ReFlow: Self-correction Motion Learning for Dynamic Scene Reconstruction

Supplementary Material

This appendix complements the main paper by providing detailed implementation and optimization settings, as well as additional experimental results. We also provide an project page for more visualization.

A. Implementation Details

A.1. Complete Canonical Space Construction

We construct the complete canonical space through a two-stage alignment process, leveraging a geometric foundation model. We employ the method of [65] as our base geometric foundation model, denoted as Φ, due to its fine-tuning on dynamic scenes, which enables robust handling of complex scenarios with moving objects. For any input frame pair (I_i, I_j), this model computes 3D point maps and their confidence:

$$(\mathbf{X}_i^i, \mathbf{X}_i^j, \mathbf{C}_i^i, \mathbf{C}_i^j) = \Phi(I_i, I_j) \tag{10}$$

where $\mathbf{X}_i^i \in \mathbb{R}^{H\times W\times 3}$ is the 3D point map of frame i in i's coordinate system, $\mathbf{X}_i^j \in \mathbb{R}^{H\times W\times 3}$ is the 3D point map of frame j in i's coordinate system, and $\mathbf{C}_i^i, \mathbf{C}_i^j \in \mathbb{R}^{H\times W}$ are their respective confidence maps. For the input video, we partition it into uniform clips of 25-30 frames, striking an optimal balance between sufficient temporal context and computational efficiency.

The overall alignment process minimizes a composite loss function $\mathcal{L}_{\text{align}}$. This loss integrates three essential components: (1) reprojection error minimization ensuring geometric consistency across frames, (2) camera trajectory smoothness regularization preventing discontinuous camera paths, and (3) dynamic constraints specifically designed to accommodate dynamic scene elements. The specific formulation of these components can be referred to in [65]. Our alignment proceeds in two stages:

1. Coarse Keyframe Alignment. We first establish a globally consistent structure by aligning $K \ll N$ keyframes selected from the video. For each selected keyframe pair (I_a, I_b), we compute their relative geometry using Equation 10. With these pairwise estimations, we construct a connectivity graph $G_K(V_K, E_K)$ where vertices V_K represent keyframes and each edge $e \in E_K$ connects a keyframe pair. This reduced graph enables efficient global optimization of keyframe camera poses K_{pose}, intrinsics K_{intr}, and depth maps K_{depth}:

$$\min_{K_{\text{pose}}, K_{\text{intr}}, K_{\text{depth}}} \sum_{(a,b)\in E_K} \mathcal{L}_{\text{align}}(\mathbf{X}_a^a, \mathbf{X}_a^b, \mathbf{C}_a^a, \mathbf{C}_a^b) \tag{11}$$

2. Fine-grained Intra-clip Alignment. With keyframes aligned, we proceed to refine the geometry of intermediate frames within each clip S_k. For temporally close frames (I_i, I_j) within clip S_k (where $|i-j| \le \delta$, with δ being the clip length or a predefined window), we compute their relative geometry using Equation 10. We initialize the parameters for these frames using the results from the coarse keyframe alignment stage (K_{params}) and then optimize the poses S_k^{pose}, intrinsics S_k^{intr}, and depth maps S_k^{depth} for all frames within the clip:

$$\min_{\{S_k^{\text{pose}}, S_k^{\text{intr}}, S_k^{\text{depth}}\}} \sum_{\substack{i,j\in S_k \\ |i-j|\le\delta}} \mathcal{L}_{\text{align}}(\mathbf{X}_i^i, \mathbf{X}_i^j, \mathbf{C}_i^i, \mathbf{C}_i^j; K_{\text{params}}) \tag{12}$$

The loss $\mathcal{L}_{\text{align}}$ in this stage enforces local geometric consistency between neighboring frames. The term K_{params} ensures global consistency by anchoring the optimization to the already aligned keyframes.

In our practical implementations, we observed that the approach in [65], even within the two-stage framework described above, can occasionally exhibit instability in complex scenes when applied without proper initialization. To address this issue, particularly for the Coarse Keyframe Alignment stage (Equation 11), we enhance the global graph optimization by incorporating two critical priors: camera parameters estimated by COLMAP [50] and semantic motion masks derived from [49]. Notably, we maintain the COLMAP camera parameters fixed throughout the coarse optimization process, providing a stable reference frame for reconstruction. These initialization priors offer a practical aid in the reconstruction process, helping to guide the model toward more stable and consistent results.

A.2. Separation-Based Dynamic Scene Modeling

Here we provide a detailed mathematical formulation of our disentangled representation approach for static and dynamic scene components. Our method systematically separates scene elements based on their motion characteristics, with specialized representation schemes for each component.

