Dynamic Point Maps: A Versatile Representation for Dynamic 3D Reconstruction

Edgar Sucar Zihang Lai Eldar Insafutdinov Andrea Vedaldi
Visual Geometry Group (VGG), University of Oxford
{edgarsucar,zlai,eldar,vedaldi}@robots.ox.ac.uk

Figure 1. We introduce the concept of **Dynamic Point Maps** (DPM). Differently from previous extension of point maps to dynamics, DPMs are *time invariant* in addition to be viewpoint invariant. Because of this, by predicting DPMs from a pair of images in a feed-forward manner we can easily solve key 3D tasks, such as recovering the camera parameters and reconstructing shape, as well 4D ones, such as estimating scene flow and 3D object tracking.

Abstract

DUSt3R has recently demonstrated that many tasks in multi-view geometry, including estimating camera intrinsics and extrinsics, reconstructing 3D scenes, and establishing image correspondences, can be reduced to predicting a pair of viewpoint-invariant point maps, i.e., pixel-aligned point clouds defined in a common reference frame. While this formulation is elegant and powerful, it is limited to static scenes. To overcome this limitation, we introduce the concept of Dynamic Point Maps (DPM), which extends standard point maps to support 4D tasks such as motion segmentation, scene flow estimation, 3D object tracking, and 2D correspondence. Our key insight is that, when time is introduced, several possible spatial and temporal references can be used to define the point maps. We identify a minimal subset of these combinations that can be regressed by a network to solve the aforementioned tasks. We train a DPM predictor on a mixture of synthetic and real data and evaluate it across diverse benchmarks, including video depth prediction, dynamic point cloud reconstruction, 3D scene flow, and object pose tracking, achieving state-of-the-art performance. Additional results are available at https://www.robots.ox.ac.uk/~vgg/research/dynamic-point-maps/.

1. Introduction

The impact of machine learning on 3D computer vision has been growing steadily. For instance, DUSt3R [69], a recent breakthrough, proposed learning a neural network that, given two images of a scene, maps each pixel to its corresponding 3D point, expressed in a shared 3D reference frame. Notably, they showed that knowledge of these viewpoint-invariant *point maps* enables solving a variety of core 3D tasks, such as estimating camera intrinsics and extrinsics (by aligning

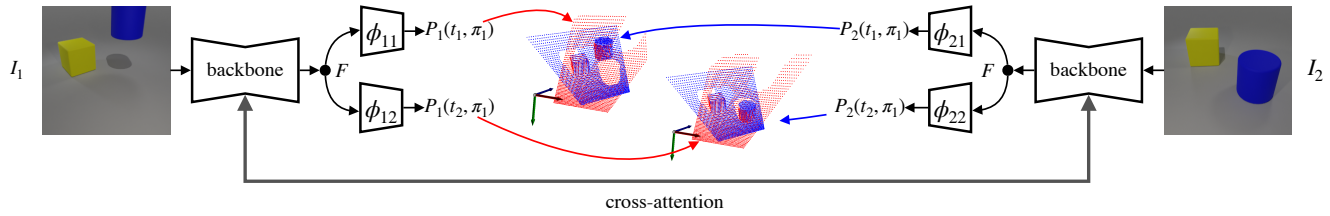

Figure 2. We propose to extend DUSt3R to predict **Dynamic Point Maps** (DPM). Each image in the pair is mapped to two point maps that correspond to the timestamps of the two images (pairs share the same colour in the figure). All points map are defined in the reference frame of image I_1, undoing the effect of viewpoint change. Scene flow and space-time correspondences can be inferred immediately.

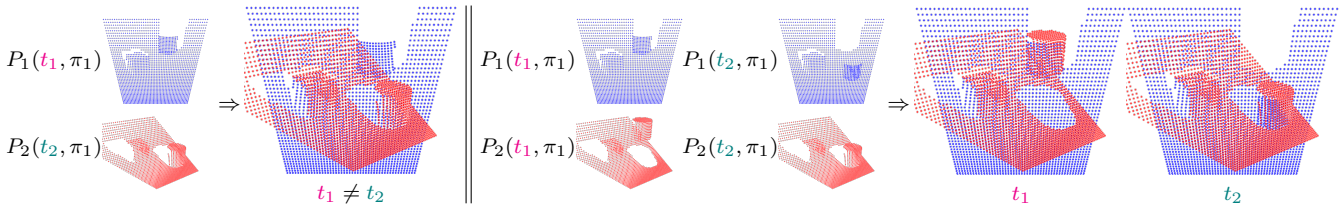

Figure 3. Left: Standard point maps applied to dynamic scenes as in MonST3R [80] fail to represent dynamics. The cylinder, which is moving downwards, breaks invariance when the point maps are overlaid. Right: Our Dynamic Point Maps correctly represent dynamics by also controlling time in addition to viewpoint. They allow to restore invariance while still representing the motion of the cylinder.

