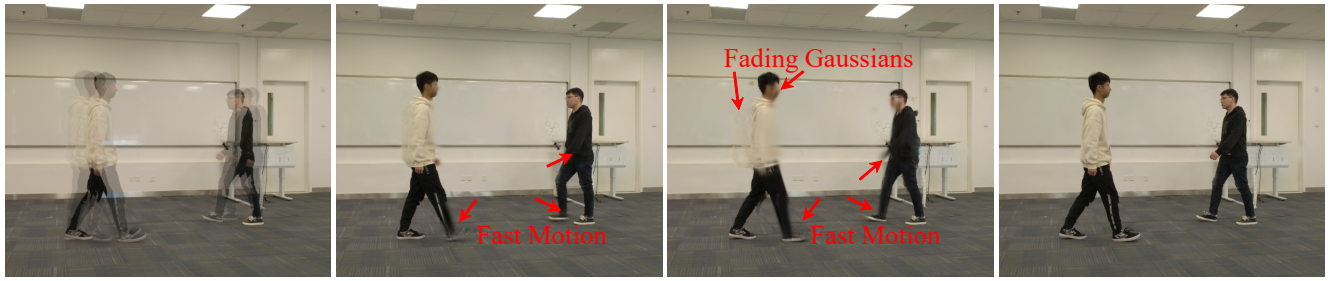
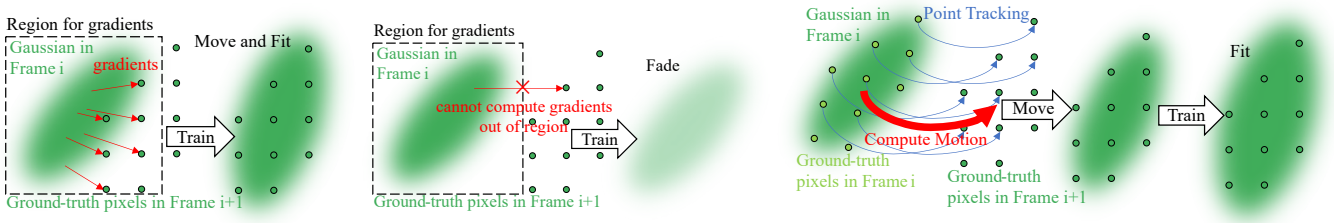
TrackerSplat: Exploiting Point Tracking for Fast and Robust Dynamic 3D Gaussians Reconstruction

DAHENG YIN, Simon Fraser University, Canada
ISAAC DING, Simon Fraser University, Canada
YILI JIN, McGill University, Canada and Simon Fraser University, Canada
JIANXIN SHI, Nankai University, China and Simon Fraser University, Canada
JIANGCHUAN LIU, Simon Fraser University, Canada



(a) Ground truth frame 61, 64, 67. (b) Frame 64 by Dynamic3DGS. (c) Frame 67 by Dynamic3DGS. (d) Frame 67 by our method.



(e) Gaussian fits for slow motion. (f) Gaussian fades for fast motion. (g) Basic idea of TrackerSplat

Fig. 1. Illustration of the motivation and basic idea of TrackerSplat. (a) Ground truth from the "walking" sequence. (b), (c) Rendered frames 64 and 67, trained 1000 iterations from frame 61 with the physically-based regularization losses introduced by Dynamic3DGS [Luiten et al. 2024]. Gaussians struggle to accurately follow the fast moving object, resulting in fading or incorrect recoloring. (e) Slow motion: object remains in the region for gradient computing, allowing Gaussians to maintain consistent color and follow the movement of object. (f) Fast motion: the object moves outside the region for gradient computing, position gradients fail to align with the movement of object, causing Gaussians to either fade or incorrectly recolor. (d), (g) TrackerSplat to adjust Gaussian position, rotation and scaling parameters according to point tracking results before training, enabling stable and robust training for fast-moving objects.

Recent advancements in 3D Gaussian Splatting (3DGS) have demonstrated its potential for efficient and photorealistic 3D reconstructions, which is crucial for diverse applications such as robotics and immersive media. However, current Gaussian-based methods for dynamic scene reconstruction struggle with large inter-frame displacements, leading to artifacts and temporal inconsistencies under fast object motions. To address this, we introduce *TrackerSplat*, a novel method that integrates advanced point tracking methods to enhance the robustness and scalability of 3DGS for dynamic scene reconstruction. TrackerSplat utilizes off-the-shelf point tracking models to extract pixel trajectories and triangulate per-view pixel trajectories onto 3D Gaussians to guide the relocation, rotation, and scaling of Gaussians before training. This strategy effectively handles large displacements between frames, dramatically reducing the fading and recoloring artifacts prevalent in prior methods. By accurately positioning Gaussians prior to gradient-based optimization, TrackerSplat overcomes the quality degradation associated with large frame gaps when processing multiple adjacent frames in parallel across multiple devices, thereby boosting reconstruction throughput while preserving rendering quality. Experiments on real-world datasets confirm the robustness of TrackerSplat in challenging scenarios with significant displacements, achieving superior throughput under parallel settings and maintaining visual quality compared to baselines. The code is available at https://github.com/yindaheng98/TrackerSplat.

CCS Concepts: • **Computing methodologies** → **Rendering**; **Image-based rendering**; *Point-based models*; *Motion capture*.

Additional Key Words and Phrases: Point Tracking, 3D Gaussian Splatting

Authors' Contact Information: Daheng Yin, Simon Fraser University, Burnaby, Canada, dya64@sfu.ca; Isaac Ding, Simon Fraser University, Burnaby, Canada, isaac_ding@sfu.ca; Yili Jin, McGill University, Montreal, Canada and Simon Fraser University, Burnaby, Canada, yili.jin@mail.mcgill.ca; Jianxin Shi, Nankai University, Tianjin, China and Simon Fraser University, Burnaby, Canada, jxshi@nankai.edu.cn; Jiangchuan Liu, Simon Fraser University, Burnaby, Canada, jcliu@cs.sfu.ca.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SA Conference Papers '25, Hong Kong, Hong Kong
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2137-3/2025/12
https://doi.org/10.1145/3757377.3763829

ACM Reference Format:
Daheng Yin, Isaac Ding, Yili Jin, Jianxin Shi, and Jiangchuan Liu. 2025. TrackerSplat: Exploiting Point Tracking for Fast and Robust Dynamic 3D Gaussians Reconstruction. In *SIGGRAPH Asia 2025 Conference Papers (SA Conference Papers '25), December 15–18, 2025, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3757377.3763829

1 Introduction

Reconstructing dynamic 3D scenes and generating photo-realistic, temporally consistent renderings have long been fundamental goals in computer vision and graphics. These capabilities are increasingly important for creating controllable, editable, high-quality 3D content, underpinning applications in film, gaming, and the metaverse [Zhang et al. 2021]. Beyond visual fidelity for human audiences, a more critical requirement emerges in embodied AI and robotics: capturing the trajectories of objects and agents in dynamic environments. For robots, temporally coherent reconstruction provides the motion cues necessary for downstream tasks such as object manipulation, navigation, and human-robot interaction [Abou-Chakra et al. 2024]. It also enables reliable tracking and prediction of surrounding agents in applications like autonomous driving [Zhou et al. 2024]. Despite these increasing demands, efficient and accurate reconstruction of dynamic scenes while preserving consistent object trajectories remains challenging due to the complexities introduced by temporal dynamics and diverse motion patterns.