For static scene components, we adapt a tri-plane-based 3D Gaussian representation using three orthogonal spatial feature planes $F_s = \{F^{xy}, F^{xz}, F^{yz}\}$, where each feature plane $F \in \mathbb{R}^{H\times W\times C}$. Given a 3D point (x, y, z), we query the Gaussian parameters by bilinearly interpolating each plane at its respective coordinate, concatenating the resulting features, and passing them through a static decoder D_s:

$$\begin{aligned} G_s(x,y,z) = D_s(&\text{Interp}(F^{xy}, x, y), \\ &\text{Interp}(F^{xz}, x, z), \\ &\text{Interp}(F^{yz}, y, z)) \end{aligned} \tag{13}$$

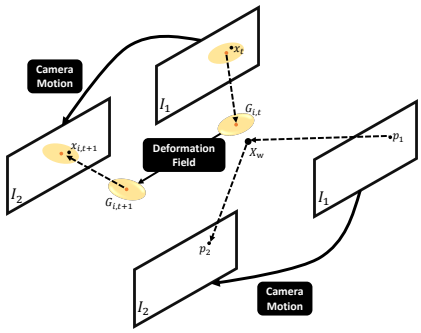

Figure 8. Illustration of the computation of Full Flow (top-left) and Camera Flow (bottom-right).

Here, $G_s = \{\mu_s, s_s, q_s, \sigma_s, c_s\}$ denotes the output Gaussian attributes, including position $\mu_s \in \mathbb{R}^3$, scale $s_s \in \mathbb{R}^3_+$, orientation quaternion $q_s \in \mathbb{S}^3$, opacity $\sigma_s \in [0, 1]$, and view-dependent appearance coefficients $c_s \in \mathbb{R}^{D_c}$. These parameters remain time-invariant and represent the static part of the scene.

For dynamic scene components that change over time, we augment the static representation with temporal feature planes $F_t = \{F^{xt}, F^{yt}, F^{zt}\}$ as described in Section 3.2.2, where each $F^{it} \in \mathbb{R}^{D\times T\times C}$ encodes the interaction between spatial dimension $i \in \{x, y, z\}$ and time t. For a 4D query (x, y, z, t), we extract temporal features via interpolation along each spatial-time plane and concatenate them with the spatial features to form a comprehensive spatio-temporal feature:

$$\hat{F}_d(x, y, z, t) = \text{Concat}_{F_{ij}\in\mathcal{F}_d}\text{Interp}(F_{ij}, i, j), \tag{14}$$

where

$$\mathcal{F}_d = \{F^{xy}, F^{xz}, F^{yz}, F^{xt}, F^{yt}, F^{zt}\}.$$

The decoder D_d then decodes this feature to produce the dynamic Gaussian parameters:

$$G_d(x, y, z, t) = D_d(\hat{F}_d(x, y, z, t)). \tag{15}$$

Our implementation of the dynamic Gaussians closely follows the approach in [55], using the same delta-based formulation for time-varying attributes. This ensures compatibility with established rendering pipelines while enabling our novel disentangled representation to effectively separate static and dynamic components.

A.3. Flow Render in Self-correction Flow Matching

In this section, we provide a more detailed exposition of the rendering processes for the *Full Flow* ($\mathbf{F}_{full}$) and *Camera Flow* ($\mathbf{F}_{cam}$), which are pivotal for our self-correction flow matching framework (see Fig. 8).

Full Flow ($\mathbf{F}_{full}$) As described in Section 3.3.1, the *Full Flow* ($\mathbf{F}_{full}$) represents the overall pixel displacement caused by both object motion and camera movement, corresponding to changes across the entire image. Given the scene states G_{t_1} and G_{t_2} (e.g., 3D Gaussians at different timestamps) and the corresponding camera parameters P_1 and P_2 include both camera intrinsics and extrinsics, the Full Flow is synthesized via a rendering function:

$$\mathbf{F}_{full} = \text{FlowRender}(G_{t_1}, G_{t_2}, P_1, P_2). \tag{16}$$

Following [19], we formulate the $\mathbf{F}_{full}$ as a 2D motion field that represents the combined effects of camera movement and object movement in the scene. This can be interpreted as an optical flow field computed from 2D video frames, but crucially, it is derived from our 4D scene representation, which consists of 3D Gaussians, feature planes, and a deformation field.

Specifically, for a pixel $\mathbf{x}_t$ in the 2D image plane, we first normalize its position relative to each contributing i-th 2D Gaussian at time t. This transforms $\mathbf{x}_t$ into a *canonical space*—a standard Gaussian distribution with zero mean and identity covariance, as referenced in [19]:

$$\hat{\mathbf{x}}_t = \mathbf{\Sigma}_{i,t}^{-1}(\mathbf{x}_t - \mu_{i,t}). \tag{17}$$

Here, $\mu_{i,t}$ and $\mathbf{\Sigma}_{i,t}$ represent the center and covariance of the i-th 2D Gaussian, respectively, which are obtained by projecting its corresponding 3D Gaussian onto the camera plane at timestamp t. This projection inherently incorporates camera infomation.