pixels to their 3D points along camera rays), monocular depth estimation (by providing two identical copies of the same image to the model), and 2D matching (by comparing the reconstructed 3D points). However, a key limitation is that their point maps cannot *explain dynamic 3D content*.

Dynamic scenes are ubiquitous in the real world, and interpreting and reconstructing them in 3D is potentially one of the most impactful applications of 3D computer vision, but also one of the most challenging. Even state-of-the-art methods for dynamic 3D reconstruction [31, 39, 50] still rely on ad hoc designs, combining multiple learned modules, including depth estimators, matchers, and segmenters, and require expensive and fragile test-time optimization. This motivates us to explore how the simple and elegant approach of DUSt3R could be extended to dynamic data.

The key question we aim to answer in this paper is whether and how the point maps representation can be used to tackle dynamic 3D reconstruction tasks. The key to point maps is their invariance to camera parameters, including viewpoint. For static scenes, where only the camera changes, viewpoint invariance suffices. However, in dynamic scenes, the 3D points themselves change over time. While we can still compute the point maps as defined in DUSt3R, the outputs are no longer invariant because, even with a fixed camera, the 3D points move.

For example, this approach was explored by the recent MonST3R [80]. While they achieved outstanding results, the technical limitations of their chosen representation are evident. Specifically, due to the lack of invariance, they cannot predict corresponding 3D points directly and must combine their output with optical flow to do so. We argue that the ability to easily establish multi-view correspondences is a fundamental strength of DUSt3R's representation, and preserving this capability is essential for any extension to dynamic scenes.

In this paper, we introduce *Dynamic Point Maps* (DPM), a new formulation that satisfies this requirement and extends DUSt3R to dynamic scenes (Fig. 2). The key insight of DPM is that invariance in dynamic scenes requires fixing both the camera viewpoint and the scene time. Consider a pair of images of a scene, each associated with a specific viewpoint and timestamp. For each image, we introduce a *pair of point maps* (Fig. 3) that map each pixel to two versions of the corresponding 'physical' 3D point: one corresponding to the timestamp of the first image and the other to the timestamp of the second image. As in DUSt3R, all 3D points are referred to the reference frame of the first image.

We argue that this is the minimal design capable of tackling 4D tasks in full generality. For static scenes, this approach directly generalizes DUSt3R, as the two dual point maps are identical; it also generalizes MonST3R, as two of the four dual point maps match theirs. Crucially, the dual point maps encode both the scene *motion* and *dynamic correspondence*, eliminating the need for additional processing, such as optical flow, as required by MonST3R. Specifically, because each physical point is reconstructed at both timestamps, we can directly infer scene flow and motion segmentation and use this information to track the motion of rigid objects. Furthermore, since physical points from different cameras are available in the same spatio-temporal reference frame—undoing the effects of viewpoint changes (by fixing the viewpoint) and scene motion (by fixing time)—it be-

comes trivial to match them across views. This also enables the fusion of point clouds despite motion in the scene.

We empirically demonstrate that the learned model can successfully handle dynamic scenes, addressing the tasks discussed above. In summary, our contributions are: (1) Introducing the concept of Dynamic Point Maps, which extends point maps to dynamic scenes, enabling solutions to various 3D and 4D reconstruction tasks; (2) Showing that the DUSt3R model can be extended to output such dynamic maps, fine-tuning it on a mixture of datasets while generalizing well to real data; (3) Demonstrating the effectiveness of the proposed method across multiple tasks, including motion estimation, 4D reconstruction, and rigid object tracking. Overall, our results highlight the potential of DPMs as a foundation for new designs of 3D models capable of tackling dynamic scenes.

