathbf{x}) = \sum_k \beta_k(\mathbf{x}) c_k(\mathbf{x})$, where

$$\beta_k(\mathbf{x}) = \frac{w_k p_k(\mathbf{x}) A_k(\mathbf{x})}{\max\Big(\sum_{j=1}^{K} w_j p_j(\mathbf{x}) A_j(\mathbf{x}), \epsilon_A\Big)}. \tag{7}$$

For sources above the coverage threshold, the color weights β_k are proportional to the product of camera-distance weight w_k and coverage A_k.

The coverage threshold selects the sources that enter b; sources at or below it are still rendered and retain their own color in the final image. To specify the implemented handling of such layers, index

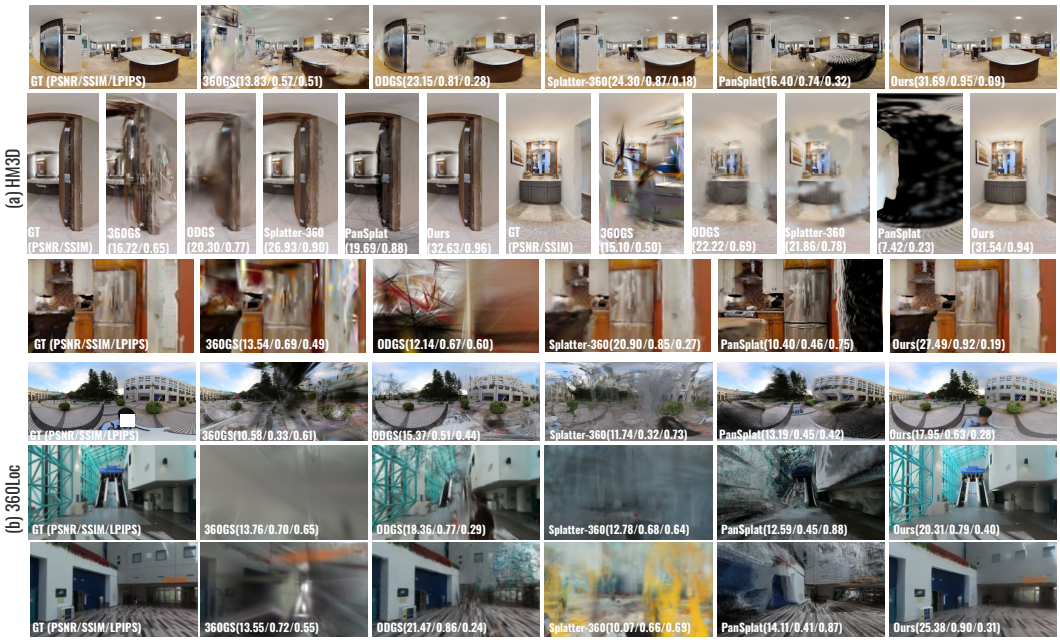

Figure 3: **Qualitative comparisons.** (a) HM3D; (b) three 360Loc views from Piatrium D0 and Concourse D0/D2. PanoWalker renders all 15 source pairs; scores are computed on the corresponding held-out views.

the pairs in increasing d_k order, preserving acquisition order for ties, and define

$$\kappa_k(\mathbf{x}) = A_k(\mathbf{x}) \prod_{j<k} \big(1 - A_j(\mathbf{x})\big),$$
$$\tilde{c}_k(\mathbf{x}) = \begin{cases} b(\mathbf{x}), & p_k(\mathbf{x}) = 1, \\ \dfrac{C_k(\mathbf{x})}{\max(A_k(\mathbf{x}), \epsilon_A)}, & p_k(\mathbf{x}) = 0. \end{cases} \tag{8}$$

The final image is

$$\hat{I}(\mathbf{x}) = \mathrm{clip}\left(\sum_{k=1}^{K} \kappa_k(\mathbf{x}) \tilde{c}_k(\mathbf{x}),\, 0,\, 1 \right). \tag{9}$$

When every nonzero-coverage layer passes the coverage threshold, the unclipped sum reduces to $b(\mathbf{x})[1 - \prod_k (1 - A_k(\mathbf{x}))]$: the blended color times union coverage. The low-coverage fallback in Eq. 8 retains order dependence in the general case. New pairs can change both blended color and coverage; preservation of earlier views is evaluated experimentally rather than imposed as an invariance guarantee.

Rendering cost as the scene grows. Our progressive implementation keeps the persistent bank on the CPU and uploads one pair's Gaussian attributes at a time. It retains the per-pair image layers needed for compositing, so their storage grows as $O(KHW)$ for an $H \times W$ image; CPU storage grows with the total Gaussian count. Each query uses two panoramic rendering passes per pair (color and coverage) and processes all K sources. The computational saving in scene updates comes from reusing previous pair predictions; uploading one pair at a time limits the GPU memory used for Gaussian attributes but does not imply constant total memory or constant rendering time. Section 4.3 measures the resulting end-to-end preview latency and memory.

4 Experiments

We evaluate PanoWalker along three axes: conventional final-scene quality, the latency and quality of successive previews during a walk, and the effect of the rendering rule used to combine independent pair predictions.

Table 1: **Final-scene reconstruction on four benchmarks.** PanoWalker leads all metrics on HM3D, Replica, and 360Loc. Row labels distinguish context views from source pairs; 15 PanoWalker pairs contain 29–30 distinct HM3D inputs and 23–30 Replica inputs. Best and second-best results are bold and underlined. WS denotes WS-PSNR (dB).