Next, we transform this normalized pixel position into the future, assuming it moves with the i-th Gaussian. This is achieved by transforming $\hat{\mathbf{x}}_t$ back into the coordinate system of the i-th Gaussian at time $t + 1$:

$$\mathbf{x}_{i,t+1} = \mathbf{\Sigma}_{i,t+1}\hat{\mathbf{x}}_t + \mu_{i,t+1}. \tag{18}$$

The parameters $\mu_{i,t+1}$ and $\mathbf{\Sigma}_{i,t+1}$ denote the center and covariance matrix of the i-th Gaussian at timestamp $t + 1$, which are derived by deforming the Gaussian from its state at time t via a learned deformation field. The individual flow contribution from the i-th Gaussian to the pixel $\mathbf{x}_t$ is then defined as the displacement between the original pixel position and its transformed future position:

$$\mathbf{F}_{full}^{i,t\to t+1} = \mathbf{x}_{i,t+1} - \mathbf{x}_t. \tag{19}$$

Finally, the total flow for the pixel $\mathbf{x}_t$ is derived by accumulating the contributions from all relevant Gaussians in an α-blending way:

$$\mathbf{F}_{full} = \sum_{i=1}^{K} w_i \mathbf{F}_{full}^{i,t\to t+1} \tag{20}$$

$$= \sum_{i=1}^{K} w_i(\mathbf{x}_{i,t+1} - \mathbf{x}_t), \tag{21}$$

where w_i is the weight of individual gaussians.

In practice, computing the Full Flow requires accounting for both the camera's egomotion and the motion of dynamic scene elements. To achieve this, we render the 2D Full Flow by projecting the 3D Gaussian deformations from time t to $t+1$ according to the corresponding camera viewpoints C_t and C_{t+1}. We adapt the implementation proposed by [19] for this process. This process ensures that the resulting 2D flow field accurately captures the total observed image-space displacement resulting from the combined motion of both the scene and the camera.

Camera Flow ($\mathbf{F}_{cam}$) As described in Section 3.3.1, the *Camera Flow* ($\mathbf{F}_{cam}$) isolates the 2D pixel displacements induced solely by the camera's egomotion, under the assumption that the 3D scene remains static. This flow component is particularly valuable as it provides a clean supervison signal in static scene regions, where observed pixel changes are attributable only to viewpoint shifts. The CamFlowRender is employed to compute this flow:

$$\mathbf{F}_{cam} = \text{CamFlowRender}(K_1, K_2, T_1, T_2, G_{t_1}), \quad (22)$$

where K_1 and K_2 are the camera intrinsic matrices at times t_1 and t_2 respectively, T_1 and T_2 are the corresponding extrinsic matrices (i.e., camera poses), and G_{t_1} denotes our 3D scene representation at time t_1 (e.g., a 3D Gaussian).

The computation of $\mathbf{F}_{cam}$ for a pixel p_1 in image I_1 (from camera C_1 at t_1) involves determining its new position p_2 in image I_2 (from camera C_2 at t_2), based on the premise that the underlying 3D scene point is stationary. Initially, p_1 (with homogeneous coordinates $\tilde{p}_1$) is back-projected to its 3D world coordinates $\mathbf{X}_w$. This step utilizes its depth $D_1(p_1)$ (rendered from G_{t_1}), the intrinsics K_1, and extrinsics T_1:

$$\mathbf{X}_w = T_1^{-1}(K_1^{-1}(D_1(p_1)\tilde{p}_1)). \quad (23)$$

Subsequently, this static 3D point $\mathbf{X}_w$ is projected into the image plane of camera C_2 (using K_2 and T_2) to yield the new 2D pixel location p_2:

$$p_2 = \text{proj}(K_2 T_2 \mathbf{X}_w). \quad (24)$$

The Camera Flow $\mathbf{F}_{cam}(p_1)$ is then defined as the displacement from p_1 to p_2:

$$\mathbf{F}_{cam}(p_1) = p_2 - p_1. \quad (25)$$

The entire operation can be expressed compactly as:

$$\mathbf{F}_{cam}(p_1) = \text{proj}\Big(K_2 T_2 \big(T_1^{-1}\big(K_1^{-1}\big(D_1(p_1)\tilde{p}_1\big)\big)\big)\Big) - p_1 \quad (26)$$

In essence, the camera flow precisely characterizes the pixel-wise motion in static regions that arises solely from camera movement. By leveraging $\mathbf{F}_{cam}$ as a matching constraint, we ensure that static scene elements maintain consistent geometric relationships across time and viewpoints.

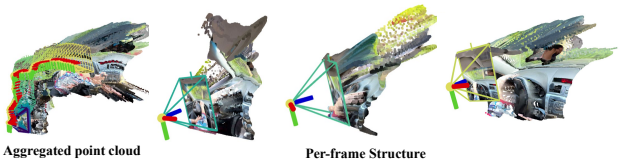

Figure 9. Failure case of the Utilized Geometry Foundation Model. **Top Left:** Aggregated point cloud exhibits noise and distortion from merging inconsistent frames. **Top Right:** Per-frame structure reconstructions show significant noise and fragmentation, demonstrating its difficulty in establishing reliable geometry.

A.4. Optimization Details

Optimization Strategy We adapt a two-stage optimization strategy following [55, 62]. In the warm-up stage, deformation fields for dynamic regions are frozen and camera flow matching is applied only to static regions, stabilizing the static scene structure. In the subsequent full optimization stage, both static and dynamic Gaussians are jointly optimized. We adjust the weights of flow constraints according to scene complexity: for scenes with simpler camera and object motion (like NVIDIA datasets), we use higher weights ($\lambda_{fullflow} = 5.0$, $\lambda_{camflow} = 0.3$) to enforce stronger constraints; for complex scenes with significant camera movement and object deformation (like HyperNeRF datasets) ($\lambda_{fullflow} = 1.0$, $\lambda_{camflow} = 0.1$), we use slightly lower weights to allow more flexibility. For cross-time rendering, the weight is set to 0.1 times the motion consistency weight. All optimization is performed pairwise on video frames. Optimizer and other settings follow [55].