Recent progress in 3D reconstruction has been driven notably by the success of 3D Gaussian Splatting (3DGS)[Kerbl et al. 2023] for its ability to efficiently represent 3D scenes with photorealism. By modeling 3D space with ellipsoids ("Gaussians"), 3DGS enables intuitive editing through manipulation of individual Gaussians, making it suitable for dynamic scene representation. Building on its strengths, follow-up studies has adapted 3DGS to dynamic scenes by explicitly encoding Gaussian parameters as temporal trajectories [Li et al. 2024a; Lin et al. 2024] or by representing motion fields using implicit features [Li et al. 2024b; Wu et al. 2024]. These methods typically rely on frame-to-frame adaptation, iteratively refining Gaussian parameters by training on consecutive frames to ensure smooth temporal transitions [Gao et al. 2024; Luiten et al. 2024].

Despite these advancements, the reconstruction process of 3DGS is computationally intensive, limiting its application in scenarios demanding both high quality and high throughput, such as live streaming and interactive virtual environments. To improve the throughput of reconstruction without adding end-to-end latency, a natural solution is to process multiple adjacent frames in parallel across multiple GPUs. However, our experiments reveal that existing methods suffer from significant quality degradation when handling large displacements between frames, leading to visible artifacts, as is shown in Figure 1b and Figure 1c.

Upon further analysis, we identify a critical issue in 3DGS that contributes to this quality degradation. Existing approaches rely heavily on fine-tuning Gaussian parameters from frame to frame using iterative training. A core idea behind these methods is to guide Gaussian motion using position gradients computed by comparing Gaussian colors with the surrounding pixels they overlap (Figure 1e). Due to computational constraints, gradient computations are restricted to a limited local neighborhood. This constraint results in inaccurate position gradients when objects experience significant inter-frame motion and move outside this restricted region (Figure 1f). In parallel setups, increased parallelism widens the frame gaps assigned to each device, amplifying the likelihood of significant object displacements and consequently exacerbating this issue, leading to prominent artifacts (Figure 1c).

To mitigate this limitation, we propose directly estimating Gaussian trajectories across frames rather than relying solely on the gradient to update their positions. Recent advancements in point tracking [Karaev et al. 2024, 2025] provide robust pixel-level motion estimation across video frames. However, integrating point tracking presents two key challenges: (1) translating 2D pixel trajectories into updates for 3D Gaussian parameters, and (2) mitigating inaccuracies in pixel trajectories to prevent error accumulation during updates.

We introduce *TrackerSplat* to address these challenges. As illustrated in Figure 1g, TrackerSplat integrates an off-the-shelf point tracking model to capture pixel trajectories for each viewpoint. To compute updates for 3D Gaussians, we propose Parallel Weighted Incremental Least Squares (PWI-LS) that derives 2D motion from pixel trajectories. These 2D motions from multiple views are triangulated to update Gaussian positions, rotations, and scales. To reduce inaccuracies, the computed updates are smoothed by Motion Regularization, and Gaussian parameters are further refined through training. By repositioning Gaussians closer to their correct locations before training, TrackerSplat maintains coherent tracking despite large displacements, significantly reducing fading or recoloring artifacts observed in prior methods. Most importantly, since tracking mitigates the impact of large frame gaps, TrackerSplat enables independent frame updates across multiple GPUs, thus increasing reconstruction throughput without sacrificing quality.

To the best of our knowledge, we are the first to identify the robustness limitations of 3DGS in handling large inter-frame displacements, and TrackerSplat is the first method to directly compute Gaussian trajectories using multi-view point tracking results to address this limitation. While prior works have incorporated tracking within Gaussian-based pipelines [Lei et al. 2025; Stearns et al. 2024], direct use of multi-view point tracking for trajectory estimation remains unexplored.

We implement TrackerSplat with a parallel pipeline across 8 GPUs and evaluate it on real-world dynamic scene datasets. Our extensive experiments demonstrate superior throughput under parallel settings, while preserving or improving visual quality compared to baselines. Our findings confirm the effectiveness of incorporating point tracking into 3DGS-based dynamic reconstruction, paving the way for scalable, accurate, and temporally consistent dynamic 3D scene reconstructions.

2 Related Work

2.1 3D Gaussian Splatting for Dynamic Scenes

Recent years have witnessed significant progress in reconstructing 3D representations from multi-view captures. Among these advancements, 3D Gaussian Splatting (3DGS) [Kerbl et al. 2023] has emerged as a leading approach. 3DGS represents scenes using a set of ellipsoids ("Gaussians") and achieves photorealistic rendering with high efficiency. Building on its success, the latest studies have extended

3DGS to dynamic scenarios. Some methods dynamically add or remove Gaussians to represent motion through their appearance and disappearance [Duan et al. 2024; Sun et al. 2024]. However, this approach may lead to significant storage overhead due to the large number of Gaussians needed to capture the dynamic nature of the scene. To address this, other methods explicitly represent dynamic scenes using Gaussians and their trajectories, significantly reducing the number of Gaussians. For instance, some approaches use implicit features conditioned on time to represent motion fields [Gao et al. 2024; Li et al. 2024b,a; Lin et al. 2024; Luiten et al. 2024; Wu et al. 2024], while others adopt triplane representations for higher spatial and temporal resolutions [Wu et al. 2024]. Other advancements also include frame-to-frame adaptation techniques, where Gaussian parameters are iteratively refined by training on consecutive frames [Gao et al. 2024; Luiten et al. 2024; Xu et al. 2024].

2.2 Point Tracking

Point tracking [Seidenschwarz et al. 2025; Wang et al. 2023] identifies the position and visibility of specific pixels across video sequences (Figure 2), providing robust trajectory estimation even under challenging conditions such as occlusion. Recently, point tracking has attracted considerable attention within the computer vision community. Empowered by semi-supervised correspondence, CoTracker [Karaev et al. 2024, 2025] achieves state-of-the-art sparse tracking performance, while DOT [Le Moing et al. 2024] further enhances dense tracking accuracy and efficiency. Point tracking has been used in 3D reconstruction, especially for reconstructing dynamic scenes from monocular videos [Lei et al. 2025; Stearns et al. 2024]. In these methods, point tracking typically separates static background from dynamic foreground or serves as a regularization term for dynamic regions. However, the direct application of point tracker results to compute 3D Gaussian splatting parameters for multi-view dynamic scene reconstruction remains largely unexplored.

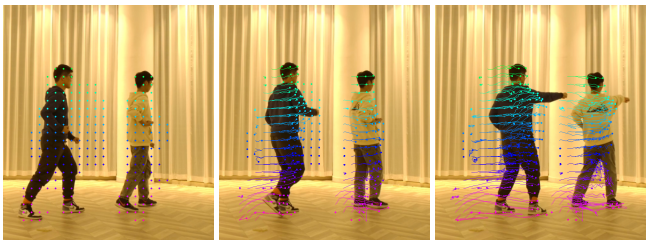


Fig. 2. DOT point tracking on a video sequence. Colored lines show pixel trajectories over time.

3 Preliminaries

3.1 Mathamatical Reperesentation of 3D Gaussians

3DGS represents the scenes with 3D Gaussians. Each 3D Gaussians is characterized by two key components: its mean $\boldsymbol{\mu}_{3D}$ represents the position of the ellipsoid, and its covariance matrix $\Sigma_{3D} = RSS^\top R^\top$, composed of a scaling matrix S and a rotation matrix R, describes the spread and orientation of the Gaussian ellipsoid respectively.

