Shape of Motion: 4D Reconstruction from a Single Video — Supplementary Material —

Qianqian Wang[1,2*], Vickie Ye[1*], Hang Gao[1*], Weijia Zeng[3*],
Jake Austin[1], Zhengqi Li[4], Angjoo Kanazawa[1]

[1]UC Berkeley [2]Google DeepMind [3]UC San Diego [4]Adobe Research

1. Additional Preprocessing Details

Obtaining Camera Poses. Our method takes video sequences with known camera poses as input. To obtain camera poses for in-the-wild videos with moving objects, we adopt one of these two approaches depending on the type of input camera motion: (1) if there is sufficient camera motion parallax, we use COLMAP [12]'s SfM pipeline to obtain the camera poses and the sparse point clouds for the static regions, where we exclude keypoints in the foreground masks during the feature extraction stage. The foreground masks are generated using Track-Anything [15], a flexible and interactive tool for video object tracking and segmentation. The static point clouds produced by COLMAP [12] can then be used for aligning the affine-invariant monocular depth maps from Depth Anything [16]. (2) If the video is captured by a roughly stationary camera (small interframe camera baseline), COLMAP tends to fail catastrophically. Initial results on DAVIS use camera poses estimated with Unidepth [11] and DroidSLAM [14]. Specifically, we first employ Unidepth to predict metric depth and camera intrinsics. Using these predictions, we then estimate camera poses with DroidSLAM. (3) Camera parameters of in-the-wild videos in supplemental videos are estimated with MegaSaM [9].

Aligning Monocular Depth Maps. The dispairty output from Depth Anything [16] model is affine-invariant, so we need to align them with the cameras and reconstruction. To do so, we solve for a per-frame global scale and shift parameters that minimizes ℓ_2 distance between the monocular disparity from Depth Anything and the disparity derived from SfM sparse point clouds or depth outputs from DroidSLAM or MegaSaM.

Computing Long-Range 2D Tracks. We utilize TAPIR [1] to compute long-range 2D tracks for a video. While the ideal scenario would involve computing full-length tracks for every pixel in every frame (i.e., exhaustive pairs of correspondences), this approach is prohibitively computationally expensive. We therefore only compute full-length tracks for pixels located on a grid for foreground moving objects in each frame. In our experiment, we set grid interval to 4 (i.e., sampling a query point every 4 pixels). Due to our low-dimensional motion representation, we observe that our method can effectively operate with semi-dense 2D tracks without significant performance degradation During training, we filter out correspondences that TAPIR predicts with high uncertainty or those that are occluded.

2. Additional Training Details

2.1. Initialization Details

During initialization stage, we first solve a Procrustes alignment problem for each cluster b, where we estimate $\mathbb{SE}(3)$ transformation between point sets $\{\mathbf{X}_0\}_b$ and $\{\mathbf{X}_\tau\}_b$ for all $\tau = 0, \ldots, T$. We exclude point pair when either one of them is occluded, and weight the point pair using the uncertainty score predicted by TAPIR [1] when solving Procrustes. This process produces the initialization for the set of basis functions $\{\mathbf{T}^{(b)}_{0\to t}\}^B_{b=1}$. To initialize the motion coefficient $\mathbf{w}^{(b)}$ for each track, we compute the distance between the 3D location of the track in the canonical frame and the 3D location of each cluster center, and initialize each of the corresponding motion coefficient value to be exponentially decay with the distance.

We then optimize the $\boldsymbol{\mu}_0$, $\mathbf{w}^{(b)}$, and set of basis functions $\{\mathbf{T}^{(b)}_{0\to t}\}^B_{b=1}$ to fit the observed 3D tracks lifted by monocular depths and 2D tracks. Specifically, we enforce an ℓ_1 loss between each of our predicted 3D track and its corresponding observed 3D track. In addition, we enforce the motion bases to be temporally smooth by adding an ℓ_2 regularization on the acceleration of both the quaternion and the translation vector. We optimize the parameters using Adam [7] optimizer for 2k steps, where the initial learning rates for $\boldsymbol{\mu}_0$, $\mathbf{w}^{(b)}$, and $\{\mathbf{T}^{(b)}_{0\to t}\}^B_{b=1}$ are 1×10^{-3}, 1×10^{-2} and 1×10^{-2}, respectively. All learning rates are exponentially decayed to $\frac{1}{10}$ of their initial values during the optimization process.

2.2. Training Details

Gaussian Initialization. The aforementioned initialization stage gives the initialization of the mean of each Gaussian in the canonical frame. We follow the original 3D-GS [6] paper to initialize the scale $\mathbf{s}$, rotation $\mathbf{R}_0$, and opacity o of each Gaussian in the canonical frame. The color $\mathbf{c}$ of each Gaussian is initialized as the pixel color at the projected location in the canonical frame.