Table 4. **Training Time and Peak GPU Memory Usage.** We compare our method with the baseline on the Nvidia Monocular dataset.

Method	**Balloon1**	**Balloon2**	**dynamicFace**	**Jumping**
Baseline [55]	35m 32s / 6.30G	37m 49s / 16.88G	58m 23s / 6.24G	37m 56s / 3.20G
Ours	44m 52s / 7.53G	46m 4s / 18.33G	68m 28s / 7.34G	46m 41s / 4.53G
Method	**Playground**	**Skating**	**Truck**	**Umbrella**
Baseline [55]	51m 19s / 6.45G	37m 19s / 2.32G	34m 56s / 4.12G	44m 45s / 5.96G
Ours	61m 9s / 7.50G	47m 30s / 3.60G	42m 58s / 5.61G	54m 15s / 7.11G

Table 5. **Rendering Speed and Model Storage Size.** We report the rendering frame rate (FPS) and the final model size (in MBs) for all scenes on the Nvidia Monocular dataset.

Method	**Balloon1**	**Balloon2**	**dynamicFace**	**Jumping**
Baseline [55]	20fps / 74M	21fps / 72M	15fps / 106M	26fps / 35M
Ours	18fps / 87M	17fps / 86M	13fps / 121M	23fps / 52M
Method	**Playground**	**Skating**	**Truck**	**Umbrella**
Baseline [55]	15fps / 93M	30fps / 33M	24fps / 45M	25fps / 54M
Ours	15fps / 105M	25fps / 47M	21fps / 65M	21fps / 72M