When projected onto the 2D image plane, the 3D Gaussian becomes a 2D Gaussian distribution. Concretely, for a point on the image plane, the density function of the 2D Gaussian distribution can be represented as:

$$\begin{aligned} G(\boldsymbol{x}) &= e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_{2D})^\top \Sigma_{2D}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_{2D})} \\ \Sigma_{2D} &= JW\Sigma_{3D}W^\top J^\top \\ \boldsymbol{\mu}_{2D} &= \frac{1}{z}PW\boldsymbol{\mu}_{3D} \end{aligned} \tag{1}$$

where J is the Jacobian of the affine approximation of the projective transformation, W is the viewport transformation matrix, P is the projection transformation matrix, and z is the depth value in $PW\boldsymbol{\mu}_{3D}$. After projection, the color of each pixel is calculated by alpha-blending each Gaussian according to its depth.

3.2 Integration of Point Tracking

To track 3D Gaussians, we rely on point tracking in video frames captured from multiple viewpoints. In particular, we employ Dense Optical Tracking (DOT) [Le Moing et al. 2024], a simple yet efficient method for point tracking. For each pixel i located at position $\boldsymbol{x}_i$ in the first frame, point tracking estimates its corresponding position $(\boldsymbol{x}_i \mapsto \boldsymbol{x}_i')$ in any subsequent target frame.

4 Method

4.1 Overview

We provide an overview of TrackerSplat in Figure 3. Our goal is to reconstruct dynamic scenes from video clips captured from multiple fixed viewpoints by tracking and refining a set of Gaussian representations. Our methods can be divided into four stages:

Initialization. For the first frame, we initialize 3D Gaussians using existing reconstruction methods for static scenes, such as InstantSplat [Fan et al. 2024].

Point Tracking. For each subsequent frame, we employ a point tracking model to extract 2D trajectories for every pixel in the initial frame throughout the video clip.

Motion Compensation. In this stage, we compute the updated parameters of the 3D Gaussians based on the point tracking results. TrackerSplat first solves the Gaussian motions (Sec. 4.2) on the 2D image plane using *Parallel Weighted Incremental Least Squares (PWI-LS)* (Sec. 4.3) and then *updates Gaussians from multi-view observations* (Sec.4.4). To mitigate the impact of errors from point tracking and PWI-LS on quality, we introduce *Motion Regularization* (Sec. 4.5), which applies median filtering and propagates motion information according to neighboring Gaussians.

Refinement. Finally, we refine the parameters of each frame by training on the input video clips (Sec. 4.6).

In this section, we detail each step of the proposed approach and show how they can be parallelized on multi-GPU devices with our *Parallel Pipeline* (Sec. 4.7).

4.2 Definition of Gaussian Motion

To serve as the basis for tracking 3D Gaussians across multiple video viewpoints, it is necessary to define the motion of a 2D Gaussian distribution on the image plane. Consider a 2D Gaussian distribution $G(\boldsymbol{x})$ on a specific image plane in the first frame, characterized by

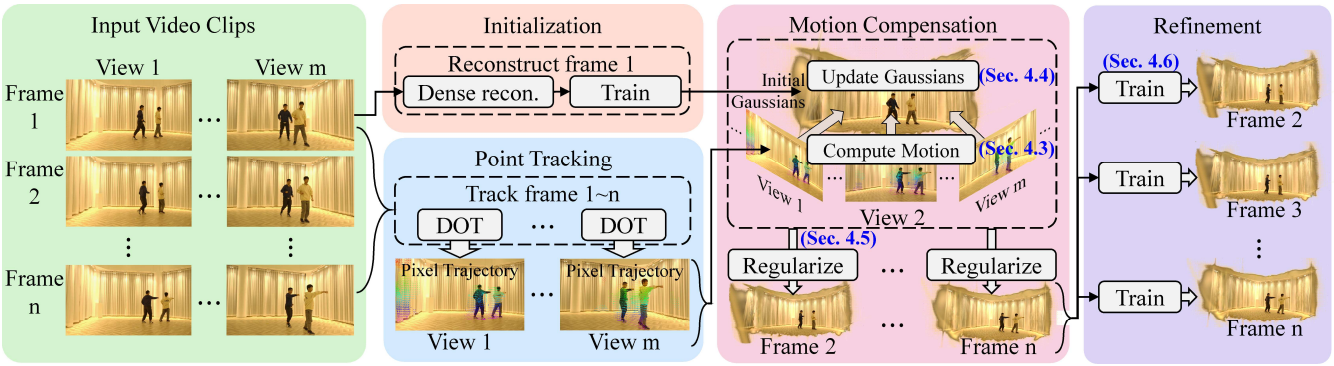


Fig. 3. TrackerSplat overview. Our method processes video clips captured from multiple fixed viewpoints. It begins by applying existing reconstruction techniques to initialize a set of 3D Gaussians for the first frame. For subsequent frames, the position, rotation, and scale of these Gaussians are updated based on point tracking across views, with their motions regularized by neighboring Gaussians. Finally, the Gaussian parameters of each frame are refined by training on input frames.

its covariance Σ_{2D} and mean $\boldsymbol{\mu}_{2D}$. We define the motion of this Gaussian as an affine transformation $[A|\boldsymbol{b}]$, which maps $G(\boldsymbol{x})$ to a new 2D Gaussian distribution $G'(\boldsymbol{x})$ in a subsequent frame. Formally, $G(\boldsymbol{x}) = G'(A\boldsymbol{x} + \boldsymbol{b})$. Under this affine transformation, the updated mean $\boldsymbol{\mu}'_{2D}$ and covariance Σ'_{2D} can be derived as follows:

$$\begin{aligned} \Sigma'_{2D} &= A\Sigma_{2D}A^\top \\ \boldsymbol{\mu}'_{2D} &= A\boldsymbol{\mu}_{2D} + \boldsymbol{b} \end{aligned} \tag{2}$$

4.3 Parallel Weighted Incremental Least Squares (PWI-LS)

4.3.1 Problem Formulation. Let $\{\boldsymbol{x}_i\}, i \in [1, n]$ denote the collection of pixels on a specific image plane covered by the 2D Gaussian in the first frame. Our goal is to find the affine transformation $[A|\boldsymbol{b}]$ that best aligns these pixel coordinates $\boldsymbol{x}_i$ with their tracked positions $\boldsymbol{x}'_i$ in the subsequent frame. This can be formulated as follows:

$$\min_{A,\boldsymbol{b}} \sum_{i=1}^{n} ||A\boldsymbol{x}_i + \boldsymbol{b} - \boldsymbol{x}'_i||^2$$

4.3.2 Naive Least Squares. A straightforward method to solve this optimization is through least squares. By stacking all point pairs $(\boldsymbol{x}_i \mapsto \boldsymbol{x}'_i)$ into matrices X and Y, the affine transformation $[A|\boldsymbol{b}]$ can be computed as:

$$\begin{aligned} [\hat{A}|\hat{\boldsymbol{b}}] &= (X^\top X)^{-1} X^\top Y \\ X &= \begin{bmatrix} x_1 & \dots & x_i & \dots & x_n \\ 1 & \dots & 1 & \dots & 1 \end{bmatrix}^\top \\ Y &= \begin{bmatrix} x'_1 & \dots & x'_i & \dots & x'_n \end{bmatrix}^\top \end{aligned} \tag{3}$$

However, in cases where many pixels fall under each Gaussian and when multiple Gaussians overlap, explicitly constructing and inverting these large matrices per Gaussian is computationally expensive. This motivates a more efficient, incremental approach.

