

# 4D-Fly: Fast 4D Reconstruction from a Single Monocular Video

## Supplementary Material

### A. Additional Implementation Details

**Details of the Learning Rate.** Our framework includes three optimization processes: the optimization of the Canonical Gaussian Map (Section 3.3), the optimization of foreground Gaussians (Section 3.4), and the optimization of background Gaussians (Section 3.4). An Adam optimizer is employed for all three processes. The specific learning rates for each of these processes are summarized in Table 4.

Table 4. Learning Rate Details.

Parameters	CGM	Foreground	Background
$\mu_i$	$2 \times 10^{-4}$	0	$2 \times 10^{-3}$
$\Delta\mu_i$	N/A	$2 \times 10^{-3}$	N/A
$\mathbf{c}_i$	$1 \times 10^{-2}$	0	$1 \times 10^{-2}$
$\Delta\mathbf{c}_i$	N/A	$1 \times 10^{-3}$	N/A
$\mathbf{S}_i$	$5 \times 10^{-3}$	0	$5 \times 10^{-3}$
$o_i$	$5 \times 10^{-2}$	0	$5 \times 10^{-2}$

**Details of the Loss Weights.** During the construction of the Canonical Gaussian Map, we set the loss weights as follows:  $\lambda_I = 1.0$ ,  $\lambda_D = 0.8$ ,  $\lambda_M = 0.8$ , and  $\lambda_{align} = 1.5$ . For the optimization of foreground Gaussians in Section 3.4, the weights are set to  $\lambda_{d,I} = 1.0$ ,  $\lambda_{d,D} = 0.8$ ,  $\lambda_{d,M} = 0.8$ ,  $\lambda_{reg,rig} = 1.5$ , and  $\lambda_{reg,col} = 1.5$ . For the optimization of background Gaussians in Section 3.4, the weights are set to  $\lambda_{s,I} = 1.0$ ,  $\lambda_{s,D} = 0.8$ , and  $\lambda_{s,M} = 0.8$ .

**Details of the KNN Algorithm.** Our framework integrates the KNN algorithm in two key components: the anchor-based Gaussian propagation algorithm and the computation of  $L_{reg,rig}$ . We use a KD-tree-based implementation of KNN provided by Open3D. Specifically, we first construct a KD-tree from the input point cloud, and then use `search_knn_vector_3d` method to retrieve the indices and squared distances of the nearest neighbors for the queried point. In the anchor-based Gaussian propagation algorithm, we set  $K = 5$ . For the computation of  $L_{reg,rig}$ ,  $K_1 = 20$  and  $K_2 = 10$  are used.

**Details of the Unreconstructed Areas Mask.** In our method, we employ an unreconstructed areas mask  $\hat{M}(\mathbf{p})$  [17] to identify regions where new Gaussians should be added. When computing  $\hat{M}(\mathbf{p})$ , the hyperparameters are set as follows:  $\lambda_1 = 0.5$  and  $\lambda_2 = 50 \times \text{Median}(D(\mathbf{p}))$ .

**Details of the view sampling.** For the experiments in Figures 6, 8, 9, 10 and Tables 1, 3, we follow the official benchmark setting to sample novel views (Fig. 4). For video visualization in *supp*, we set the camera to follow the wandering trajectory.

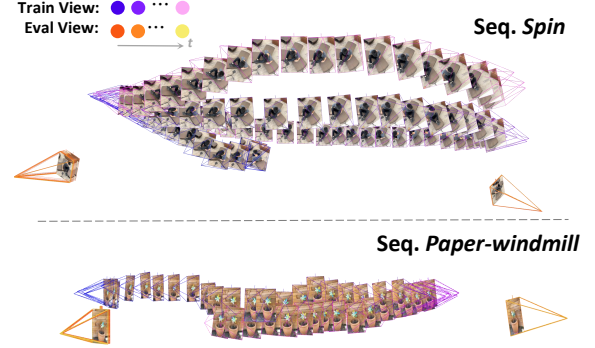


Figure 4. Train and eval view visualization on Dycheck dataset.