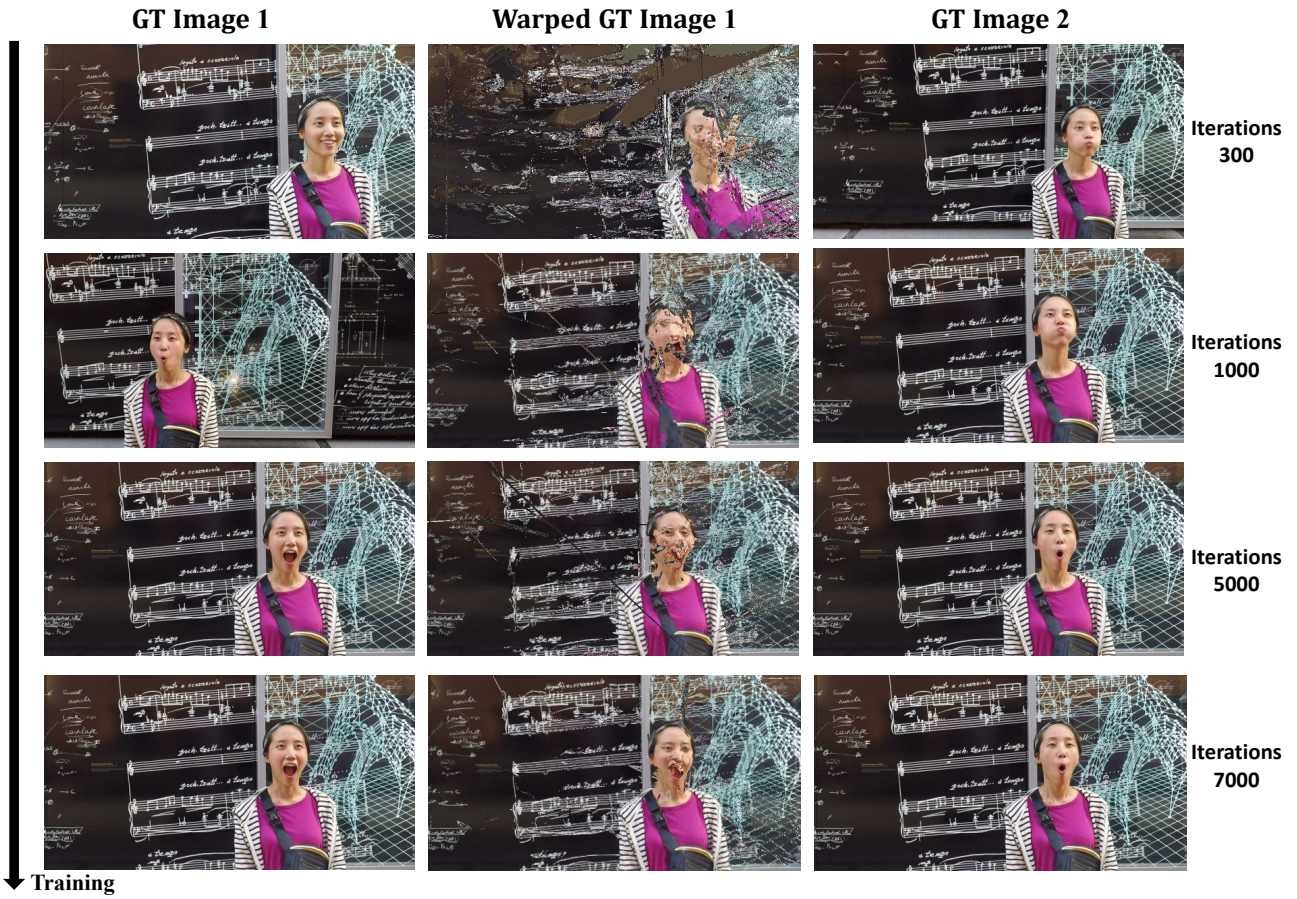

Figure 10. Visualization of our self-correction flow matching progress across training iterations, using the DynamicFace sequence from the Nvidia Monocular dataset [18]. Each row shows results at different training iterations: 300 (top row), 1000 (second row), 5000 (third row), and 7000 (bottom row). The columns present: **Left column (I_1):** Ground truth image at time t_1. **Middle column (I_1^{warped}):** Ground truth image from t_1 warped using our predicted flow field $\mathbf{F}$, demonstrating how content transforms to match the target frame. **Right column (I_2):** Ground truth image at time t_2, serving as the reference for evaluating warping accuracy. As training progresses, I_1^{warped} increasingly aligns with I_2, demonstrating that our self-correction flow matching effectively learns to model dynamic scene motion without requiring external motion guidances.

Compute Resources All experiments reported in this paper were conducted using a single NVIDIA RTX A6000 GPU with 48GB of memory. Memory is primarily consumed by the geometry foundation model used during the canonical space construction stage of our method. Actual memory usage is affected by the clip size used in our coarse-to-fine alignment strategy. Processing larger input clips consumes more memory because handling these larger windows directly dictates the memory overhead needed for the second-stage fine alignment.

Runtime Analysis We report the per-scene training time, peak GPU memory usage, rendering speed, and storage for our method in Table 4 and Table 5. Our approach introduces moderate increases in training time and memory due to additional flow rendering and canonical space construction. The core of our method is on exploring a self-correction motion learning mechanism during training without adding extra computational burden during inference. Thus, the rendering speed and inference time of our method are comparable to the baseline 4DGS, ensuring improved reconstruction quality without sacrificing rendering efficiency.

B. Dataset Details

For the HyperNeRF [46] and Nerfies [45] datasets, we adapt a resolution that is downsampled by a factor of 2 from the standard resolution, resulting in 536×960 pixels. For the NVIDIA [18] dataset, we implement the temporal interpolation setting as outlined in [29]. We sample one frame per timestamp sequentially from the multi-view video sequence, collecting a total of 192 monocular frames. To establish our test set, we systematically hold out every 8th frame from

training, which gives us 168 frames for training and reserves 24 frames for testing. This protocol challenges our model to reconstruct 3D motion between captured timepoints not observed during training.

C. Additional Results

Detailed Per-scene Results We visualize per-scene results for the NVIDIA dataset in Figure 11, and for the Nerfies-HyperNeRF dataset in Figure 12. The per-scene breakdown performance for Nerfies-HyperNeRF dataset is presented in Table 8 and Table 9. We also provide a visualization demonstrating our model's learned temporal dynamics in Figure 13.

Table 6. Quantitative comparison using PSNR(↑), SSIM(↑), and LPIPS(↓) metrics. Best results are in **bold**.

Method	Balloon1	Balloon2	Jumping	dynamicFace
Ours	**27.65 / 0.903 / 0.092**	**29.01 / 0.908 / 0.098**	24.85 / **0.855 / 0.151**	30.98 / 0.973 / 0.036
Ours (External Flow)	27.35 / 0.899 / 0.095	27.74 / 0.843 / 0.179	**24.92** / 0.853 / 0.162	**31.10 / 0.975 / 0.034**
Method	**Playground**	**Skating**	**Truck**	**Umbrella**
Ours	**25.58 / 0.889 / 0.086**	**30.17** / 0.946 / 0.076	**30.18 / 0.917 / 0.115**	27.21 / **0.831 / 0.170**
Ours (External Flow)	20.97 / 0.820 / 0.099	30.09 / **0.976 / 0.032**	29.30 / 0.905 / 0.133	**27.35 / 0.831** / 0.171

Comparison on External Flow Constraints. We present a quantitative comparison between our self-correction approach and a variant that relies on external optical flow supervision (Ours (External Flow)) in Table 6. The key difference lies in the source of motion supervision. Approaches using external optical flow treat these estimates as pseudo-ground truth, forcing the learned 3D motion to follow potentially inaccurate 2D cues. This indirect supervision can conflict with the primary photometric reconstruction objective, particularly when the external flow contains errors. A notable example is the Playground scene, which features dominant camera motion and small, fast-moving foreground objects. External flow estimators primarily capture global camera-induced motion and often fail to represent fine-scale object dynamics, providing inaccurate motion supervision that misguides the learned 3D motion and results in a performance drop compared to our self-correction method. Similar degradations are also observed in Balloon2 and Truck.