	HM3D				Replica			
Method	PSNR↑	WS↑	SSIM↑	LPIPS↓	PSNR↑	WS↑	SSIM↑	LPIPS↓
360GS	14.85	13.95	0.543	0.541	14.92	14.07	0.625	0.442
ODGS	17.90	16.68	0.661	0.500	18.97	17.76	0.755	0.420
PanSplat (2 views)	13.82	14.33	0.531	0.471	16.35	15.50	0.630	0.423
MVSplat360 (GS)	16.84	15.80	0.585	0.479	19.40	18.65	0.714	0.326
Splatter-360 (2 views)	17.76	17.11	0.575	0.413	20.92	19.96	0.769	0.241
Splatter-360 (15 views)	<u>23.26</u>	<u>22.34</u>	<u>0.807</u>	<u>0.246</u>	<u>23.52</u>	<u>22.55</u>	<u>0.841</u>	<u>0.203</u>
PanoWalker (15 pairs)	**30.53**	**29.57**	**0.938**	**0.095**	**28.12**	**27.28**	**0.883**	**0.163**

	Insta360			360Loc		
Method	WS-PSNR↑	SSIM↑	LPIPS↓	WS-PSNR↑	SSIM↑	LPIPS↓
360GS	14.47	0.436	0.483	12.23	0.326	0.626
ODGS	**21.45**	**0.710**	<u>0.280</u>	<u>16.86</u>	<u>0.586</u>	<u>0.376</u>
PanSplat (2 views)	11.33	0.336	0.625	11.88	0.329	0.625
MVSplat360 (GS)	16.08	0.501	0.534	13.55	0.340	0.648
Splatter-360 (2 views)	7.92	0.167	0.814	8.57	0.135	0.829
Splatter-360 (15 views)	11.20	0.351	0.710	12.25	0.335	0.694
PanoWalker (15 pairs)	<u>19.27</u>	<u>0.662</u>	**0.262**	**19.61**	**0.683**	**0.359**

Table 2: **Progressive reconstruction on 360Loc.** Results average two walks with five new panoramic pairs per update.

Method	Update latency (s)↓			Total	Final quality (30 inputs)		
	10 inputs	20 inputs	30 inputs	(min)↓	WS↑	SSIM↑	LPIPS↓
Predict new pairs (no scene optimization)							
Ours	18.2	17.9	20.5	0.94	18.57	0.663	0.246
Optimize from scratch (30k iterations per update)							
ODGS	1804.9	1556.5	1403.8	79.42	15.97	0.577	0.378
360GS	707.9	923.7	939.1	42.84	11.61	0.276	0.635
Refine previous scene (30k iterations per update)							
ODGS	1804.9	1714.5	1658.8	86.30	16.42	0.603	0.347
360GS	707.9	1154.5	1256.0	51.97	11.54	0.259	0.658
Refine previous scene (time budget per update)							
ODGS	32.7	40.9	53.5	2.12	14.42	0.459	0.589
360GS	34.9	40.7	53.3	2.15	13.67	0.424	0.794

4.1 Experimental Protocol

Datasets and inputs. We evaluate 10 indoor HM3D scenes (Ramakrishnan et al., 2021), 10 cross-domain Replica scenes (Straub et al., 2019), nine windows from one outdoor Insta360 recording (Zhang et al., 2025), and six mixed indoor–outdoor 360Loc trajectories (Huang et al., 2024). Each scene has eight test views. PanoWalker uses $K{=}15$ pairs per final scene, corresponding to 23–30 unique panoramas after deduplication. Appendix A.1 specifies the frame sets, pose units, pair baselines, target exceptions, and scoring grids.

ODGS (Lee et al., 2024) and 360GS (Bai et al., 2025) run for 30k iterations using 30 panoramas, except on Replica, where they use the locked 23–30-view manifests. PanSplat (Zhang et al., 2025) and Splatter-360 (Chen et al., 2025c) are evaluated with two inputs, and Splatter-360 additionally

Table 3: **Source-weight ablation.** Only distance and coverage coefficients in the shared color blend are changed; Gaussian banks, source renders, opacity, visibility, and outer compositing remain fixed.

	Color weights		Near-input		Off-path	
Variant	Distance	Coverage	WS↑	LPIPS↓	WS↑	LPIPS↓
w/o both	Off	Off	23.314	0.2600	17.171	0.4635
w/o distance	Off	On	24.122	0.2413	17.529	0.4533
w/o coverage	On	Off	29.560	0.0947	17.842	0.4047
Full	**On**	**On**	**29.567**	**0.0946**	**17.945**	**0.4037**



Figure 4: **Fixed-view scene growth on the Piatrium D0 segment of 360Loc.** The same query camera is rendered after 5, 10, and 15 pairs. This offline view lies beyond the region captured at the earlier updates. Labels show full-ERP WS-PSNR/SSIM/LPIPS and recorded cumulative update times. ODGS and 360GS refine their preceding time-budget results; their per-update budgets exceed PanoWalker's measured update times.

with 15 contexts. MVSplat360 (GS) (Chen et al., 2024b) denotes its released Gaussian stage without diffusion refinement.

Metrics and scoring. We report PSNR, solid-angle-weighted WS-PSNR (Sun et al., 2017), SSIM (Wang et al., 2004), and LPIPS (Zhang et al., 2018); tables abbreviate WS-PSNR as WS. HM3D, Replica, and the progressive experiment use a common 512×1024 scoring grid. The final-scene 360Loc metrics and Insta360 WS-PSNR/SSIM use each method's output resolution, while Insta360 LPIPS uses 512×1024. Appendix A.1 documents these protocol and input-budget differences.

Implementation. All runtimes use one NVIDIA L40. The frozen predictor is Splatter-360 at 512×1024 on HM3D/Replica and PanSplat at 1024×2048 on Insta360/360Loc, with batch size one and no test-time ensembling. Rendering follows Section 3.3. On HM3D, construction takes 152 s for PanoWalker, 680 s for 360GS, and 1,185 s for ODGS; Appendix B defines the unequal input counts and timing boundaries. At $K=15$, PanoWalker stores 15.73M Gaussians on HM3D.

4.2 Final-Scene Reconstruction

Table 1 reports final-scene quality. On HM3D, PanoWalker reaches 30.53 dB PSNR and 0.938 SSIM; its PSNR exceeds the strongest baseline, Splatter-360 with 15 contexts, by 7.27 dB. It also ranks first on all four Replica metrics, improving PSNR from 23.52 to 28.12 dB and WS-PSNR from 22.55 to 27.28 dB. These gains are obtained without per-scene optimization and persist on Replica despite its cross-domain appearance. On 360Loc, PanoWalker again ranks first in WS-PSNR, SSIM, and LPIPS, reaching 19.61 dB, 0.683, and 0.359. The result is less uniform on Insta360: ODGS leads in WS-PSNR and SSIM, whereas PanoWalker gives the best LPIPS (0.262). Figure 3 shows

that PanoWalker preserves sharper boundaries and more stable layouts, while the baselines exhibit blur, distortion, or incomplete geometry. Additional Insta360 examples appear in Figure 5. The real-capture outputs can still contain isolated spurious geometry and inconsistent surfaces; the current benchmarks therefore do not establish a universal indoor–outdoor advantage.