2. Related Work

Structure from Motion (SfM). Recovering the geometry of a static scene from a collection of images has been a long-standing problem in computer vision [15, 19, 45]. COLMAP [46] is perhaps the most popular implementation of the 'classical' pipelines based on keypoint detection, description, matching, and bundle adjustment [1, 16]. With the advent of machine learning, many researchers have attempted to use neural networks for SfM, often to improve individual components like point matching [10, 12, 25, 34, 44, 49, 58, 79], but also by experimenting with fully differentiable SfM pipelines [53, 55, 60, 65, 70]. VGGSfM [66] is perhaps the most successful example to date of a method that improves bundle adjustment by learning data priors and using them to address ambiguous situations, such as matching texture-less regions. Of particular relevance to this paper is DUSt3R [69], which introduces a unified representation for jointly predicting camera motion and 3D geometry.

Non-rigid Structure from Motion (NR-SfM). When dynamic elements are present in the scene, the problem becomes significantly more challenging due to higher ambiguity and the inability to directly triangulate points. Early works [6, 57] proposed assumptions under which the problem becomes meaningfully solvable. These assumptions were further refined in follow-up works such as [2, 3, 11, 61]. Several researchers also explored dense dynamic reconstruction [30, 41, 43].

With the advent of neural radiance fields, several researchers extended the classic NR-SfM problem to include scene appearance reconstruction. Methods like DyniBar [33], DCT-NeRF [63], NSFF [32], and Dynamic View Synthesis [17] fit deformations between frames, whereas works such as D-NeRF [40], NeRFlow [14], Nerfies [37], Spacetime NeRF [72], HyperNeRF [38], Deformable 3D Gaussians [77], 4D GS [71], and [78] instead fit deformations of each frame back to a shared canonical reconstruction. Shape of Motion [67] models longer-term deformations by explicitly fitting individual 3D Gaussian tracks to entire videos. All these methods require expensive test-time optimization.

Concurrently with our work, DUSt3R was recently extended by MonST3R [80], CUT3R [68], and Stereo4D [26] to tackle dynamic content with very good results. However, MonST3R and CUT3R do not explore the full formulation design space, and their extensions only consider freezing time, which cannot solve all the 4D tasks that our method addresses. Stereo4D does consider dynamic flow but lacks the concept of invariant point clouds and the exploration of downstream tasks.

Optical Flow. Finding dense correspondences between images is usually cast as an optical flow prediction problem [5, 7–9, 20, 35]. Early optical flow methods that used deep learning include Flow Fields [4], FlowNet [13, 22], PWC-Net [51], and RAFT [54]. Some works [23, 42, 47] consider multi-frame optical flow. Others [24] suggested that the priors learned by neural networks are helpful for tracking points behind occlusions. GMFlow [74] proposed treating optical flow estimation as a global matching problem rather than a local one. Transformer-based approaches, such as FlowFormer [21, 48], have also been explored.

Scene Flow. The *scene flow* is a set of 3D correspondences between scene points at two different times. Examples of methods that can recover scene flow include RAFT-3D [56], SpatialTracker [73], and SceneTracker [62]. Shape of Motion [67] also estimates scene flow as a byproduct. TAPVid-3D [29] proposes a new benchmark to assess scene flow.

3. Method

We start by reviewing the concept of point maps as used in DUSt3R in Sec. 3.1. We then introduce our new Dynamic Point Maps representation in Sec. 3.2 and explain how we extend DUSt3R to predict it. In Sec. 3.3, we describe how we train our model and discuss the training datasets.

3.1. Point Maps

Let I be an image containing a scene we wish to reconstruct in 3D. We cast this as the problem of predicting the corresponding *point map* P, which associates each pixel $\boldsymbol{u}$ with the corresponding scene point $\boldsymbol{p} = P(\boldsymbol{u}) \in \mathbb{R}^3$, expressed in the reference frame π of the camera. A point map is thus similar to a depth map but contains strictly more information. In fact, the point map can be recovered from the depth map only if the intrinsic parameters of the camera (e.g., the focal length) are given; conversely, knowledge of the point map allows us to infer the camera intrinsics.