Optimization. We use Adam [7] Optimizer to optimize all scene parameters. The learning rates for Gaussian's canonical mean μ_0, opacity o, scale $\mathbf{s}$, rotation $\mathbf{R}_0$ (parameterized as quaternion) and color $\mathbf{c}$ are set to 1.6×10^{-4}, 1×10^{-2}, 5×10^{-3}, 1×10^{-3}, and 1×10^{-2}, respectively. The learning rates for the $\mathbb{SE}(3)$ motion bases and the motion coefficients are set to 1.6×10^{-4} and 1×10^{-2} respectively. During each training iteration, we randomly select a batch of 8 query frames. For each query frame, we render the color, mask, depth, and the 3D track locations for 4 randomly selected target frames.

Loss weights and details. Our first set of loss function enforces our rendered results to match the per-frame pixelwise color, depth, and masks inputs. The coefficients for depth loss λ_{depth} and mask loss λ_{mask} are set to 0.5 and 1.0, respectively. We additionally add regularization to the estimated surface geometry via a depth gradient loss and per-Gaussian scale penalty. In particular, We add a pixelwise ℓ_1 loss of spatial gradient between the depth renderings $\hat{\mathbf{D}}_t$ and corresponding reference depth maps $\mathbf{D}$, with a weight set to 1.0. In addition, we enforce the foreground Gaussian to be isotropic by incorporating a regularization term that penalizes standard deviations of the scale $\mathbf{s}$ along all three axes.

Our second set of losses supervise the motion of the Gaussian. The weights for the 2D tracking loss $\lambda_{\text{track-2d}}$ and the track depth loss $\lambda_{\text{track-depth}}$ are set to 2.0 and 0.1, respectively. The $L_{\text{track-2d}}$ is applied on normalized pixel coordinates (i.e., divided by the maximum image edge length). For the rigidity loss L_{rigidity}, we use foreground part masks from SAM automatic segmentation independently on each frame. For each training iteration, we sample 4 part masks for each query frame. For each mask, we sample 32 center points and find their 16 nearest neighbors within the mask. We compute the 3D track locations for all point samples in 4 target frames, and regularize the distances between the center points and their neighbors in the target frames to be similar to their distances in the query frame. We let $\lambda_{\text{rigidity}} = 0.1$. We weight the neighbor distances with an exponential kernel $\exp(-\beta\|x - \bar{x}\|)$ with $\beta = 2$, where $\bar{x}$ is the center point and x is one of its neighbors. We additionally add motion smoothness regularization that enforces an ℓ_2 loss on acceleration of the motion translation bases and motion quaternion bases, with a weight set to 0.1 and an ℓ_2 loss on the acceleration along z-axis (in the camera frame) of the μ_t, both through finite difference approximation.

Training with 2D Gaussian Splatting (2DGS) To enhance scene geometry estimation, we replace 3D Gaussian Splatting with 2D Gaussian Splatting [5]. Specifically, we employ an off-the-shelf monocular normal estimator [4] to generate monocular normal maps, $\tilde{N}$, which are then used to supervise both the rendered normals and the normals derived from the rendered depth. The supervision is achieved through the following losses:

$$L_n = \sum_i \omega_i (1 - n_i^T \tilde{N}) \tag{1}$$

$$L_N = 1 - N^T \tilde{N} \tag{2}$$

where the summation is over the splats intersect the current ray, ω_i represents the blending weight for i-th splat, and n_i denotes the normal of i-th splat. Here, N represents the normal derived from rendered depth map. This supervision ensures that the orientation and depth of the 2D splats are aligned with the monocular normal prediction.

3. Additional Evaluation Details

In our evaluation, since the synthetic Kubric dataset [3] comes with groundtruth camera poses, we directly use the groundtruth camera poses for our experiments. For iPhone dataset [2], we observed that the provided camera poses from ARKit are not accurate, we thus perform an additional global bundle adjustment using COLMAP [12] to refine the camera poses while fixing the camera intrinsics. To maintain metric scale after refinement, we compute a global $\mathbb{SIM}(3)$ transformation for each scene to align the refined camera poses with the original metric-scale camera poses. This allows us to evaluate 3D tracking performance in metric scale.

4. Visualization of Kubric Experiment

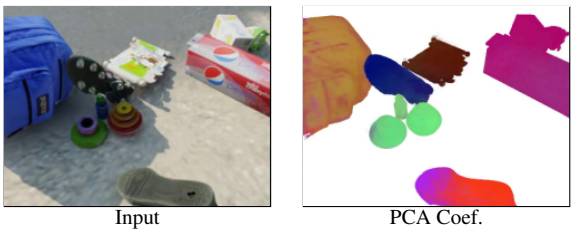
We find qualitatively that the optimized motion coefficients of the scene representation are coherently grouped with each moving object in the scene. We demonstrate this in Figure 1, where we show the first 3 PCA components of our optimized motion coefficients of evaluation scenes.

5. Visualization of Complex Dynamic Scene

Our method is able to handle scenes with multiple moving objects. We visualize novel views and motion PCA of a real world complex scene in Fig. 2.