4.3.3 Incremental Least Squares. To mitigate high computational costs, we exploit the additive structure of $X^\top X$ and $X^\top Y$ in Equation 3 and decompose them into per-pixel contributions:

$$\begin{aligned} X^\top X &= \sum_{i=1}^{n} P_i = \sum_{i=1}^{n} \begin{bmatrix} x_i \\ 1 \end{bmatrix} \cdot [x_i^\top, 1] \\ X^\top Y &= \sum_{i=1}^{n} Q_i = \sum_{i=1}^{n} \begin{bmatrix} x_i \\ 1 \end{bmatrix} \cdot x_i'^\top \end{aligned}$$

By expressing the solution as sums of P_i and Q_i, the contribution of each pixel can be handled independently. This incremental scheme avoids building full matrices for all pixels, allowing us to accumulate partial results P_i and Q_i in parallel, paving the way for efficient parallel implementations.

4.3.4 Weighted Least Squares. In regions of partial coverage (e.g., object boundaries with overlapping foreground/background Gaussians), unweighted alignment can be misled by pixels that do not truly belong to a particular motion. For example, consider a static background Gaussian and a high-density moving Gaussian in front of it. The moving Gaussian may partially cover a moving pixel with very high opacity, and a static pixel with very low opacity. In this situation, the static pixel should not contribute to the motion of the moving Gaussian. To achieve this, we adopt a weighted formulation:

$$\begin{aligned} [\hat{A}|\hat{\boldsymbol{b}}] &= V_1^{-1} V_2 \\ V_1 = \sum_{i=1}^{n} w_i P_i \quad V_2 &= \sum_{i=1}^{n} w_i Q_i \end{aligned} \tag{4}$$

where each pixel carries a weight w_i representing the likelihood that its motion corresponds to a particular Gaussian. In our implementation, w_i is set to $\alpha_i T_i$, where α_i and T_i are the opacity and the transparency of Gaussian at the pixel i in alpha-blending. This ensures that boundary pixels with low opacity contribute less to the accumulated statistics, thereby reducing sensitivity to irrelevant motions from overlapping regions.

4.3.5 Acceleration by GPU. The Weighted Incremental Least Squares algorithm can be optimized for GPU execution to exploit its parallel processing capabilities for acceleration, which results in the *Parallel Weighted Incremental Least Squares* algorithm:

For each Gaussian, we allocate GPU memory for the (3×3) matrix V_1 and the (3 × 2) matrix V_2. We modify the rendering process to compute V_1 and V_2, where the $w_i P_i$ and $w_i Q_i$ are computed in parallel and aggregated by atomic addition into the corresponding V_1 and V_2. After processing all pixels, the motion $[\hat{A}|\hat{\boldsymbol{b}}]$ of each Gaussian are computed by Equation 4 to obtain affine transformation $[A|\boldsymbol{b}]$ of each Gaussian. To avoid numerical instability, we discard the motion of Gaussians that cover fewer than three pixels, or have a near-singular matrix V_1.

4.4 Update Gaussians from Multi-view Observations

Once we have motion $[A|\boldsymbol{b}]$ of each Gaussian in multiple views, we can compute its updated covariance matrix Σ'_{2D} and 2D mean $\boldsymbol{\mu}'_{2D}$ according to Equation 2, and then the updated 3D covariance matrix Σ'_{3D} and 3D mean $\boldsymbol{\mu}'_{3D}$ can be derived from the multi-view 2D means and covariance.

4.4.1 Compute 3D Mean. Determining the 3D mean $\boldsymbol{\mu}'_{3D}$ from the 2D means $\boldsymbol{\mu}'_{2D}$ across multiple views is a typical triangulation problem. With known camera intrinsic and extrinsic, at least two viewpoints are required to compute the $\boldsymbol{\mu}'_{3D}$ of a Gaussian. We solve this triangulation problem using Singular Value Decomposition (SVD). To maintain numerical stability, results are discarded for Gaussians observed in fewer than three views or those with low accumulated alpha values.

4.4.2 Compute 3D Covariance Matrix. Given the covariance matrices Σ'_{2D} from multiple views and the parameters J and W of these views, the relationship $\Sigma'_{2D} = JW\Sigma'_{3D}W^\top J^\top$ in Equation 1 yields a linear system in Σ'_{3D}. Each Σ'_{2D} contributes three constraints, while Σ'_{3D} has six unknown parameters. Hence, at least two distinct views are required to solve for Σ'_{3D}.

4.4.3 Decompose Covariance into Rotation and Scale. According to Equation 1, the 3D covariance matrix Σ_{3D} comprises a rotation matrix R and a scaling matrix S. To update these parameters, we perform eigen decomposition on the modified covariance matrix Σ'_{3D}, extracting the updated rotation matrix R' and scaling matrix S'. In eigen decomposition, the eigenvector matrix corresponds to the rotation matrix, while the eigenvalues represent the squared scaling matrix.

However, naively using eigen decomposition can lead to two problems: 1) Negative eigenvalues cannot be square-rooted, making them unsuitable for computing the scaling matrix. 2) Eigenvalues are sorted from largest to smallest, which can disrupt the consistent order of rotation vectors and scaling factors between frames. This inconsistency can cause adjacent regions to appear to have similar relative rotations but significantly different rotation matrices, leading to instability in subsequent motion propagation.

To resolve these issues, we discard those Σ'_{3D} with negative eigenvalues, and reorder the eigenvalues and corresponding eigenvectors to ensure their magnitudes match the order from the first frame.

4.5 Motion Regularization

Pixel tracking and partial observations can introduce outlier Gaussians or wrong motions, leading to noticeable deviations. We introduce a motion regularization to address these issues. Specifically, we apply median filtering based on K-nearest neighbors to smooth the motion and then propagate the motion to the neighbor Gaussians that are not determined to be static.

4.5.1 Median Filtering. We observe that the majority of Gaussians are stable but a few are outliers. This situation is similar to salt-and-pepper noise in image processing. Therefore, we heuristically apply a median filtering approach. Take $\Delta\mu_{3D} = \mu'_{3D} - \mu_{3D}$, $\Delta R = R' - R$, and $\Delta S = S' - S$ as the difference of a Gaussian between its first frame. For each Gaussian, we find its K nearest neighbors in 3D space, and then compute the median of their $\Delta\mu_{3D}$, ΔR, and ΔS and add these median values to the Gaussian parameters for the update.

4.5.2 Propagation. The motion of certain Gaussians may not be reliably determined, for example, due to low visibility (e.g., low opacity or visible in too few views) or negative eigenvalues arising from eigen decomposition. For these Gaussians, we apply the motion propagation that propagates the motion from their neighbor to them. Before propagation, we first figure out those static Gaussians by checking whether the pixels they cover are moving or not. Specifically, we treat the pixels with the movement $|x'_i - x_i|$ given by the point tracking model as less than 1 pixel as static pixels, count them, and accumulate their alpha for each Gaussian in the rendering process. The rule for detecting the static Gaussians can be varied. In this paper, we take those Gaussians hit by more than 9 pixels and have more than 90% of its pixels fixed in at least 2 views as static Gaussians, discard their computed motion, and exclude them from the motion propagation. We then compute the average of their $\Delta\mu_{3D}$, ΔS and an average of rotations ΔR in Euler angle form and add these median values to the Gaussian parameters μ_{3D}, S and R.

4.6 Refinement

Even with point tracking and multi-view constraints, errors may persist in the recovered Gaussians. We hence run a final refinement step by training the Gaussian parameters with the input video frames. As the Gaussians have been moved to an approximately right position, the training process would be very fast and stable.

4.7 Parallel Pipeline

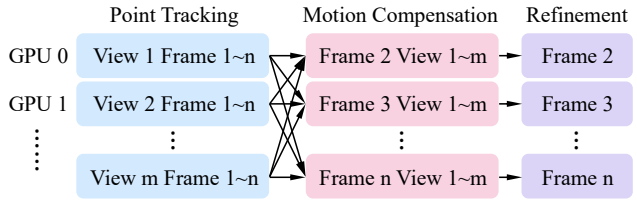


Fig. 4. TrackerSplat parallel pipeline.