Next, consider two images I_1 and I_2 taken by two cameras with different viewpoints π_1 and π_2 and corresponding point maps P_1 and P_2. DUSt3R recently proposed a

model where both point maps are expressed in the same reference frame as the first image. We make this explicit by writing $P_1(\pi_1)$ and $P_2(\pi_1)$, so that $P_i(\pi_j)$ means that the point map is extracted from image i and expressed in the reference frame of camera j. This simple change has profound consequences, as it means that the point maps are *viewpoint invariant*. This implies that, if $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ are two pixels corresponding to the same 3D point, then $P_1(\pi_1)(\boldsymbol{u}_1) = P_2(\pi_1)(\boldsymbol{u}_2)$.

As DUSt3R noted, this invariance is sufficient to solve a number of core 3D tasks, such as estimating the camera extrinsics (*i.e.*, the relative camera motion), establishing point correspondences between images, aligning and fusing point clouds, and so on. It also allows us to infer the camera intrinsics and depth. Their idea, then, is to learn a neural network Φ that predicts the point maps from the two images, as $(P_1(\pi_1), P_2(\pi_1)) = \Phi(I_1, I_2)$. The fact that the point maps are viewpoint invariant simplifies the network's task, making it more similar to a labeling problem.

The most significant limitation of the DUSt3R model is that it cannot handle dynamic scenes. We tackle this issue in the next section.

3.2. Dynamic Point Maps

Consider two images I_1 and I_2 taken at two different times t_1 and t_2. If we consider the point maps $P_1(\pi_1)$ and $P_2(\pi_1)$ defined in Sec. 3.1, which is also the approach taken by [80], we encounter a problem: the representation is *no longer invariant*. While both point maps are defined with respect to the *same* reference frame π_1, the 3D points themselves move over time, so in general, $P_1(\pi_1)(\boldsymbol{u}_1) \neq P_2(\pi_1)(\boldsymbol{u}_2)$. This is why MonST3R relies on an additional image matching network to establish correspondences $(\boldsymbol{u}_1, \boldsymbol{u}_2)$ between pixels, which is necessary for estimating scene motion and other tasks.

We can restore invariance by controlling not only for viewpoint but also for time, estimating point maps $P_1(t_1, \pi_1)$ and $P_2(t_1, \pi_1)$. With this notation, we mean that the 3D points in both point maps are referred to the same reference frame π_1 *and* time t_1 as the first camera, effectively undoing both viewpoint change and scene motion. In this way, we can re-establish the invariance property $P_1(t_1, \pi_1)(\boldsymbol{u}_1) = P_2(t_1, \pi_1)(\boldsymbol{u}_2)$. However, by undoing scene motion, we lose the ability to estimate it, which is key for motion analysis.

Our solution is to introduce the concept of *Dynamic Point Maps* (DPM). This involves predicting, for each image, a *pair* of point maps, expressing the points at two timestamps t_1 and t_2. With our notation, image I_1 is associated with the pair of point maps $P_1(t_1, \pi_1)$ and $P_1(t_2, \pi_1)$, and image I_2 with point maps $P_2(t_1, \pi_1)$ and $P_2(t_2, \pi_1)$.

Note that all point maps are expressed in the same reference frame π_1 of the first camera.[1] Pairs with the same argument share the same spatio-temporal reference frame and are thus invariant.[2] For example, we can establish correspondences between the two images by matching the 3D points in $P_1(t_1, \pi_1)$ and $P_2(t_1, \pi_1)$. At the same time, we can recover scene flow simply by taking the difference $P_1(t_2, \pi_1) - P_1(t_1, \pi_1)$.

Just as DUSt3R is a powerful representation for static scenes, DPM is a powerful representation for dynamic ones. In addition to subsuming all tasks that can be addressed in the static case (e.g., estimation of camera intrinsics and extrinsics, 3D reconstruction, etc.), it can also solve a number of tasks specific to 4D reconstruction, such as deformation-invariant matching and scene flow estimation, and assist with others, such as estimating rigid body motion. In Appendix A, we show some of these reductions more formally. Most importantly, as we show in the next section, we can equip DUSt3R with DPM relatively easily.

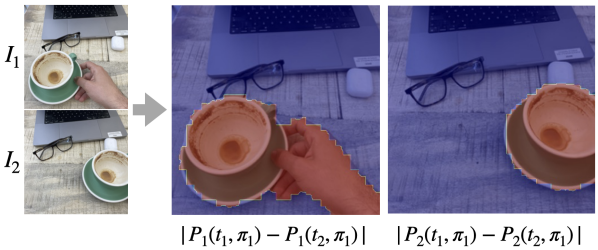

Figure 4. **Motion segmentation:** from a pair of images using the dynamic point cloud predicted by the network we can segment out the dynamic elements of the scene, despite the camera motion.