In contrast, our key insight is that rather than aligning with indirect and often unreliable external priors, a more robust and principled approach is to derive motion supervision directly from the video itself. By grounding the learning signal directly ensure the reconstructed motion explains the observed frame-to-frame appearance changes, the learned motion naturally captures the true dynamics present in the video, rather than being forced to match external motion patterns. This approach embodies a more principled learning strategy for dynamic scene reconstruction and delivers improved robustness across diverse scenarios.

D. Further Discussion

Limitation. While our method learns 3D motion in a simple and self-correction way, the overall performance remains constrained by the quality of the scene geometry. In scenes involving rapid object appearance/disappearance or topological changes, correspondence becomes unreliable, which fundamentally limits the geometry foundation model itself (see Figure 9) and thus affects our entire pipeline. Nevertheless, the core contribution of our work lies in a novel self-correction motion supervision mechanism. This formulation is inherently flexible and can be readily integrated into a wide range of geometry foundation models or reconstruction pipelines. As geometry foundation models such as Align3r [41] and MegaSAM [37] continue to advance, our framework can seamlessly benefit from improvements in these models for capturing more complex scenes.

Discussion on Paradigms. Recently, Feed-forward GS method and pretraining-based 4D reconstruction method have gained significant attention. To discuss these new paradigms alongside the per-scene optimization used in Reflow, We provide a brief comparison in Tab. 7. Pretraining-based methods like D4RT [64] achieves impressive results in 4D reconstruction, but only predict sparse point cloud and needs a large amount of annotated training data. Feed-forward GS pipelines like 4DGT [60] can be trained without geometry annotation, but still requires huge training resources and suffers from pixel-aligned GS representation. Per-scene optimization methods can be trained with limited resources, but are sensitive to initialization. As a potential improvement, D4RT results can serve as the initialization of our methods, which can be further optimized and enable high-quality 4D reconstruction and real-time rendering.

Table 7. Comparison of 4D Reconstruction Paradigms.

	Pretraining-based	Feed-forward GS	Per-scene Optimization
Methods	D4RT [64]	DGS-LRM [39], 4DGT [60]	Ours, 4DGS [55]
Representation	Point Cloud	3D Gaussian	3D Gaussian
Annotation	Video, Camera Pose, Point Cloud, Point Tracking	Video, Camera Pose, External Priors (Depth/Flow)	Video, Camera Pose
Robustness	Very High	High	Low
Resources	64 TPU	64 GPU	Single GPU
Training Time	2 days	15 days	30 mins per scene
Limitation	Sparse Rendering	Pixel-aligned GS	Sensitive to initialization

Broader Impacts. This work poses no significant negative societal risks. Our framework enables self-correction 3D reconstruction and motion modeling from monocular video, making dynamic scene digitization more accessible. Users can convert everyday video footage—such as those captured by smartphones—into explicit 4D assets. These assets can facilitate downstream applications like content editing, digital scene creation, virtual reality, and educational visualization. It focuses on general scenes and does not target personal identification or sensitive data collection.