4.3 Progressive Walk-and-Build Evaluation

Separately from the final-scene benchmark, we request an update after every five new pairs on the Piatrium D0 and Concourse D2 walks from 360Loc. The three prefixes contain 10, 20, and 30 panoramas and span 27–29 m by the final request. Inputs and held-out queries are fixed in advance, poses are supplied, and each stage can use only the images acquired so far. PanoWalker predicts the five new pairs and keeps all earlier Gaussians unchanged. ODGS and 360GS are evaluated both by optimizing a new scene from all available inputs and by refining the previous scene. Refinement reuses its Gaussian and optimizer states, adds depth-based seeds from the new images, and optimizes both old and new content. We run both strategies for 30k iterations per update and separately test refinement under per-update time budgets; each budgeted stage inherits its own preceding budgeted result. PanoWalker produces the three requested previews in 18.2, 17.9, and 20.5 s, or 56.6 s in total. With 30k iterations per update, ODGS requires 79.42 min from scratch and 86.30 min with refinement; the corresponding 360GS totals are 42.84 and 51.97 min. These are fixed-iteration costs, not time-to-equal-quality speedups. Averaged over the eight final queries per walk, PanoWalker reaches 18.57 dB WS-PSNR, 0.663 SSIM, and 0.246 LPIPS, compared with 16.42 dB, 0.603, and 0.347 for the strongest 30k optimizer stream. Time-budgeted ODGS and 360GS still take 2.12 and 2.15 min and reach only 14.42 and 13.67 dB WS-PSNR.

On Concourse, PanoWalker reaches 18.38 dB, versus 17.80 dB for 30k ODGS refinement. Its mean WS-PSNR and SSIM changes at fixed early views round to $+0.000$, indicating that appending new predictions need not perturb already rendered content in this test. Each final bank contains 53.19M Gaussians stored on the CPU, while peak allocated GPU memory is 4.06 GB. Appendix D gives per-walk results, timing boundaries, query definitions, and memory accounting.

4.4 Component Ablation

Table 3 changes only the distance and coverage coefficients in the shared color blend, while holding the stored banks, source renders, opacity gain, visibility, and outer compositing rule fixed. At $K{=}15$, distance weighting improves WS-PSNR by 5.445 dB near the inputs and 0.415 dB off the capture path. The near-input queries lie only 0.020 m from an input camera at the median, so the larger gain should not be read as evidence of improved distant-view synthesis. Off-path, distance weighting improves 33 of 48 queries and every scene mean, but the 2–3 m distance bin averages -0.077 dB. Coverage weighting has a smaller effect: $+0.007$ dB near-input and $+0.103$ dB off-path at $K{=}15$, but -0.024 dB off-path at $K{=}5$. The two cues are therefore useful on average but not uniformly beneficial across queries or scene sizes.

Table A4 provides a broader rendering comparison. Joint rendering rasterizes all Gaussians together, whereas the layer control renders each input pair independently and overlays the resulting images in source-distance order. PanoWalker additionally replaces participating source colors with their distance- and coverage-weighted average before the same outer composite. Relative to joint rendering, it improves HM3D/Replica PSNR by 0.19/1.19 dB and Insta360/360Loc WS-PSNR by 0.15/0.61 dB, with a 1.91 dB gain on the off-path queries. Because joint rendering also changes opacity handling and cross-source visibility, this comparison is not a single-component ablation; Appendix E details the controlled variants and query populations. These results are diagnostics used during method development rather than independent validation.

5 Conclusion

We presented PanoWalker, a walk-and-build approach that appends frozen pairwise Gaussian predictions to a persistent, source-labelled scene without per-scene optimization. Independent source rendering and distance/coverage-aware blending resolve overlap at query time. PanoWalker leads all metrics on HM3D, Replica, and 360Loc, and produces three progressive previews in 56.6 s with 18.57 dB final WS-PSNR. These results demonstrate a favorable quality–waiting-time trade-off.

AI USE STATEMENT

AI tools partially assisted with text polishing and figure drawing. Research content including code, experiments, and results are authors' own.

REFERENCES

Jiayang Bai, Letian Huang, Jie Guo, Wen Gong, Yuanqi Li, and Yanwen Guo. 360-GS: Layout-guided panoramic Gaussian splatting for indoor roaming. In *International Conference on 3D Vision (3DV)*, pp. 1042–1053, 2025.

Zhenyu Bao, Guibiao Liao, Kaichen Zhou, Kanglin Liu, Qing Li, and Guoping Qiu. LoopSparseGS: Loop-based sparse-view friendly Gaussian splatting. *IEEE Transactions on Image Processing*, 34: 3889–3902, 2025. doi: 10.1109/tip.2025.3574929.

David Charatan, Sizhe Lester Li, Andrea Tagliasacchi, and Vincent Sitzmann. pixelSplat: 3D Gaussian splats from image pairs for scalable generalizable 3D reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19457–19467, 2024.

Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. MVSNeRF: Fast generalizable radiance field reconstruction from multi-view stereo. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 14124–14133, 2021.

Kangjie Chen, Yingji Zhong, Zhihao Li, Jiaqi Lin, Youyu Chen, Minghan Qin, and Haoqian Wang. Quantifying and alleviating co-adaptation in sparse-view 3D Gaussian splatting. In *Advances in Neural Information Processing Systems*, volume 38, 2025a. doi: 10.52202/085713-3866.

Yuedong Chen, Haofei Xu, Chuanxia Zheng, Bohan Zhuang, Marc Pollefeys, Andreas Geiger, Tat-Jen Cham, and Jianfei Cai. MVSplat: Efficient 3D Gaussian splatting from sparse multi-view images. In *European Conference on Computer Vision (ECCV)*, pp. 370–386, 2024a.

Yuedong Chen, Chuanxia Zheng, Haofei Xu, Bohan Zhuang, Andrea Vedaldi, Tat-Jen Cham, and Jianfei Cai. MVSplat360: Feed-forward 360 scene synthesis from sparse views. In *Advances in Neural Information Processing Systems*, volume 37, pp. 107064–107086, 2024b.