Dynamic DUSt3R. Recall that DUSt3R learns a network Φ that maps images I_1 and I_2 to a pair of point maps. Here, we extend this network to simply output six instead of three channels per image by adding suitable heads, so that each image is mapped to two maps. This can also be seen as a function

$$\{P_i(t_j, \pi_1)\}_{i,j\in\{1,2\}} = \Phi(I_1, I_2). \tag{1}$$

Hence, four point maps are estimated in total, two for each image $i = 1, 2$ and time $j = 1, 2$. Note that all point maps are referred to the reference frame π_1 of the first camera.[3] By predicting such a DPM, this network is sufficient to solve a number of 4D tasks, as we have discussed above and further in Appendix A.

3.3. Training Formulation

To supervise the model Φ, we require video sequences with corresponding dynamic point maps;

[1] The problem is symmetric, so we can obtain all quantities referred to the second camera by simply swapping the inputs to the network.

[2] *I.e.*, $P_1(t_1, \pi_1)(\boldsymbol{u}_1) = P_2(t_1, \pi_1)(\boldsymbol{u}_2)$ and $P_1(t_2, \pi_1)(\boldsymbol{u}_1) = P_2(t_2, \pi_1)(\boldsymbol{u}_2)$ for all pairs of corresponding pixels $(\boldsymbol{u}_1, \boldsymbol{u}_2)$.

[3] Switching to the second camera is trivial, as we can simply swap the inputs to the network, and recovers four more point maps, for a total of eight possible combinations.

namely, the training data $\mathcal{D}$ is a collection of tuples $(I_1, I_2, P_1(t_1, \pi_1), P_1(t_2, \pi_1), P_2(t_1, \pi_1), P_2(t_2, \pi_1))$. To obtain such examples, we use a mixture of synthetic and real data providing various degrees of supervision. Synthetic data has already been shown to be very effective for a number of low-level computer vision tasks, such as optical flow [13], tracking [27], and depth prediction [76]. With synthetic data, we have perfect knowledge of the underlying scene geometry, including its *deformation* due to the dynamic parts of the scene. With this, we can determine for each pixel $\boldsymbol{u}$ in image I_1 which is the corresponding 3D point $\boldsymbol{p}(t_1, \pi_1)$. Because we know the camera motion, we can then recover $\boldsymbol{p}(t_1, \pi_2)$. Because we know the deformation of the scene, we can also recover $\boldsymbol{p}(t_2, \pi_1)$ and $\boldsymbol{p}(t_2, \pi_2)$ for image I_2.

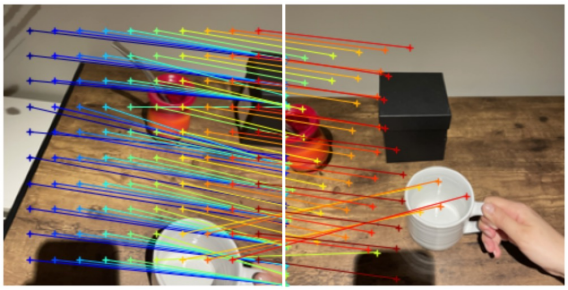

Figure 5. **Point correspondence:** DPMs allow to recover correspondences accounting both for dynamic objects as well as camera motion even when challenging viewpoint changes.

Training Loss. We note that the scale of a 3D scene cannot be determined uniquely from any number of views; therefore, we relax the predictions to be determined up to a scaling factor. In order to do so, let $P \in \mathbb{R}^{3\times HW}$ be a ground-truth point cloud and let $\hat{P}$ be its predicted counterpart. We then define per-pixel regression loss given by [69]

$$L_{\text{reg}}(\hat{P}, P, i) = \left\| \frac{\hat{P}_{:,i}}{\frac{1}{HW}\sum_{j=1}^{HW} \|\hat{P}_{:,j}\|} - \frac{P_{:,i}}{\frac{1}{HW}\sum_{j=1}^{HW} \|P_{:,j}\|} \right\|.$$