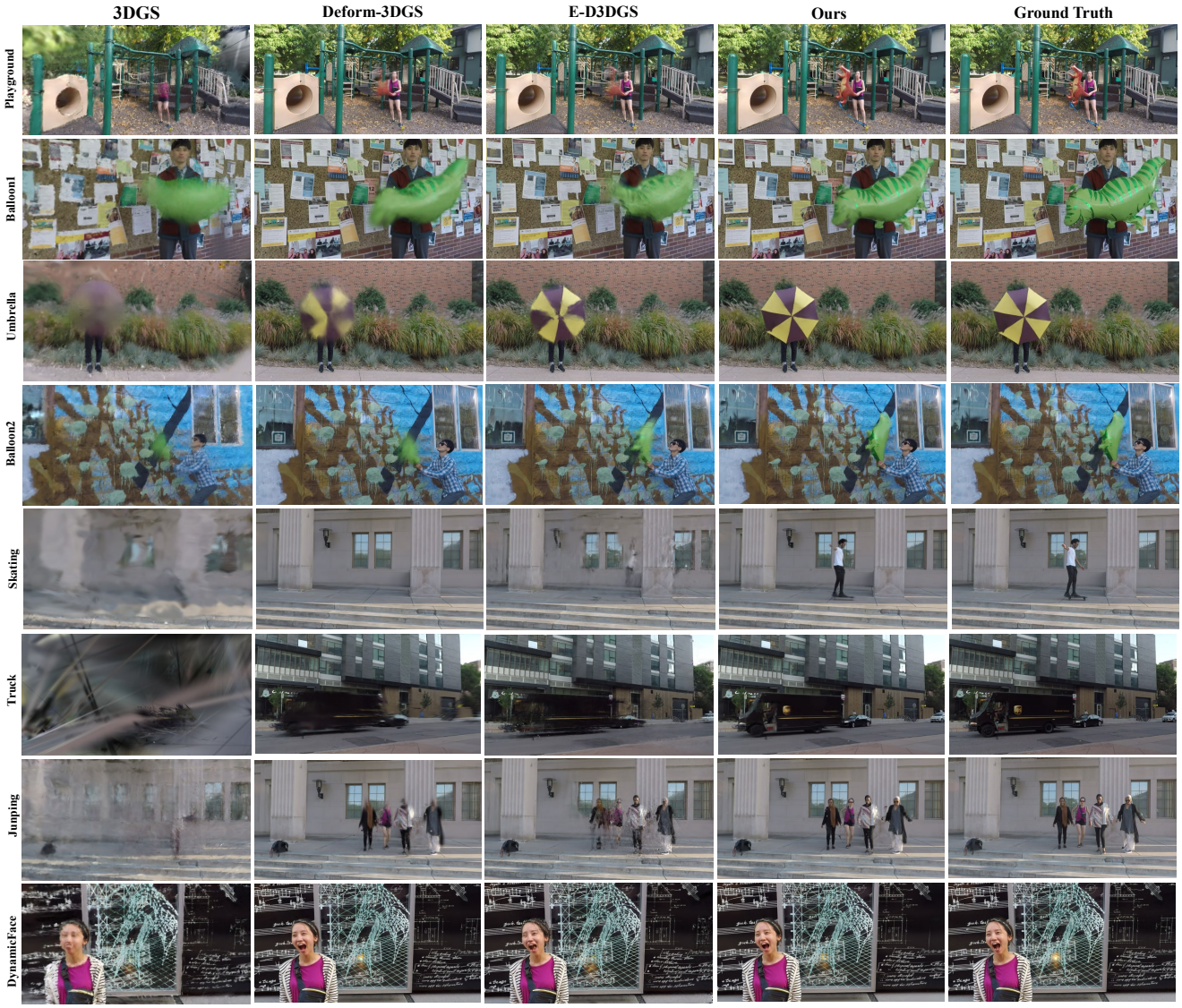

Figure 11. **Qualitative comparison on all scenes from the Nvidia Monocular dataset [18].**

E. Data and Code Availability

The datasets used in this study are publicly available: NVIDIA Monocular [17] (https://github.com/gaochen315/DynamicNeRF), Nerfies [45] (https://github.com/google/nerfies/releases/tag/0.1), and HyperNeRF [46] (https://github.com/google/hypernerf/releases/tag/v0.1), all under permissive licenses. Our baseline code [55] is at https://github.com/hustvl/4DGaussians.

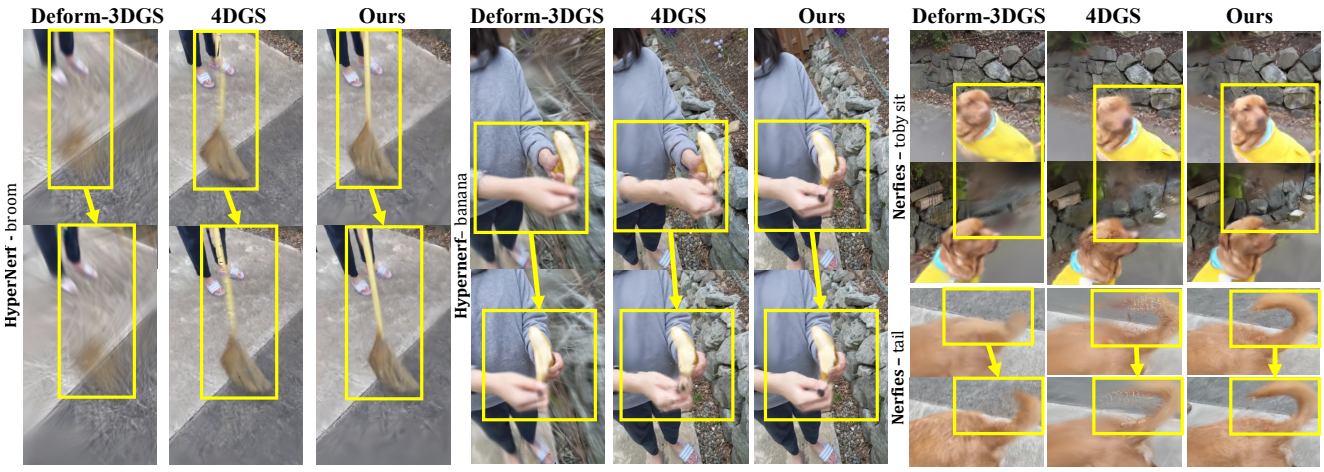

Figure 12. **More qualitative comparisons from the Nerfies-HyperNeRF dataset [45, 46].**