### B. Additional Method Details

**Illustration of Gaussian Propagation and Regularization.** Figure 5 presents an overview of our anchor-based Gaussian propagation algorithm and spatial-velocity-based regularization. Specifically, we calculate the offset, which is also defined as the initial velocity of a dynamic Gaussian for simplicity, as the weighted sum of the deformation of nearby anchor points between consecutive frames. This computed initial velocity is then incorporated as part of the distance metric when calculating the rigidity regularization term, ensuring consistency and stability across dynamic scenes.

**Detailed Algorithm.** In Algorithm 1, several key processes are abstracted into function calls due to space limitations. To enhance clarity and reproducibility, we provide a step-by-step implementation of each summarized function from Algorithm 1 in Algorithm 2, Algorithm 3, and Algorithm 4.

#### Algorithm 2 Anchor Based Gaussian Propagation

- 1: **Input:** 4D scene map constructed from observations between time steps 1 and  $t$ , denoted as  $\mathcal{G}^{1 \rightarrow t}$ , the depth Map at time step  $t + 1$ , denoted as  $\mathbf{D}_{t+1}$ , the 2D tracks from time step  $t$  to  $t + 1$ , denoted as  $\mathbf{U}_{t \rightarrow t+1}$ , camera extrinsic  $\mathbf{E}_t$ , and camera intrinsic  $\mathbf{K}_t$ .
- 2: **Output:** 4D scene map  $\mathcal{G}_{init}^{1 \rightarrow t+1}$ .
- 3: **for**  $\mathbf{p}_j$  in  $\text{Domain}(\mathbf{U}_{t \rightarrow t+1})$  **do**
- 4:    $\mathbf{x}_j \leftarrow \mathcal{F}(\mathbf{p}_j, \mathbf{D}_t(\mathbf{p}_j))$ , where  $\mathcal{F}$  is unproject function with respect to  $\mathbf{E}_t$  and  $\mathbf{K}_t$ .
- 5:    $\mathbf{x}'_j \leftarrow \mathcal{F}(\mathbf{U}_{t \rightarrow t+1}(\mathbf{p}_j), \mathbf{D}_{t+1}(\mathbf{U}_{t \rightarrow t+1}(\mathbf{p}_j)))$
- 6:    $\mathbf{a}_j = \mathbf{x}'_j - \mathbf{x}_j$
- 7: **end for**
- 8: **for**  $G_{d,i}^{s_i} \in \mathcal{G}_d^{1 \rightarrow t}$  **do**
- 9:    $\{\mathbf{x}_{ji}\}_{j=1}^K \leftarrow \text{KNN}(\mu_{i,t})$ , where  $\mu_{i,t} = \mu_i + \Delta\mu_{i,t}$  is the mean of  $G_{d,i}^{s_i}$  at timestep  $t$
- 10:    $\Delta\mu_{i,t+1}^{init} \leftarrow \Delta\mu_{i,t} + \sum_{j=1}^K \text{Softmax}(-\|\mathbf{x}_{ji} - \mu_i\|) \mathbf{a}_j$
- 11: **end for**