Yuedong Chen, Haofei Xu, Qianyi Wu, Chuanxia Zheng, Tat-Jen Cham, and Jianfei Cai. Explicit correspondence matching for generalizable neural radiance fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(12):11708–11719, 2025b. doi: 10.1109/tpami.2025.3598711.

Zheng Chen, Yan-Pei Cao, Yuan-Chen Guo, Chen Wang, Ying Shan, and Song-Hai Zhang. PanoGRF: Generalizable spherical radiance fields for wide-baseline panoramas. In *Advances in Neural Information Processing Systems*, volume 36, pp. 6961–6985, 2023.

Zheng Chen, Chenming Wu, Zhelun Shen, Chen Zhao, Weicai Ye, Haocheng Feng, Errui Ding, and Song-Hai Zhang. Splatter-360: Generalizable 360 Gaussian splatting for wide-baseline panoramic images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21590–21599, 2025c.

Chong Cheng, Yu Hu, Sicheng Yu, Beizhen Zhao, Zijian Wang, and Hao Wang. RegGS: Unposed sparse views Gaussian splatting with 3DGS registration. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 8100–8109, 2025.

Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised NeRF: Fewer views and faster training for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12882–12891, 2022.

Yijing Guo, Mengjun Chao, Luo Wang, Tianyang Zhao, Haizhao Dai, Yingliang Zhang, Jingyi Yu, and Yujiao Shi. PanoVGGT: Feed-forward 3D reconstruction from panoramic imagery. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 36444–36453, 2026.

Huajian Huang, Changkun Liu, Yipeng Zhu, Hui Cheng, Tristan Braud, and Sai-Kit Yeung. 360Loc: A dataset and benchmark for omnidirectional visual localization with cross-device queries. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22314–22324, 2024.

Ajay Jain, Matthew Tancik, and Pieter Abbeel. Putting NeRF on a diet: Semantically consistent few-shot view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 5885–5894, 2021.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):1–14, 2023. doi: 10.1145/3592433.

Suyoung Lee, Jaeyoung Chung, Jaeyoo Huh, and Kyoung Mu Lee. ODGS: 3D scene reconstruction from omnidirectional images with 3D Gaussian splattings. In *Advances in Neural Information Processing Systems*, volume 37, pp. 57050–57075, 2024.

Hao Li, Yuanyuan Gao, Chenming Wu, Dingwen Zhang, Yalun Dai, Chen Zhao, Haocheng Feng, Errui Ding, Jingdong Wang, and Junwei Han. GGRt: Towards pose-free generalizable 3D Gaussian splatting in real-time. In *European Conference on Computer Vision (ECCV)*, pp. 325–341, 2024.

Yang Li, Jinglu Wang, Lei Chu, Xiao Li, Shiu-Hong Kao, Ying-Cong Chen, and Yan Lu. StreamGS: Online generalizable Gaussian splatting reconstruction for unposed image streams. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 25841–25850, 2025.

Xin Lin, Xian Ge, Dizhe Zhang, Zhaoliang Wan, Xianshun Wang, Xiangtai Li, Wenjie Jiang, Bo Du, Dacheng Tao, Ming-Hsuan Yang, and Lu Qi. One flight over the gap: A survey from perspective to panoramic vision. *arXiv preprint arXiv:2509.04444*, 2025a.

Xin Lin, Shi Luo, Xiaojun Shan, Xiaoyu Zhou, Chao Ren, Lu Qi, Ming-Hsuan Yang, and Nuno Vasconcelos. HQGS: High-quality novel view synthesis with Gaussian splatting in degraded scenes. In *International Conference on Learning Representations*, 2025b.

Xin Lin, Meixi Song, Dizhe Zhang, Wenxuan Lu, Haodong Li, Bo Du, Ming-Hsuan Yang, Truong Nguyen, and Lu Qi. Depth any panoramas: A foundation model for panoramic depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 26835–26844, 2026.

Yuan Liu, Sida Peng, Lingjie Liu, Qianqian Wang, Peng Wang, Christian Theobalt, Xiaowei Zhou, and Wenping Wang. Neural rays for occlusion-aware image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7824–7833, 2022.

Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics*, 38(4):1–14, 2019. doi: 10.1145/3306346.3322980.

Michael Niemeyer, Jonathan T. Barron, Ben Mildenhall, Mehdi S. M. Sajjadi, Andreas Geiger, and Noha Radwan. RegNeRF: Regularizing neural radiance fields for view synthesis from sparse inputs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5480–5490, 2022.

Hyunwoo Park, Gun Ryu, and Wonjun Kim. DropGaussian: Structural regularization for sparse-view Gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21600–21609, 2025.

Santhosh K. Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alex Clegg, John Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X. Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-Matterport 3D dataset (HM3D): 1000 large-scale 3D environments for embodied AI. *arXiv preprint arXiv:2109.08238*, 2021.

Jiahui Ren, Mochu Xiang, Jiajun Zhu, and Yuchao Dai. PanoSplatt3R: Leveraging perspective pretraining for generalized unposed wide-baseline panorama reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 28959–28969, 2025.

Barbara Roessle, Jonathan T. Barron, Ben Mildenhall, Pratul P. Srinivasan, and Matthias Nießner. Dense depth priors for neural radiance fields from sparse input views. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12892–12901, 2022.

Meixi Song, Xin Lin, Dizhe Zhang, Haodong Li, Xiangtai Li, Bo Du, and Lu Qi. D^2GS: Depth-and-density guided Gaussian splatting for stable and accurate sparse-view reconstruction. In *International Conference on Learning Representations*, 2026.

Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, Anton Clarkson, Mingfei Yan, Brian Budge, Yajie Yan, Xiaqing Pan, June Yon, Yuyang Zou, Kimberly Leon, Nigel Carter, Jesus Briales, Tyler Gillingham, Elias Mueggler, Luis Pesqueira, Manolis Savva, Dhruv Batra, Hauke M. Strasdat, Renzo De Nardi, Michael Goesele, Steven Lovegrove, and Richard Newcombe. The Replica dataset: A digital Replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

Yule Sun, Ang Lu, and Lu Yu. Weighted-to-spherically-uniform quality evaluation for omnidirectional video. *IEEE Signal Processing Letters*, 24(9):1408–1412, 2017. doi: 10.1109/LSP.2017.2720693.

Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P. Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. IBRNet: Learning multi-view image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4690–4699, 2021.

Xiaoyuan Wang, Yizhou Zhao, Botao Ye, Xiaojun Shan, Weijie Lyu, Lu Qi, Kelvin C. K. Chan, Yinxiao Li, and Ming-Hsuan Yang. HoliGS: Holistic Gaussian splatting for embodied view synthesis. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Yunsong Wang, Tianxin Huang, Hanlin Chen, and Gim Hee Lee. FreeSplat: Generalizable 3D Gaussian splatting towards free-view synthesis of indoor scenes. In *Advances in Neural Information Processing Systems*, volume 37, pp. 107326–107349, 2024.

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004. doi: 10.1109/TIP.2003.819861.

Haofei Xu, Anpei Chen, Yuedong Chen, Christos Sakaridis, Yulun Zhang, Marc Pollefeys, Andreas Geiger, and Fisher Yu. MuRF: Multi-baseline radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20041–20050, 2024.

Jiaqi Yao, Zhongmiao Yan, Jingyi Xu, Songpengcheng Xia, Yan Xiang, and Ling Pei. 360-GeoGS: Geometrically consistent feed-forward 3D Gaussian splatting reconstruction for 360 images. *arXiv preprint arXiv:2601.02102*, 2026.

Botao Ye, Sifei Liu, Haofei Xu, Xueting Li, Marc Pollefeys, Ming-Hsuan Yang, and Songyou Peng. No pose, no problem: Surprisingly simple 3D Gaussian splats from sparse unposed images. In *International Conference on Learning Representations*, 2025.

Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou. AbsGS: Recovering fine details in 3D Gaussian splatting. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 1053–1061, 2024.

Cheng Zhang, Haofei Xu, Qianyi Wu, Camilo Cruz Gambardella, Dinh Phung, and Jianfei Cai. PanSplat: 4K panorama synthesis with feed-forward Gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11437–11447, 2025.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 586–595, 2018.

Zehao Zhu, Zhiwen Fan, Yifan Jiang, and Zhangyang Wang. FSGS: Real-time few-shot view synthesis using Gaussian splatting. In *European Conference on Computer Vision (ECCV)*, 2024.

Table A1: **Evaluation sets and PanoWalker inputs.** Each scene, recording window, or trajectory contributes eight test panoramas; input counts denote unique panoramas in 15 pairs.

Benchmark	Evaluation set	Frozen backbone	Inputs
HM3D	10 scenes	Splatter-360	29–30
Replica	10 regenerated scenes	Splatter-360	23–30
Insta360	9 windows, one recording	PanSplat	29–30
360Loc	6 trajectories	PanSplat	29–30

A Detailed Experimental Protocols

A.1 Inputs and evaluation settings

For PanoWalker, K is the number of accumulated panoramic pairs (each pair feeds two views to the frozen backbone, so K=15 covers 29–30 unique input panoramas on HM3D and 23–30 on Replica after deduplication). Multi-view Splatter-360 uses 15 context panoramas, sharing PanoWalker's backbone and reference panoramas on HM3D; PanoWalker also processes each reference panorama's paired view. Processing all pair frames jointly with Splatter-360 exceeds the available GPU memory at the evaluated resolution.

Pose and sampling conventions. The estimated mean distance between the first panoramas of consecutive source pairs is 0.28 m on HM3D, 1.04 raw SLAM units (5.2 m after the supplied scale recovery) on Insta360, and 1.86 m on 360Loc. The corresponding within-pair baselines are 0.12 m, 0.15 raw SLAM units (0.74 m), and 0.46 m. Insta360 scene construction uses raw monocular-SLAM coordinates; HM3D and 360Loc poses are metric.

Evaluation details for Table 1. We compute LPIPS with AlexNet. The two-view Splatter-360 HM3D row differs from the other six rows on 5 of its 80 held-out frames. No method is evaluated on its input frames.

Scoring resolution. All methods use the same metric implementation. Common-grid scoring resamples predictions and ground truth to one resolution; render-native scoring instead resamples ground truth to each method's output resolution. HM3D and Replica use a common 512×1024 grid, matching the ground truth. On HM3D, PanoWalker and two-view PanSplat render at 1024×2048 before downsampling, while the other five rows render at 512×1024. All 360Loc metrics and Insta360 WS-PSNR and SSIM use render-native scoring, with ground truth resampled from 3072×6144 and 2048×4096, respectively. Insta360 LPIPS uses a common 512×1024 grid. Cross-resolution comparisons for every metric depend on the scoring grid.

Replica. We regenerate 10 scenes and trajectories with Habitat following the Splatter-360 protocol, rather than using the released evaluation set; all methods share eight target panoramas per scene. PanoWalker and Splatter-360 use the same frozen HM3D checkpoint; the other feed-forward methods use their released frozen checkpoints, with no Replica fine-tuning. PanSplat and two-view Splatter-360 share the same deterministic non-degenerate pair per scene (camera-center separation closest to 0.20 m). ODGS and 360GS use the 23–30 unique panoramas in the same 15-pair set, depth initialization, and 30k iterations.

On HM3D, Insta360, and 360Loc, ODGS and 360GS are trained per scene with 30 input panoramas, compared with the 29–30 unique panoramas consumed by PanoWalker at $K=15$.

MVSplat360 (GS) uses the Gaussian stage of the released DL3DV checkpoint, with diffusion refinement disabled. Five nearest same-direction 256×256 cubemap faces provide context; the Gaussian-rendered faces are stitched to ERP before scoring.

B Construction Cost and Stored Scene Size

Measured per-scene construction time on HM3D, 10 scenes. We measured construction time for PanoWalker and the two per-scene-optimization baselines; the four feed-forward baselines of Table 1 were not timed. All three measurements exclude rendering, metric computation, and model



Figure 5: **Three windows from one continuous Insta360 walk.** Each row compares the held-out panorama (left) with PanoWalker's rendering at the same camera (right). The three windows use the scene construction and per-pair blending described in Section 3; all panels show the complete ERP.

archiving, giving ten-scene means of 152 s for PanoWalker, 680 s for 360GS, and 1,185 s for ODGS. PanoWalker's measurement includes backbone inference and scene construction; the optimizer measurements include COLMAP-format dataset construction. The preprocessing boundaries and input counts therefore differ. PanoWalker uses 29–30 unique input panoramas at $K{=}15$. ODGS and 360GS timings use 16 input panoramas per scene and 30k iterations, whereas their quality in Table 1 uses the 30-panorama protocol.