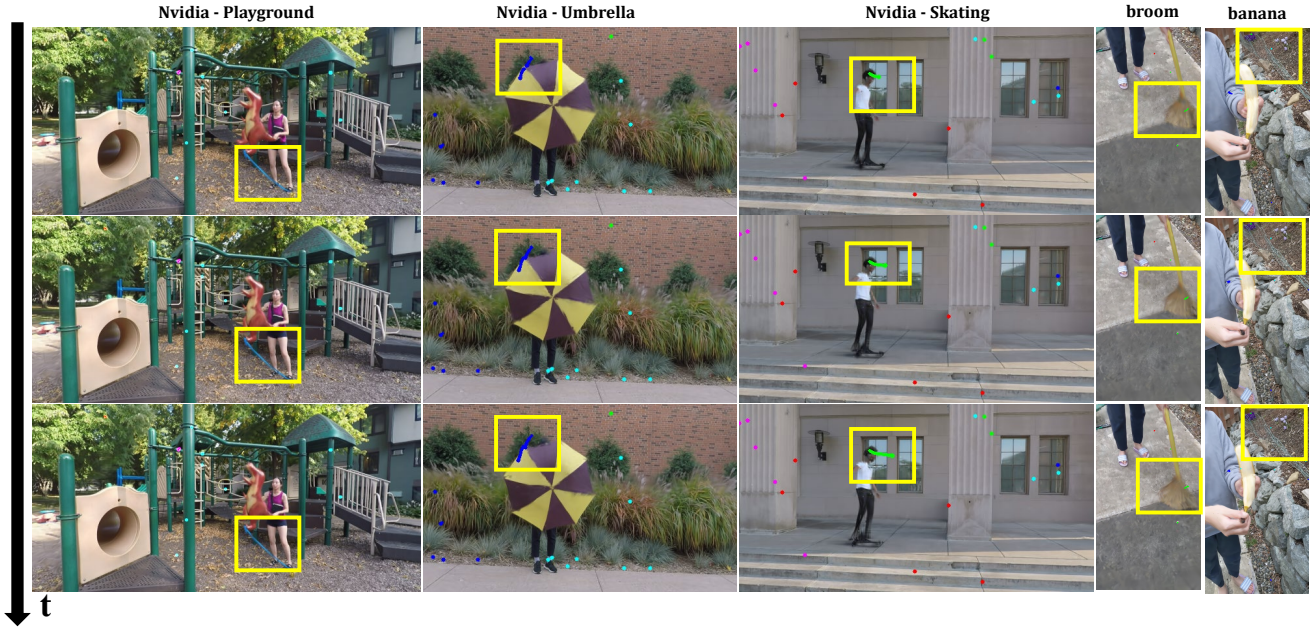

Figure 13. **Novel time synthesis results with trajectory visualization across different dynamic scenes.** Each column shows a different dataset: Nvidia-Playground, Nvidia-Umbrella, Nvidia-Skating, Hypernerf-broom, and Hypernerf-banana. The yellow boxes highlight dynamic/static regions with our tracked trajectories visualized using the Deform-GS approach. Rows represent different timesteps with fixed camera positions, demonstrating how our method correctly models temporal scene evolution. Note how static background elements remain perfectly stable across frames while dynamic components exhibit physically plausible motion paths. The Playground scene (leftmost column) particularly demonstrates our method's capability to preserve fine structures like blue ribbons during motion, which are typically challenging to reconstruct accurately.

Table 8. **Per-scene performance metrics on Nerfies-HyperNeRF [45, 46] dataset.** We highlight the best and second best results.

Method	Broom		Tail		Toby-sit	
	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑
T-NeRF [47]	20.17	0.257	22.11	0.385	18.53	0.330
NSFF [34]	20.46	0.247	21.72	0.388	18.65	0.329
Nerfies [45]	19.51	0.202	21.17	0.305	18.41	0.326
HyperNeRF [46]	19.23	0.197	21.13	0.301	18.33	0.324
Deformable-3DGS [62]	20.51	0.352	22.10	0.474	21.13	0.427
4DGS [55]	22.00	0.366	24.02	0.426	22.07	0.365
Ours	23.97	0.481	27.11	0.598	24.12	0.502
Method	**3DPrinter**		**Chicken**		**Peel-banana**	
	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑
T-NeRF [47]	18.60	0.591	21.11	0.764	22.07	0.721
NSFF [34]	16.26	0.426	20.72	0.619	18.62	0.530
Nerfies [45]	18.81	0.588	22.71	0.742	19.85	0.609
HyperNeRF [46]	18.73	0.586	23.88	0.753	21.08	0.641
Deformable-3DGS [62]	20.53	0.641	22.82	0.618	26.05	0.833
4DGS [55]	21.99	0.704	28.65	0.814	28.01	0.852
Ours	22.91	0.738	29.60	0.868	28.13	0.858

Table 9. **Per-scene performance metrics on HyperNeRF [46] dataset.** We highlight the best and second best results.

Method	3D Printer		Chicken		Broom		Banana	
	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑
T-NeRF [47]	18.60	0.591	24.41	0.764	20.17	0.257	22.07	0.721
NSFF [34]	16.26	0.426	20.72	0.619	20.46	0.247	18.62	0.530
Nerfies [45]	18.81	0.588	22.71	0.742	19.51	0.202	19.85	0.609
HyperNeRF [46]	18.73	0.583	23.88	0.753	19.23	0.197	21.08	0.641
Deformable-3DGS [62]	20.53	0.641	22.82	0.618	20.51	0.352	26.05	0.833
SC-GS [21]	18.79	0.613	21.85	0.616	18.66	0.269	25.49	0.806
MoDec-GS [29]	22.00	0.706	28.77	0.834	21.04	0.303	28.25	0.873
Ours	22.91	0.738	29.60	0.868	23.97	0.481	28.13	0.858