---

**Algorithm 3** Build Canonical Gaussian Map
 

---

```

1: Input: Image  $\mathbf{I}_{t+1}$ , depth  $\mathbf{D}_{t+1}$ , foreground mask  $\mathbf{M}_{t+1}$ , camera
   extrinsic  $\mathbf{E}_{t+1}$ , and camera intrinsic  $\mathbf{K}_{t+1}$ .
2: Output: Canonical Gaussian Map  $GCM(\mathbf{p})$  at timestep  $t + 1$ .
3: for  $\mathbf{p}_i$  in  $\text{Domain}(\mathbf{I})$  do
4:    $\mu_i \leftarrow \mathcal{F}(\mathbf{p}_i, \mathbf{D}_{t+1}(\mathbf{p}_i))$ , where  $\mathcal{F}$  is unproject function with re-
     spect to  $\mathbf{E}_{t+1}$  and  $\mathbf{K}_{t+1}$ 
5:    $\mathbf{c}_i \leftarrow \mathbf{I}_{t+1}(\mathbf{p}_i)$ 
6:    $s_i \leftarrow 2\mathbf{D}(\mathbf{p}_i)/(f_x + f_y)$ , where  $f_x$  and  $f_y$  are the camera lengths
     get by  $\mathbf{K}_{t+1}$ 
7:    $o_i \leftarrow \text{Sigmoid}(1)$ 
8:    $CGM(\mathbf{p}_i) \leftarrow \text{Initialize}(\mu_i, \mathbf{c}_i, s_i, o_i)$ 
9: end for
10: for  $i = 1, \dots, n_{CGM}$  do
11:   Optimize( $GCM(\mathbf{p})$ ,  $\mathbf{I}_{t+1}$ ,  $\mathbf{D}_{t+1}$ ,  $\mathbf{M}_{t+1}$ ,  $\mathbf{E}_{t+1}$ ,  $\mathbf{K}_{t+1}$ )
12: end for

```

---

**Algorithm 4** Foreground/Background Optimization Step
 

---

```

1: Input: 4D scene map  $\mathcal{G}^{1 \rightarrow t+1}$ , image  $\mathbf{I}_{t+1}$ , depth map  $\mathbf{D}_{t+1}$ ,
   foreground mask  $\mathbf{M}_{t+1}$ , camera extrinsic parameters  $\mathbf{E}_{t+1}$ , and
   camera intrinsic parameters  $\mathbf{K}_{t+1}$ . Note that  $t_i = t$  for the Fore-
   ground Optimization.
2: Output: Foreground-optimized or background-optimized 4D scene
   map ( $\mathcal{G}^{1 \rightarrow t+1}$ ) after the optimization step.
3:  $\hat{\mathbf{I}}(\mathbf{p}), \hat{\mathbf{M}}(\mathbf{p}), \hat{\mathbf{D}}(\mathbf{p}) \leftarrow \text{Render}(\mathcal{G}_{init}^{1 \rightarrow t+1}, t_i, \mathbf{E}_{t+1}, \mathbf{K}_{t+1})$ 
4:  $\mathcal{L} \leftarrow \lambda_I \|\hat{\mathbf{I}} - \mathbf{I}\|_1 + \lambda_M \|\hat{\mathbf{M}} - \mathbf{M}\|_1 + \lambda_D \|\hat{\mathbf{D}} - \mathbf{D}\|_1$ 
5: if Is Foreground Optimization Step then
6:    $\mathcal{L}_{reg,rig}, \mathcal{L}_{reg,col} \leftarrow 0$ 
7:   for  $G_i \in \mathcal{G}_d^{1 \rightarrow t+1}$  do
8:      $N'_i \leftarrow K1NN(G_{d,i})$ , where spatial distance is used as the dis-
       tance metric.
9:      $N_i \leftarrow K2NN(N'_i)$ , where velocity ( $\mathbf{v}_{i,t+1}^{init}$ ) is used as the dis-
       tance metric.
10:    for  $G_j \in N_i$  do
11:       $\Delta_{\mu,t} \leftarrow \mu_{j,t} - \mu_{i,t}$ 
12:       $\Delta_{\mu,t+1} \leftarrow \mu_{j,t+1} - \mu_{i,t+1}$ 
13:       $\Delta_{\mathbf{v},t+1}^{init} \leftarrow \mathbf{v}_{j,t+1}^{init} - \mathbf{v}_{i,t+1}^{init}$ 
14:       $w_{i,j} = \exp(-\lambda_{\mu} \|\Delta_{\mu,t}\|_2 - \lambda_v \|\Delta_{\mathbf{v},t+1}^{init}\|_2)$ 
15:       $\mathcal{L}_{reg,rig} \leftarrow \mathcal{L}_{reg,rig} + w_{i,j} \|\Delta_{\mu,t} - \Delta_{\mu,t+1}\|_2$ 
16:    end for
17:     $\mathcal{L}_{reg,col} \leftarrow \|\mathbf{c}_{i,t+1} - \mathbf{c}_{i,t}\|_2$ 
18:  end for
19:   $\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_{reg,rig}/k_2|\mathcal{G}_d| + \mathcal{L}_{reg,col}/|\mathcal{G}_d|$ 
20: end if
21: Backward_and_step( $\mathcal{L}, \mathcal{G}^{1 \rightarrow t+1}$ , optimizer)