As illustrated in Figure 4, three key stages of TrackerSplat can be executed in parallel. Since our point tracking operates per video clip, the point tracking can be parallelized by processing video from each view independently. Additionally, the Motion Compensation and Refinement are both per-frame operations, they can be parallelized by processing each frame independently, only depending on the

results of Point Tracking and initialization of the first frame. The parallel pipeline can be implemented on multi-GPU systems and significantly improve the throughput of TrackerSplat.

5 Experiments

5.1 Datasets

We evaluate our method on four widely-used dynamic scene datasets:
Meeting Room dataset [Li et al. 2022a] includes 4 scenes, each of 300 frames captured from 13 viewpoints at a resolution of 1280×720.
Neural 3D Video Synthesis (**N3DV**) dataset [Li et al. 2022b] contains 6 scenes, each of 300 frames captured from 18 views at 2704×2028 resolution.
Dynamic3DGS dataset [Luiten et al. 2024] comprise 6 scenes (150 frames each) captured from 27 views at 640×360 resolution.
st-nerf dataset [Zhang et al. 2021] consists of 3 scenes, each contains 75–100 frames from 15 views at 1920×1080 resolution.
RH20T dataset [Fang et al. 2024] includes 110k robotic manipulation sequences from 8-10 views at 640x360 resolution. For evaluation, we selected 14 videos and extracted the longest continuous multi-view segments synchronized within 100ms across views.

We use these datasets in two experiment setups:
Short-clip experiments: Videos are segmented into non-overlapping clips of 2, 3, 5, or 9 frames. The first frame of each clip is reconstructed in the initialization stage, and subsequent frames are processed in parallel using 1, 2, 4, or 8 GPUs, respectively. This setup evaluates quality degradation caused solely by increased parallelism without cumulative errors across clips.
Long-video experiments: Clips are sequentially connected, with the last frame of each clip serving as the first frame of the next. Only the first frame of the first clip is reconstructed in the initialization stage. This setup evaluates robustness and temporal consistency over longer sequences.

5.2 Implementation Details

We carefully evaluated various hyperparameter settings and selected those achieving the best balance between quality and efficiency.
Initialization. Following InstantSplat [Fan et al. 2024], we initialize Gaussians from COLMAP point clouds and optimize Gaussian parameters for 10,000 iterations, providing a stable starting point for subsequent frames.
Point Tracking. We evaluated DOT [Le Moing et al. 2024], CoTracker [Karaev et al. 2024, 2025], and TAPIR [Doersch et al. 2023]. DOT provides similar accuracy at much higher speed and was therefore selected. To improve efficiency, input images are resized (Meeting Room & Dynamic3DGS: 640×360, N3DV: 800×600, st-nerf: 960×540).
Regularization. Gaussians are marked unsolvable in PWI-LS if their V_1 determinant is below 10^{-12}, their accumulated alpha is below 10^{-3}, or they cover fewer than 2 pixels. Gaussians are marked unsolvable during multi-view updates if visible in fewer than 2 views, accumulated alpha is below 10^{-3}, or covering fewer than 3 pixels across all views. In the regularization stage, parameters of unsolvable Gaussians are updated based on their 8 nearest neighbors.
Refinement. In the refinement stage, Gaussian parameters are optimized following the original 3DGS method [Kerbl et al. 2023] for 1,000 iterations without densification.
Runtime. Our implementation is built on PyTorch, Taichi [Hu et al. 2019], and CUDA. All experiments were conducted on a server with 8 NVIDIA A100-SXM4-40GB GPUs.

5.3 Baseline and Ablation

We compare TrackerSplat with representative dynamic scene reconstruction methods that support Gaussian trajectory tracking from multi-view videos with fixed camera poses. The original implementations of these methods do not directly support multi-GPU parallel processing. Thus, to ensure a fair comparison, we carefully copy and adapt their codebases to our parallel framework, strictly preserving their original designs and hyperparameters to minimize any deviation from their original implementations:
Parallel HiCoM. Hierarchical Coherent Motion (HiCoM) [Gao et al. 2024] associates Gaussian motions with distinct regions. In their open-source code, HiCoM is implemented as a hierarchical grid with multiple density levels. We copy and adapt this HiCoM grid implementation to our parallel framework.
Parallel Dynamic 3DGS. Dynamic 3DGS [Luiten et al. 2024] sequentially train Gaussian parameters frame-by-frame with physics-based regularization. We copy their regularization term and adapt their sequential training to parallel processing by initializing the training from the first frame of each clip rather than previous frame.
Parallel 4DGS. 4DGS [Wu et al. 2024] introduces a deformation field to represent Gaussian motion. We copy and adapt the deformation field implementation for parallel processing by training a separate deformation field per frame, warping Gaussians from the initial frame to subsequent frames within each clip.
Parallel ST-4DGS. ST-4DGS [Li et al. 2024b] extends 4DGS by incorporating explicit temporal regularization. We copy and adapt their regularization term to our framework with 4DGS implementation.
TrackerSplat without regularization (Ablation). As an ablation study, we remove the regularization stage and directly trained the Gaussian parameters based on the motion compensation results.

We observed that the regularization in ST-4DGS, 4DGS, and Dynamic 3DGS, originally tuned for small motions, tends to over-constrain Gaussians in our parallel setting, producing severe artifacts (e.g., sticking and fading) that obscure the true performance of these baselines. To ensure a fair comparison, we reduce their regularization weights, which significantly alleviates these artifacts. HiCoM is less affected because its regularization is weaker and more localized. Moreover, since our refinement stage is essentially training without regularization, including HiCoM in our evaluation highlights that the robustness of TrackerSplat is not solely due to the absence of regularization during refinement. All baselines share the same initial frame for both short-clip and long-video experiments.

5.4 Comparison of Visual Quality

We evaluate rendering quality using three widely accepted metrics: structural similarity (SSIM), peak signal-to-noise ratio (PSNR), and perceptual similarity (LPIPS) [Zhang et al. 2018].

Table 1. Quantitative comparison of average visual quality (*PSNR* ↑ / *SSIM* ↑ / *LPIPs* ↓) in short-clip experiments under varying GPU parallelism settings (1, 2, 4, and 8 GPUs). Our method achieves better visual quality in most cases and demonstrates greater robustness than baselines as parallelism increases. Results for all other scenes are included in the supplementary material.