HM3D stored-Gaussian count. $N_{\mathcal{G}}$ counts stored Gaussians in the final scene representation. PanoWalker retains every prediction, giving $N_{\mathcal{G}} = K \times 2 \times 512 \times 1024$ on HM3D. At $K{=}15$, this is 15,728,640 Gaussians (15.73 M).

For the Atrium D1 and Hall D1 segments of 360Loc (54 and 61 m, respectively), we measured 37.45 GB of stored Gaussian attributes after 60 inputs.

C Qualitative Examples and Scoring

In Figure 3, scores for panel (a)'s top two rows and panel (b)'s first row use complete 512×1024 equirectangular panoramas (ERPs), including where the displayed images are cropped. The remaining rows use 1080×1920 perspective images with a $70°$ horizontal field of view (all dimensions are height $\times$ width). PanoWalker renders these perspective views directly; ODGS and 360GS sample their 512×1024 ERPs at the same camera orientation. Scores describe individual held-out views and use unmasked ground truth; the white patch in panel (b)'s first row is a display-only privacy mask.

Figure 5 shows held-out panoramas from three windows of the same Insta360 recording, each reconstructed from 15 source pairs.

D PROGRESSIVE PREVIEW PROTOCOL AND ADDITIONAL RESULTS

Inputs and queries. The two 360Loc segments are Piatrium D0 and Concourse D2, where D0/D2 denote daytime capture sequences 0/2. We sample a pair of consecutive panoramas every four frames. Their accumulated input sets contain 5/10/15 pairs and cover approximately 7/17/27 m and 8/18/29 m, respectively. We select the held-out queries in advance and evaluate 2, 5, and 8 views at the three stages, timing only the first requested preview at each stage. Across the two walks, PanoWalker's requested previews average 18.57/18.30/19.05 dB WS-PSNR, respectively, whereas Table 2 averages all eight final query views. Figure 4 renders the same later camera from all three scene stages to show growth at a fixed location. This is an offline diagnostic beyond the early capture extent; no future images enter scene construction. Final queries lie 0.209–0.598 m from the nearest input center. Each segment has one measured run; no run-to-run uncertainty is estimated.

Causal construction and initialization. All methods receive the same images acquired so far and registered poses. PanoWalker predicts each of the 15 pairs exactly once and adds every Gaussian to a source-labelled bank. The shared frozen PanSplat MVS network supplies predicted depths using only currently available pairs; no ground-truth depth is used. Optimization from scratch initializes a new scene with 100k Gaussians at points sampled using the predicted depths of all inputs acquired so far. Refinement starts from the previous scene's Gaussian parameters and Adam optimizer state, adds 100k Gaussians initialized from the new ten images, and restarts each baseline's learning-rate and densification schedules using the spatial extent of the currently available cameras. It then optimizes both earlier and newly added Gaussians using all arrived inputs. At the first update, all optimizer runs initialize from scratch. This refinement procedure is our adaptation of the released baseline code. 360GS uses no layout prior.

Optimization and stopping. The native renderer, photometric objective, optimizer, and within-stage densification are retained. Both optimizing from scratch and refining the previous scene are evaluated for 30k iterations per update; this is a fixed reference schedule rather than a convergence guarantee or quality ceiling. We also evaluate refinement under a per-update time budget, with each stage inheriting its own preceding budgeted result. The per-update time budgets average 32.4, 40.5, and 53.1 s over the two segments at the three updates. These budgets are larger than PanoWalker's measured update times of 18.2, 17.9, and 20.5 s; the comparison does not use equal latency budgets. The budget includes dataset conversion and scene initialization. Small overruns arise from the final iteration and first preview and remain in the reported times. Neither stopping nor the state passed to the next stage is selected using held-out quality.

Timing and resolution. Each run uses one L40 and eight allocated CPU threads. We synchronize the GPU when measuring update latency, which includes new-pair decoding/prediction and CPU transfer, model loading and renderer initialization at the first stage, construction or optimization, and the first saved preview. PanoWalker's first-stage startup takes 1.83–1.92 s. A preceding 35.77 s warm-up is excluded, so the reported times measure warmed execution. The first-preview component includes rendering, area-downsampling, and image output. For baselines, depth-prediction time using only images acquired by that stage is measured separately and added to dataset conversion, initialization, optimization, and preview time. These component times are summed without assuming overlap. Pose estimation, initial framework/data preparation, depth-map export, scoring, additional evaluation renders, and scene archival are excluded. Ours uses native 1024×2048 inference/rendering, area-downsampled for scoring; optimizers use native 512×1024. Every score uses the complete 512×1024 ERP, including sky and camera-carrier regions.

Quality, memory, and limits. Table A2 reports both individual segments, including the higher SSIM on Concourse of ODGS refining the previous scene for 30k iterations per update (0.685 versus 0.673 for PanoWalker). The optimizer GPU allocations in Table A3 exclude the shared depth predictor. PanoWalker stores its Gaussian bank on the CPU, with peak resident memory of 23.19–23.49 GB across the two walks. At the last stage each stored scene has 53.19M Gaussians, and our first-preview component takes 4.67–5.30 s, compared with roughly 0.2–0.3 s for the optimized models. Our advantage concerns construction and repeated-preview waiting time, not rendering frame rate or constant-time updates. This experiment assumes supplied registered poses and synchronous preview requests.

We also test these fixed optimizer budgets with dataset conversion and scene initialization excluded from the timed optimization budget. The final WS-PSNR is 13.52 dB (ODGS) and 12.90 dB (360GS),

Table A2: **Per-segment progressive reconstruction.** Piatrium and Concourse denote the Piatrium D0 and Concourse D2 capture segments of 360Loc. WS denotes WS-PSNR (dB). Groups use the update strategies and budgets defined in Table 2. Total time covers all three updates and first previews; quality averages eight query views from the final 30-input scene.