6. NVIDIA Dataset Evaluation

We conduct experiments on seven scenes from the NVIDIA Dynamic Scenes dataset [17], following the evaluation protocol of Dynamic Gaussian Marbles [13]. Specifically, we



Input PCA Coef.

Figure 1. **First three PCA components of the optimized motion coefficients.**

Figure 2. Novel view and motion coefficient PCA visualizations at time steps 0 and 54 of the *school-girl* sequence from the DAVIS dataset.

use video footage from the static camera 4 for training and employ videos from cameras 3, 5, and 6 for evaluation. For consistency with Dynamic Gaussian Marbles [13]'s experiments, we use images at half-resolution for all seven experiments, though our method is compatible with high-resolution images as well. Camera poses are estimated using COLMAP [12], and depths are predicted with Depth Anything [16] and subsequently aligned with the COLMAP point cloud. The quantitative results reported in this paper differ from those in DGM [13]. Specifically, we compute covisibility masks between training and test views and apply them during evaluation, whereas DGM [13] inpaints unobserved regions in test views for evaluation. This difference in evaluation procedure accounts for the variation in reported performance. We will make the covisibility masks publicly available to facilitate future research.

7. DynMF Implementation Detail

DynMF [8] shares similar design choices with our method. Since they don't have public code available, we implement our own version of it and provide implementation details here. Follow equation (4) in DynMF [8], each motion basis is represented as a MLP:

$$b_j(t) = MLP_j\left(\frac{t}{T}\right) \tag{3}$$

where b_j is the $j-th$ motion basis, t is the queried time step, and T is the number of time frames. We use a total of 10 motion bases. We apply positional encoding with 10 frequency bandwidths [10] to the input $\frac{t}{T}$. Following DynMF [8], we apply displacement motion to Gaussian's means and rotation quaternions as follows:

$$\mu(t) = \mu_c + \sum_{j=1}^{10} c_j b_j^{\mu}(t) \tag{4}$$

where μ_c is the Gaussian means in the canonical frame and $\{b_j^{\mu}(t)\}_{j=1}^{10}$ are the mean motion bases represented as MLPs.
and

$$q(t) = q_c + \sum_{j=1}^{10} c_j b_j^{R}(t) \tag{5}$$

where q_c is the Gaussian rotation represented as quaternion in the canonical frame and $\{b_j^{R}(t)\}_{j=1}^{10}$ are the rotation motion bases represented as MLPs.The motion coefficients $\{c_j\}_{j=1}^{10}$ are shared between mean and rotation motions.

References

[1] Carl Doersch, Yi Yang, Mel Vecerik, Dilara Gokay, Ankush Gupta, Yusuf Aytar, Joao Carreira, and Andrew Zisserman. Tapir: Tracking any point with per-frame initialization and temporal refinement. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023. 1

[2] Hang Gao, Ruilong Li, Shubham Tulsiani, Bryan Russell, and Angjoo Kanazawa. Dynamic novel-view synthesis: A reality check. In *NeurIPS*, 2022. 2

[3] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, et al. Kubric: A scalable dataset generator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 2

[4] Jing He, Haodong Li, Wei Yin, Yixun Liang, Leheng Li, Kaiqiang Zhou, Hongbo Zhang, Bingbing Liu, and Ying-Cong Chen. Lotus: Diffusion-based visual foundation model for high-quality dense prediction, 2024. 2

[5] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery, 2024. 2

[6] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 2023. 2

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1, 2

[8] Agelos Kratimenos, Jiahui Lei, and Kostas Daniilidis. Dynmf: Neural motion factorization for real-time dynamic view synthesis with 3d gaussian splatting. *ECCV*, 2024. 3

[9] Zhengqi Li, Richard Tucker, Forrester Cole, Qianqian Wang, Linyi Jin, Vickie Ye, Angjoo Kanazawa, Aleksander Holynski, and Noah Snavely. Megasam: Accurate, fast and robust structure and motion from casual dynamic videos. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, 2025. 1

[10] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 3

[11] Luigi Piccinelli, Yung-Hsu Yang, Christos Sakaridis, Mattia Segu, Siyuan Li, Luc Van Gool, and Fisher Yu. UniDepth:

Universal monocular metric depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 1

[12] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 2, 3

[13] Colton Stearns, Adam Harley, Mikaela Uy, Florian Dubost, Federico Tombari, Gordon Wetzstein, and Leonidas Guibas. Dynamic gaussian marbles for novel view synthesis of casual monocular videos. *Siggraph Asia*, 2024. 2, 3

[14] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Advances in neural information processing systems*, 2021. 1

[15] Jinyu Yang, Mingqi Gao, Zhe Li, Shang Gao, Fangjing Wang, and Feng Zheng. Track anything: Segment anything meets videos. *arXiv preprint arXiv:2304.11968*, 2023. 1

[16] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2024. 1, 3

[17] Jae Shin Yoon, Kihwan Kim, Orazio Gallo, Hyun Soo Park, and Jan Kautz. Novel view synthesis of dynamic scenes with globally coherent depths from a monocular camera. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 2