Method	GPUs	"basketball"	"boxes"	"juggle"	"stepin"	"vrheadset"	"taekwondo"	"coffee martini"
P. ST-4DGS	1	30.6 / .941 / .074	30.4 / .946 / .064	30.9 / .950 / .062	30.6 / .885 / .104	32.4 / .950 / .067	36.0 / .976 / .022	27.6 / **.918** / **.114**
P. 4DGS	1	30.5 / .940 / .072	30.6 / .948 / .064	30.8 / .951 / .063	32.6 / .950 / .068	32.5 / .951 / .066	36.3 / .977 / .022	27.5 / .917 / .115
P. Dy.3DGS	1	30.9 / .942 / .075	31.5 / .951 / .061	31.7 / .953 / .061	32.4 / .948 / .070	32.4 / .951 / .067	36.6 / .978 / .021	27.6 / .917 / .115
P. HiCoM	1	31.4 / .943 / .074	31.6 / .949 / .065	31.9 / .951 / .063	32.9 / .951 / .067	32.9 / .953 / .064	**36.9** / **.979** / **.020**	**28.0** / .917 / .116
Ours	1	**32.6** / **.950** / **.066**	**32.4** / **.953** / **.060**	**32.6** / **.956** / **.058**	**32.9** / **.952** / **.065**	**33.0** / **.954** / **.063**	36.8 / .978 / .020	27.9 / .917 / .115
P. ST-4DGS	2	30.2 / .936 / .081	30.4 / .944 / .066	31.0 / .951 / .063	32.1 / .947 / .071	32.2 / .950 / .067	35.0 / .971 / .027	27.6 / **.920** / **.110**
P. 4DGS	2	30.7 / .940 / .073	30.2 / .941 / .068	30.6 / .947 / .066	32.6 / .950 / .068	32.4 / .950 / .066	35.5 / .974 / .024	27.4 / .918 / .113
P. Dy.3DGS	2	30.2 / .936 / .083	31.3 / .949 / .063	31.4 / .951 / .064	32.2 / .947 / .072	32.0 / .950 / .067	36.4 / .977 / .023	27.6 / .918 / .113
P. HiCoM	2	31.0 / .940 / .080	31.6 / .948 / .066	31.9 / .951 / .065	32.8 / .951 / .068	32.8 / .953 / .065	36.7 / **.978** / .021	**28.0** / .919 / .113
Ours	2	**32.6** / **.950** / **.067**	**32.2** / **.953** / **.061**	**32.7** / **.956** / **.059**	**32.9** / **.953** / **.065**	**33.0** / **.954** / **.062**	**36.8** / .977 / **.021**	27.9 / .919 / .112
P. ST-4DGS	4	29.7 / .932 / .089	30.2 / .943 / .069	31.0 / .949 / .066	31.9 / .946 / .074	31.9 / .949 / .068	28.4 / .778 / .172	27.5 / .919 / **.111**
P. 4DGS	4	30.0 / .932 / .082	30.1 / .944 / .068	30.9 / .950 / .066	32.6 / .950 / .068	32.0 / .949 / .068	34.8 / .972 / .028	27.4 / .917 / .114
P. Dy.3DGS	4	29.1 / .927 / .096	30.8 / .946 / .068	30.8 / .946 / .071	31.7 / .945 / .075	31.5 / .948 / .070	35.8 / .975 / .026	27.6 / .918 / .114
P. HiCoM	4	30.4 / .936 / .088	31.4 / .946 / .069	31.6 / .949 / .068	32.6 / .950 / .070	32.6 / .952 / .066	36.4 / .977 / .022	**28.0** / .919 / .114
Ours	4	**32.4** / **.948** / **.070**	**32.4** / **.953** / **.062**	**32.6** / **.955** / **.060**	**32.8** / **.952** / **.066**	**32.9** / **.954** / **.063**	**36.7** / **.977** / **.021**	27.9 / **.919** / .112
P. ST-4DGS	8	29.1 / .924 / .099	30.0 / .941 / .073	30.6 / .947 / .069	30.1 / .883 / .108	31.6 / .947 / .071	33.7 / .968 / .033	27.4 / .916 / .113
P. 4DGS	8	29.6 / .928 / .091	29.7 / .937 / .076	30.2 / .940 / .071	32.4 / .949 / .069	31.6 / .947 / .070	33.8 / .966 / .035	27.3 / .916 / .115
P. Dy.3DGS	8	27.7 / .915 / .117	30.1 / .942 / .074	29.7 / .938 / .082	31.2 / .943 / .079	30.8 / .945 / .073	35.0 / .971 / .032	27.5 / .917 / .115
P. HiCoM	8	29.6 / .928 / .102	31.0 / .943 / .074	31.3 / .945 / .075	32.2 / .948 / .075	32.1 / .950 / .069	36.0 / .975 / .026	**28.0** / .918 / .115
Ours	8	**31.9** / **.944** / **.076**	**31.9** / **.951** / **.064**	**32.5** / **.954** / **.062**	**32.7** / **.952** / **.067**	**32.7** / **.953** / **.064**	**36.5** / **.976** / **.022**	27.9 / **.919** / **.112**

Visual quality in short clips: Table 1 presents quantitative results from short-clip experiments under different GPU parallelism settings. In most cases, our method outperforms baselines in visual quality on single GPUs and maintains higher quality as parallelism increases. This demonstrates robustness of TrackerSplat across different parallelism settings. In scenes with slow motion (e.g., "coffee martini"), baselines experience minor quality drops (around 0.1 PSNR), and achieve performance comparable to ours. Figure 6 provides visual comparisons, illustrating that baselines often suffer from fading or drifting artifacts in fast-moving regions, while our method better preserves visual fidelity, even in highly dynamic scenes. This demonstrates that explicit motion compensation prior to training effectively mitigates the impact of large displacements, maintaining high quality across different levels of parallelism.

Visual quality in long videos: Figure 5 shows quantitative results over long sequences. Our method achieves higher and more stable visual quality compared to baselines in most cases. Sample rendered videos are provided in the supplementary material.

5.5 Ablation Study Results

Figure 5 compares TrackerSplat with and without motion regularization. Motion regularization generally improves robustness but can slightly reduce quality in certain cases (e.g., "coffee martini"). Profiling reveals that regularization effectively corrects outliers but may also shift already well-positioned Gaussians, slightly degrading quality. For short, robustness of our method against point tracking error stems from several design choices: 1) Point tracker provides accurate pixel trajectories in most cases. 2) PWI-LS (Sec. 4.3) mitigates individual pixel tracking errors by computing transparency-weighted averages. 3) Multi-view triangulation (Sec. 4.4) corrects trajectory errors by averaging results from multiple viewpoints. 4) Median filtering (Sec.4.5.1) and propagation (Sec.4.5.2) handle severe trajectory errors by replacing incorrect trajectories with neighboring Gaussians. 5) The refinement stage (Sec. 4.6) further optimizes Gaussian positions, enhancing final accuracy.

5.6 Comparison of Parallel Performance

Table 2 compares parallel performance across different GPU settings. TrackerSplat achieves higher throughput in most settings. Our parallel framework can also improve the throughput of existing methods in multi-GPU environments. Baselines are generally slower due to the computational demands of their regularization terms or deformation fields. In contrast, although includes additional tracking stages, our method benefits from accurately aligned Gaussians before training, eliminating the need for complex regularization or deformation fields, and thus improving reconstruction throughput. Dynamic3DGS dataset contains more views (27) but at a lower resolution (only 640×360). In this dataset, the increased number of views introduces additional overhead in the point tracking stage. Furthermore, since all methods perform the same number of training/refinement iterations (1,000), the overhead from tracking outweighs the performance gains obtained from simplified training. As a result, our method shows comparatively lower throughput than baselines on this dataset.

Table 2. Average throughput comparison (seconds per frame) between TrackerSplat and baselines under different GPU parallelism settings (1, 2, 4, and 8 GPUs). Our method achieves the highest throughput in most cases. We also separately report the runtime of the tracking stages (point tracking and motion compensation) and the refinement stage.