Segment	Method	Total (min)	WS↑	SSIM↑	LPIPS↓
Predict new pairs (no scene optimization)					
Piatrium	**Ours**	0.96	18.76	0.654	0.261
Concourse	**Ours**	0.92	18.38	0.673	0.230
Optimize from scratch (30k iterations per update)					
Piatrium	ODGS	77.75	14.27	0.488	0.473
Concourse	ODGS	81.09	17.67	0.667	0.283
Piatrium	360GS	38.50	11.64	0.325	0.616
Concourse	360GS	47.19	11.57	0.228	0.653
Refine previous scene (30k iterations per update)					
Piatrium	ODGS	88.80	15.04	0.520	0.425
Concourse	ODGS	83.81	17.80	0.685	0.269
Piatrium	360GS	48.54	11.69	0.303	0.646
Concourse	360GS	55.41	11.39	0.215	0.670
Refine previous scene (time budget per update)					
Piatrium	ODGS	2.16	13.26	0.463	0.635
Concourse	ODGS	2.08	15.59	0.455	0.543
Piatrium	360GS	2.22	13.87	0.475	0.700
Concourse	360GS	2.07	13.48	0.372	0.889

Table A3: **Resources used in progressive reconstruction.** Groups match Table 2. GPU peak is the maximum allocated GPU memory across each method's two runs. For ODGS and 360GS, it excludes the shared depth predictor. PanoWalker stores its Gaussian bank on the CPU, so this peak does not include all system memory. Stored counts average the two final scenes.

Method	GPU peak (GB)	Stored Gaussians (M)
Predict new pairs (no scene optimization)		
Ours	4.06	53.19
Optimize from scratch (30k iterations per update)		
ODGS	5.44	1.13
360GS	5.05	2.25
Refine previous scene (30k iterations per update)		
ODGS	5.71	1.27
360GS	6.12	2.82
Refine previous scene (time budget per update)		
ODGS	1.26	0.36
360GS	0.98	0.32

with total times including setup of 157.0 and 193.3 s, respectively. More optimization does not invariably improve held-out quality: some checkpoints fit the sparse training views strongly while later held-out scores decrease. These results characterize the tested initialization and schedules, without ruling out improved optimizer adaptation.

Visualization protocol. The fixed camera in Figure 4 was chosen from trajectory geometry before the additional renders were scored. Its labels report cumulative times from the timed preview runs; the additional fixed-camera renders are untimed offline evaluations.

Figure 6 shows PanoWalker's scene growth at the three stages. Gaussian centers use RGB colors from their stored appearance coefficients, with a fixed camera, display volume, and scale for each segment. For display, we subsample points and suppress spatial outliers within each newly added batch. Retained sample identities, positions, and colors are then frozen: later stages add new batches without revising earlier plotted points. The display frame uses the complete route only for retrospective

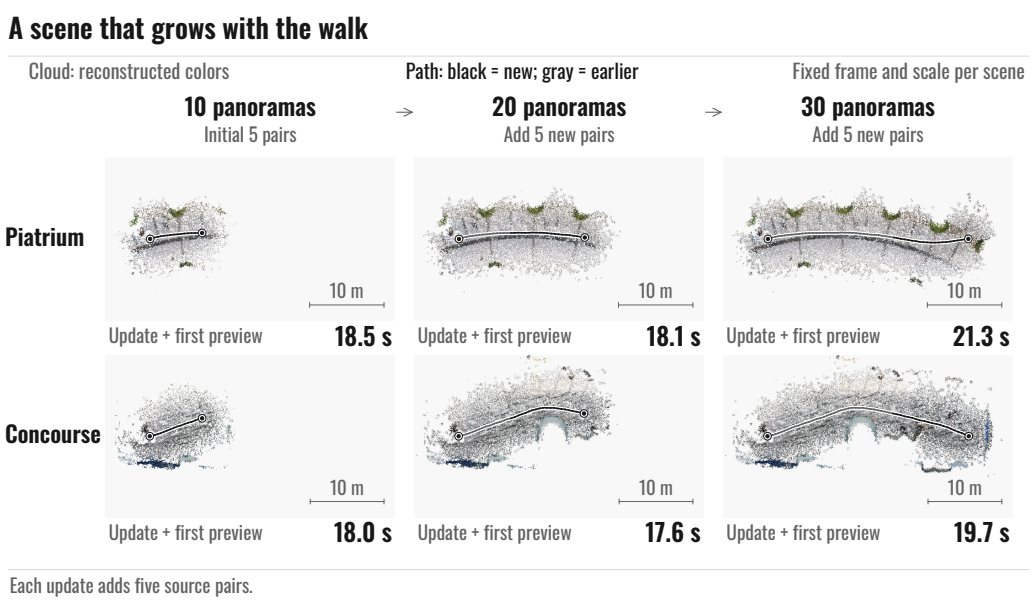

Figure 6: **Progressive scene growth on the 360Loc segments Piatrium D0 (top) and Concourse D2 (bottom).** PanoWalker adds five source pairs per update, reaching 10, 20, and 30 input panoramas from left to right. Colored points show reconstructed appearance in a fixed frame and scale per scene; black and gray paths denote newly acquired and earlier frames. Labels report the measured time for each update and its first preview, rather than cumulative time.

visualization, not scene construction. These display choices leave the stored Gaussian banks, rendered views, metrics, and measured update times unchanged. The panels illustrate accumulated scene structure, not quantitative surface coverage or reconstruction accuracy.

E Source-Weight Ablation and Rendering Controls

Using the source colors c_k and participation masks p_k defined in Section 3.3, Table 3 tests four color-averaging coefficients: p_k (neither weight), $p_k A_k$ (coverage only), $p_k w_k$ (distance only), and $p_k w_k A_k$ (full). Each color average divides by the coefficient sum with the same numerical safeguard. All variants retain the same opacity transformation, independent source renders, source-distance order, outer coverage composite, and low-coverage fallback. Thus, disabling coverage weighting does not disable visibility or coverage computation, and disabling distance weighting does not remove the fixed compositing order.