```

---

## C. Additional Experiments and Analysis

### C.1. Comparison with NeRF-based Baselines

In Figure 8, we present a qualitative comparison of 4D-Fly with NeRF-based 4D reconstruction methods, including T-NeRF [11], Nerfies [31], and HyperNeRF [32]. We show results for the same sequence, viewpoint, and timestep as in Figure 3 for comparison. As shown, NeRF-based approaches are less effective at reconstructing high-frequency areas, often resulting in overly smooth transitions. Furthermore, our method outputs empty (white) values for regions not visible in the training views, whereas NeRF-based methods produce random, noisy values. Finally, our method

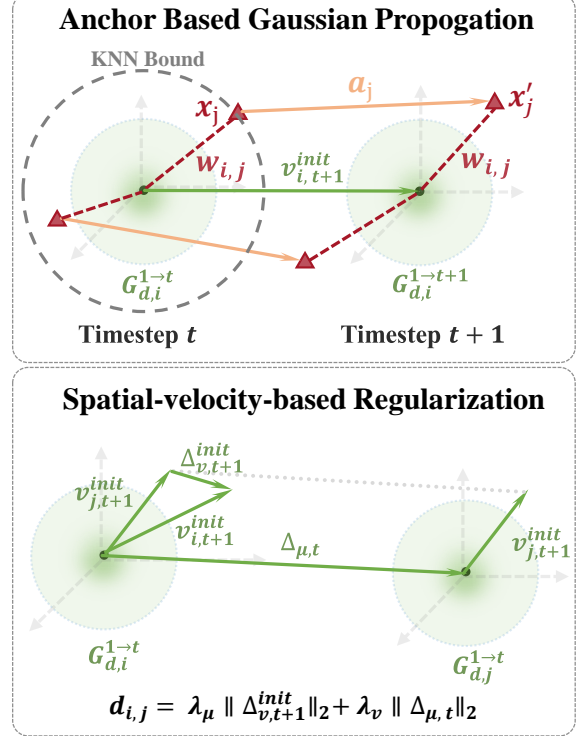


Figure 5. **Upper:** An illustration of anchor-based Gaussian propagation. For each dynamic Gaussian  $G_{d,i}^{1 \rightarrow t}$ , its motion is computed as a weighted sum of the offsets of its  $K$  nearest anchor points, where the weights are determined based on the spatial distances between  $G_{d,i}^{1 \rightarrow t}$  and each anchor point. We also define the motion of Gaussian as  $\mathbf{v}_{i,t+1}^{init}$ . **Lower:** We use a spatial (*i.e.*,  $\mu_i$  and  $\mu_j$ ) and velocity (*i.e.*,  $\mathbf{v}_{i,t+1}^{init}$  and  $\mathbf{v}_{j,t+1}^{init}$ ) weighted distance to identify the Gaussians that should maintain local rigidity relationships during fast 4D scene optimization.

significantly reduces both training and inference times compared to NeRF-based methods.