Method	GPUs	Dy.3DGS	Meet.Room	st-nerf	N3DV
Parallel ST-4DGS	1	217.8	344.7	361.9	464.2
Parallel 4DGS	1	52.3	84.9	109.9	168.0
Parallel Dyn.3DGS	1	31.7	42.1	72.1	123.3
Parallel HiCoM	1	13.1	26.9	54.1	119.0
Ours (total)	1	**13.0**	**20.1**	**51.3**	**109.3**
Ours (track+refine)	1	5.3+7.6	3.3+16.9	6.3+45.1	10.0+99.3
Parallel ST-4DGS	2	108.9	172.3	181.0	232.1
Parallel 4DGS	2	26.1	42.4	54.9	84.0
Parallel Dyn.3DGS	2	15.8	21.1	36.1	61.6
Parallel HiCoM	2	**6.6**	13.4	27.0	59.5
Ours (total)	2	7.7	**10.7**	**26.6**	**55.5**
Ours (track+refine)	2	3.9+3.8	2.3+8.4	4.1+22.5	5.9+49.6
Parallel ST-4DGS	4	54.4	86.2	90.5	116.0
Parallel 4DGS	4	13.1	21.2	27.5	42.0
Parallel Dyn.3DGS	4	7.9	10.5	18.0	30.8
Parallel HiCoM	4	**3.3**	6.7	14.8	29.7
Ours (total)	4	5.2	**6.0**	**13.5**	**28.1**
Ours (track+refine)	4	3.3+1.9	1.7+4.2	2.2+11.3	3.3+24.8
Parallel ST-4DGS	8	27.2	43.1	45.2	58.0
Parallel 4DGS	8	6.5	10.6	13.7	21.0
Parallel Dyn.3DGS	8	4.0	5.3	9.0	15.4
Parallel HiCoM	8	**1.6**	3.4	7.4	14.9
Ours (total)	8	3.4	**3.4**	**7.1**	**14.7**
Ours (track+refine)	8	2.4+1.0	1.3+2.1	1.5+5.6	2.3+12.4

6 Limitations and Future Work

Small or thin objects. Point trackers often miss subtle motions in small or thin objects (e.g., human hands or fingers shown in Figure 6), leading to blurred or missing details.

Jittering artifacts. We observe temporal jitter (see the supplementary video) from two sources: 1) tracking errors, especially in low-texture regions where correspondence is ambiguous; although Motion Regularization (Sec.4.5) and Refinement (Sec.4.6) mitigate these issues, they do not eliminate them; and 2) spatiotemporal inconsistencies of the 3DGS representation under noisy video data [Yun et al. 2025].

Accumulated errors. While the refinement step can correct some errors from earlier frames, it is not sufficiently robust. As a result, inaccuracies may accumulate over time, and errors in the first frame can propagate to subsequent frames, especially in challenging cases with limited training views or complex scenes.

Occlusions. Typical point tracking methods establish trajectories by matching pixels in subsequent frames with those in the first input frame. Therefore, if an object is fully occluded in the first frame and becomes visible later in a clip, point tracker may fail to estimate its trajectory correctly, causing incomplete or missing reconstructions.

Potential Solutions. Recent studies [Duan et al. 2024; Sun et al. 2024] propose techniques that dynamically remove faded Gaussians and add new ones in regions with high gradients, potentially addressing these limitations. As these methods do not inherently maintain consistent Gaussian trajectories across frames, integrating such techniques with TrackerSplat would require additional mechanisms to match newly added Gaussians to existing trajectories. Moreover, error decomposition for 3DGS [Yun et al. 2025] directly targets spatiotemporal inconsistency and could be incorporated into our refinement stage. Exploring this integration represents a promising direction for future research.

7 Conclusion

We presented TrackerSplat, a robust and scalable approach for dynamic scene reconstruction using 3DGS. TrackerSplat integrates off-the-shelf point tracker to extract pixel trajectories, and triangulates them across views to update Gaussians before refinement. This design enables TrackerSplat to effectively manage large inter-frame motions, substantially improving reconstruction throughput in multi-GPU environments while maintaining high visual quality. Evaluation results on real-world dynamic scenes demonstrate the effectiveness and scalability of TrackerSplat, paving the way toward real-time dynamic scene reconstruction.

Acknowledgments

This research is supported by an NSERC Discovery Grant and a MITACS Accelerate Cluster Grant.

References

Jad Abou-Chakra, Krishan Rana, Feras Dayoub, and Niko Suenderhauf. 2024. Physically Embodied Gaussian Splatting: A Visually Learnt and Physically Grounded 3D Representation for Robotics. In *8th Annual Conference on Robot Learning*.

Carl Doersch, Yi Yang, Mel Vecerik, Dilara Gokay, Ankush Gupta, Yusuf Aytar, Joao Carreira, and Andrew Zisserman. 2023. TAPIR: Tracking Any Point with Per-Frame Initialization and Temporal Refinement. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 10061–10072.

Yuanxing Duan, Fangyin Wei, Qiyu Dai, Yuhang He, Wenzheng Chen, and Baoquan Chen. 2024. 4D-Rotor Gaussian Splatting: Towards Efficient Novel View Synthesis for Dynamic Scenes. In *ACM SIGGRAPH 2024 Conference Papers (SIGGRAPH '24)*. 1–11.

Zhiwen Fan, Wenyan Cong, Kairun Wen, Kevin Wang, Jian Zhang, Xinghao Ding, Danfei Xu, Boris Ivanovic, Marco Pavone, Georgios Pavlakos, Zhangyang Wang, and Yue Wang. 2024. InstantSplat: Unbounded Sparse-view Pose-free Gaussian Splatting in 40 Seconds. doi:10.48550/ARXIV.2403.20309

Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, Chenxi Wang, Junbo Wang, Haoyi Zhu, and Cewu Lu. 2024. RH20T: A Comprehensive Robotic Dataset for Learning Diverse Skills in One-Shot. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 653–660.

Qiankun Gao, Jiarui Meng, Chengxiang Wen, Jie Chen, and Jian Zhang. 2024. HiCoM: Hierarchical Coherent Motion for Dynamic Streamable Scenes with 3D Gaussian Splatting. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. 2019. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 201.

Nikita Karaev, Iurii Makarov, Jianyuan Wang, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. 2024. CoTracker3: Simpler and Better Point Tracking by Pseudo-Labelling Real Videos. doi:10.48550/ARXIV.2410.11831

Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. 2025. CoTracker: It Is Better to Track Together. In *Computer Vision – ECCV 2024*, Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (Eds.). 18–35.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (2023), 139:1–139:14.

Guillaume Le Moing, Jean Ponce, and Cordelia Schmid. 2024. Dense Optical Tracking: Connecting the Dots. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 19187–19197.

Jiahui Lei, Yijia Weng, Adam W. Harley, Leonidas Guibas, and Kostas Daniilidis. 2025. MoSca: Dynamic Gaussian Fusion from Casual Videos via 4D Motion Scaffolds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Deqi Li, Shi-Sheng Huang, Zhiyuan Lu, Xinran Duan, and Hua Huang. 2024b. ST-4DGS: Spatial-Temporally Consistent 4D Gaussian Splatting for Efficient Dynamic Scene Rendering. In *ACM SIGGRAPH 2024 Conference Papers (SIGGRAPH '24)*. 1–11.

Lingzhi Li, Zhen Shen, Zhongshu Wang, Li Shen, and Ping Tan. 2022a. Streaming Radiance Fields for 3D Video Synthesis. *Advances in Neural Information Processing Systems* 35 (2022), 13485–13498.

Tianye Li, Mira Slavcheva, Michael Zollhöfer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, and Zhaoyang Lv. 2022b. Neural 3D Video Synthesis From Multi-View Video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5521–5531.

Zhan Li, Zhang Chen, Zhong Li, and Yi Xu. 2024a. Spacetime Gaussian Feature Splatting for Real-Time Dynamic View Synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8508–8520.

Youtian Lin, Zuozhuo Dai, Siyu Zhu, and Yao Yao. 2024. Gaussian-Flow: 4D Reconstruction with Dynamic 3D Gaussian Particle. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 21136–21145.

Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. 2024. Dynamic 3D Gaussians: Tracking by Persistent Dynamic View Synthesis. In *3DV*.

Jenny Seidenschwarz, Qunjie Zhou, Bardienus Duisterhof, Deva Ramanan, and Laura Leal-Taixé. 2025. DynOMo: Online Point Tracking by Dynamic Online Monocular Gaussian Reconstruction. arXiv:2409.02104 [cs] doi:10.48550/arXiv.2409.02104

Colton Stearns, Adam Harley, Mikaela Uy, Florian Dubost, Federico Tombari, Gordon Wetzstein, and Leonidas Guibas. 2024. Dynamic Gaussian Marbles for Novel View Synthesis of Casual Monocular Videos. In *SIGGRAPH Asia 2024 Conference Papers*.

Jiakai Sun, Han Jiao, Guangyuan Li, Zhanjie Zhang, Lei Zhao, and Wei Xing. 2024. 3DGStream: On-the-Fly Training of 3D Gaussians for Efficient Streaming of Photo-Realistic Free-Viewpoint Videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 20675–20685.

Qianqian Wang, Yen-Yu Chang, Ruojin Cai, Zhengqi Li, Bharath Hariharan, Aleksander Holynski, and Noah Snavely. 2023. Tracking Everything Everywhere All at Once. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 19795–19806.

Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 2024. 4D Gaussian Splatting for Real-Time Dynamic Scene Rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 20310–20320.

Zhen Xu, Yinghao Xu, Zhiyuan Yu, Sida Peng, Jiaming Sun, Hujun Bao, and Xiaowei Zhou. 2024. Representing Long Volumetric Video with Temporal Gaussian Hierarchy. *ACM Trans. Graph.* 43, 6 (2024), 171:1–171:18.

Youngsik Yun, Jeongmin Bae, Hyunseung Son, Seoha Kim, Hahyun Lee, Gun Bang, and Youngjung Uh. 2025. Compensating Spatiotemporally Inconsistent Observations for Online Dynamic 3D Gaussian Splatting. In *ACM SIGGRAPH 2025 Conference Papers (Siggraph '25)*.

Jiakai Zhang, Xinhang Liu, Xinyi Ye, Fuqiang Zhao, Yanshun Zhang, Minye Wu, Yingliang Zhang, Lan Xu, and Jingyi Yu. 2021. Editable Free-Viewpoint Video Using a Layered Neural Representation. *ACM Transactions on Graphics* 40, 4 (2021), 149:1–149:18.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*.

Xiaoyu Zhou, Zhiwei Lin, Xiaojun Shan, Yongtao Wang, Deqing Sun, and Ming-Hsuan Yang. 2024. DrivingGaussian: Composite Gaussian Splatting for Surrounding Dynamic Autonomous Driving Scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 21634–21643.

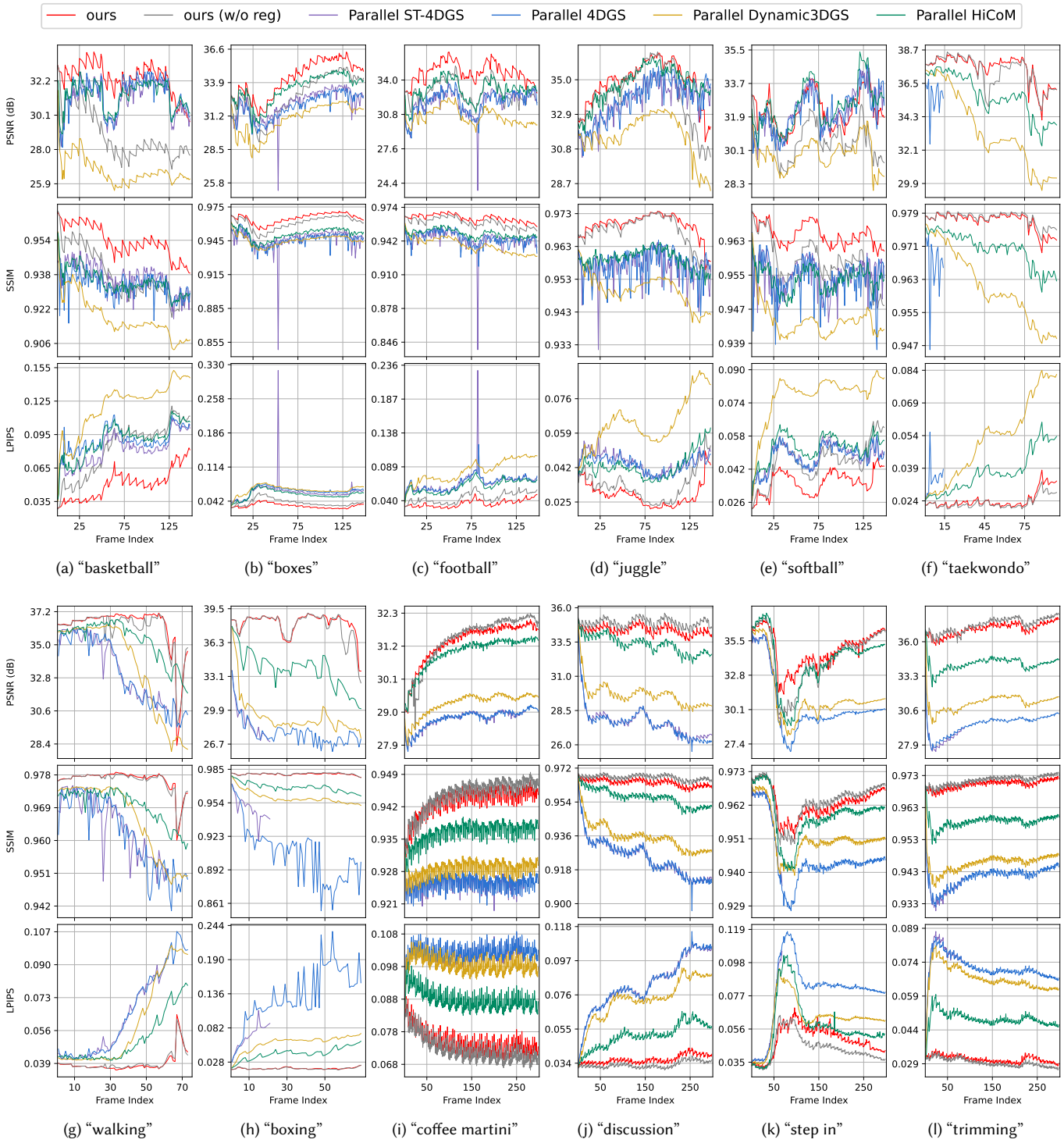


Fig. 5. Average visual quality (*PSNR* ↑ / *SSIM* ↑ / *LPIPs* ↓) over long-video sequences using our parallel pipeline with 8 GPUs (long-video experiments). Our method achieves higher and more stable visual quality than baselines in most cases, demonstrating its robustness. Lines ending prematurely for 4DGS and ST-4DGS indicate training failures due to GPU memory overflow (exceeding the 40GB limit of the A100 GPU) or numerical instabilities (NaN gradients). Corresponding rendered videos are provided in the supplementary material.

(a) Ground Truth (b) Ours (c) Parallel HiCoM (d) Parallel Dynamic 3DGS (e) Parallel 4DGS

Fig. 6. Qualitative comparison of rendered results from the final frame of representative 9-frame clips processed in parallel using 8 GPUs (short-clip experiments). Our method generates fewer artifacts and better preserves visual details compared to baselines, particularly in highly dynamic regions.