The near-input population uses all 10 HM3D benchmark scenes and eight queries per scene at $K{=}15$. Query-to-nearest-input distance ranges from 0 to 0.220 m, with median 0.020 m; 26 of 80 query centers coincide with an input center. One scene has equal source distances and serves as a control where distance weighting cannot distinguish the sources. Off-path evaluation uses 12 fixed queries in each of four HM3D scenes at $K{=}5, 10, 15$. Its 48 queries include 24 representative and 24 ground-truth-selected stress queries. These queries were used during method development, so the results are diagnostic rather than independent validation. All cases are retained, and means average queries within a scene before averaging scenes equally.

Scores use complete 512×1024 ERPs (height $\times$ width). Near-input 1024×2048 outputs use bilinear downsampling; off-path outputs use area downsampling. Both use the same AlexNet-based LPIPS model. The two populations are reported separately.

With coverage enabled, the distance-weight gains at off-path $K{=}5, 10, 15$ are $+0.089, +0.468, +0.415$ dB WS-PSNR. At $K{=}15$, the median is $+0.254$ dB and the worst query is -2.356 dB. The 0.5–1, 1–2, and 2–3 m nearest-input bins contain 16 queries each and average $+1.177, +0.146, -0.077$ dB. With distance enabled, coverage-weight gains across the

Table A4: **Alternative ways to render the same Gaussian scene.** Section E defines the three rendering rules; joint rendering also changes opacity and cross-source visibility. WS denotes WS-PSNR (dB). Off-path reports the WS-PSNR difference from the first row on 48 queries away from the capture path at 15 pairs. Old-view reports SSIM at 15 pairs minus SSIM at 5 pairs at fixed early query cameras on six 360Loc routes.

Rendering rule	HM3D PSNR↑	Replica PSNR↑	Insta360 WS↑	360Loc WS↑	Off-path ΔWS↑	Old-view ΔSSIM
Render all Gaussians together	30.33	26.93	19.12	19.00	0	−0.030
Render pairs, then overlay images	30.24	27.75	19.09	19.38	+1.07	0.000
PanoWalker (weighted color blend)	**30.53**	**28.12**	**19.27**	**19.61**	**+1.91**	**0.000**

three scene stages are $-0.024, +0.028, +0.103$ dB; LPIPS is slightly worse at the 5- and 10-pair stages. These stage and distance breakdowns do not add independent samples or establish universal improvement.

In Table A4, *Render all Gaussians together* uses one rasterization of the concatenated scene, with source-distance multipliers applied to Gaussian optical depths. Gaussians from different input pairs can therefore occlude one another. *Render pairs, then overlay images* instead renders each pair independently with the same opacity gain as PanoWalker, then combines the original color images by front-to-back alpha compositing (opacity-based image overlay). Here, ordering is by the distance from the query to the nearer input camera of each pair, with nearer pairs first; it is not a cross-pair depth sort of Gaussian points. PanoWalker uses these same pair renders and outer compositing steps but first replaces participating source colors with their distance- and coverage-weighted color average. The first control changes opacity and cross-source visibility together; Table 3 isolates the color-weight components.

F Source-Layer Rendering

From the unified scene to pixel colors. Figure 7 illustrates a kitchen view rendered from a unified HM3D scene containing 15 source pairs. Rasterizing each source component $\mathcal{B}_k$ at the query yields premultiplied color C_k and coverage A_k. Coverage reflects the projected Gaussian footprints, opacities, and within-source visibility. The camera-distance weight w_k is computed separately from the query's distance to the closer capture camera of each source pair, as in Eq. 4.

The color weights β_k combine camera distance and coverage as in Eq. 7, giving the shared color $b = \sum_k \beta_k c_k$. All 15 sources are rendered; at each pixel, layers above the coverage threshold contribute to b and use it in the final coverage composite, while other layers retain their own colors. The figure shows the three source layers with the largest mean color weights over the full panorama, selected without ground truth.

The broader rendering comparison in Table A4 improves mean WS-PSNR by 1.911 dB over rendering all Gaussians together across 48 off-path development queries at $K{=}15$, with 46 queries improving. One query drops from 14.526 to 13.440 dB. This comparison also changes opacity and cross-source visibility; Table 3 isolates the color-weight components.

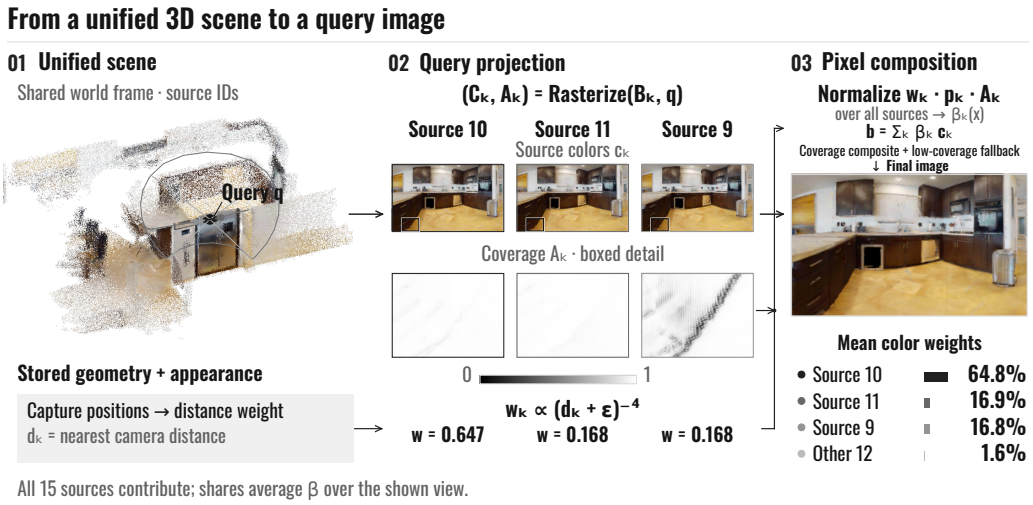

Figure 7: **Rendering from a unified Gaussian scene.** Left to right: the stored scene and query camera, three of the 15 projected source layers, and their contributions to the final image. Point colors show reconstructed appearance. Point subsampling and outlier removal are for display only and do not affect the stored scene or rendered images. The outline marks the shown angular region. Coverage panels enlarge the boxed RGB regions on a shared 0–1 scale. Percentages are mean β_k over the entire shown kitchen crop, not fractions of Gaussian points.