### C.2. Comparison on Dynamic Scenes Dataset.

In Figure 9, we present a qualitative comparison of 4D-Fly against 4D GS [46], Dynamic Gaussian Marbles [35], and Shape of Motion [44] on the Nvidia Dynamic Scene Dataset. Since the training sequences are captured with a stationary camera on the dataset, 4D GS [46] suffers from limited multi-view information and often cannot reconstruct clear foregrounds. Compared to Dynamic Gaussian Marbles [35] and Shape of Motion [44], 4D-Fly produces qualitatively better renderings, delivering more accurate appearances and fewer outliers across most sequences.

### C.3. Comparison with Some Recent Baselines.

We compare with more recent baselines (including GFlow [45], MonST3R [52] and Align3R [27]) in Table 5 and Figure 6. As the code of GFlow is not released, we use its demo

Table 5. Quantitative results of train view fitting on DAVIS dataset (GFlow’s setting) and novel view synthesis on Dycheck dataset.

Method	View fitting			Novel view synthesis		
	SSIM $\uparrow$	LPIPS $\downarrow$	PSNR $\uparrow$	mSSIM $\uparrow$	mLPIPS $\downarrow$	mPSNR $\uparrow$
MonST3R	0.52	0.18	23.47	0.32	0.52	11.06
Align3R	0.60	0.17	24.02	0.35	0.46	12.74
GFlow	0.92	0.12	29.74	-	-	-
<b>4D-Fly</b>	<b>0.96</b>	<b>0.04</b>	<b>34.69</b>	<b>0.60</b>	<b>0.37</b>	<b>17.03</b>

and reported values (lacking NVS metrics). For MonST3R and Align3R, we first align the output point maps to the world coordinates using the predict extrinsics and GT extrinsics. Then we render the point map of eval timestep with Open3D. As point-based geometric models, they underperform in view fitting and synthesis tasks.

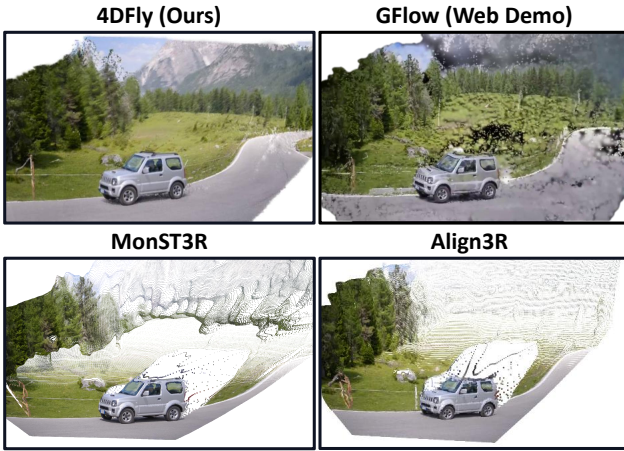


Figure 6. The NVS comparison with GFlow [45], MonST3R [52] and Align3R [27] on DAVIS dataset.

#### C.4. More Reconstruction visualizations

We provide visualizations with larger camera deviation (similar to the SOM [44] website demo) in Figure 7.



Figure 7. More visualization with larger camera deviation. Upper: Seq. breakdance-flare. Lower: Seq. breakdance-flare.

#### C.5. Visualization of Point Tracking

In Figure 10, we present the visualization of point tracking generated by 4D-Fly, which includes several sequences from the Dycheck iPhone dataset [11] and NVIDIA Dynamic Scenes dataset [51]. As shown, 4D-Fly demonstrates robust long-range tracking capabilities across various types of dynamic scenes.





Figure 8. Qualitative Comparison on the DyCheck iPhone Dataset with NeRF-based Baselines.



Figure 9. Qualitative Comparison on the Nvidia Dynamic Scenes Dataset.





Figure 10. **Visualization of Point Tracking Generated by 4D-Fly.** We first render novel view images and then overlay the predicted 3D point tracks on top. The sequences are from the Dycheck iPhone dataset and NVIDIA Dynamic Scenes dataset.