This makes sense because points are expressed in the reference frame of one of the cameras, so, for example, $\|P_{:,i}\|$ is the distance of point $P_{:,i}$ with respect to the camera center. The full loss follows the confidence-calibrated formulation from [28, 36, 69]:

$$L_{\text{conf}}(\hat{P}, P) = \frac{1}{HW}\sum_{i=1}^{HW} C_i L_{\text{reg}}(\hat{P}, P, i) - \alpha \log C_i. \quad (2)$$

Recall that network Φ predicts four separate point clouds. For simplicity, we stack them into point clouds P and $\hat{P}$ and minimize $L_{\text{conf}}(\hat{P}, P)$. Again, this makes sense because all point clouds are defined in the same reference frame π_1.

Dataset Details. We train our model on a total of 7 datasets, see the Appendix B.1. We use the Kubric data pipeline [18] to generate a new synthetic dataset, which we call MOVi-G, to train our model following the Multi-Object Video (MOVi) dataset format but with more complex camera trajectories through spline interpolation and randomized intrinsics. For generating dynamic point cloud ground truth, we use GT bounding box tracking and NOCS [64] coordinates to transform points between different reference times and viewpoints. We render 9,750 clips and sample 6 random pairs of images from each clip for a total of 58,500 training samples. We also add the Waymo dataset [52] where we compute dynamic point maps from LiDAR data in a similar way, and PointOdyssey which includes sparse ground-truth 3D tracks. For the rest of the datasets, we either omit supervision for cross-time reconstructions ($P_2(t_1, \pi_1)$ and $P_1(t_2, \pi_1)$) or in the case of fully static scenes—they represent the identical geometry and therefore require no dynamic deformations.

Network Details. The model architecture is based on the network design from DUSt3R (Fig. 2). For $T = 2$, it shares the same backbone but adds two additional heads ϕ_{ij} for prediction, for a total of four regression outputs. Each output comprises both a point map $P_i(t_j, \pi_1)$ and a confidence map $C_i(t_j, \pi_1) \in [0, 1]^{HW}$, similar to [69]. They are thus predicted as $(P_i(t_j, \pi_1), C_i(t_j, \pi_1)) = \phi_{ij}(F)$, $i, j \in \{1, 2\}$, where F are the features computed by the transformer encoder-decoder backbone. We initialize both heads associated with each point cloud with the single head from DUSt3R, which should approximate at the start of training the static reconstruction component of that image. We train our network with $(512, 288)$ and $(512, 336)$ resolutions.

4. Evaluation

We evaluate DPM on several 3D and 4D reconstruction tasks. In Sec. 4.1, we show that DPM is on par with the state-of-the-art in depth estimation. Next, in Sec. 4.2, we shift our focus to dynamic 3D reconstruction, our key novelty, and show that here DPM is significantly better than alternatives. Finally, in Sec. 4.3, we provide a qualitative analysis.

For a fair comparison on the dynamic reconstruction, we fine-tune MonST3R using our custom Kubric dataset. The results for original MonST3R are shown in the Appendix C.

4.1. Depth prediction

Two-view depth evaluation. Since DPM is a generalization of DUSt3R, we first evaluate it on the task of stereo depth prediction in Tab. 1. We consider several standard benchmarks of dynamic scenes, including Bonn, Sintel, Point Odyssey, Kubric, and KITTI (crop).[4] We report standard depth metrics: Absolute Relative Error (Abs Rel) and $\delta < 1.25$ accuracy [75]. Stereo pairs are formed by choosing pairs of

[4]We use a cropped version of KITTI because the original images have an extreme aspect ratio (512×144).

Model	Sintel		Point Odyssey		Bonn		Kubric		KITTI (crop)	
	Abs Rel	$\delta < 1.25$	Abs Rel	$\delta < 1.25$	Abs Rel	$\delta < 1.25$	Abs Rel	$\delta < 1.25$	Abs Rel	$\delta < 1.25$
MonST3R	0.347	57.3	0.065	95.3	**0.071**	**94.1**	0.166	77.9	0.069	94.5
DPM	**0.296**	**59.2**	**0.056**	**96.1**	0.075	92.7	**0.078**	**95.0**	**0.052**	**96.8**

Table 1. **Depth Evaluation from 2-View Input**: Comparison of MonST3R and DPM across Sintel, Point Odyssey, Bonn, Kubric, and KITTI datasets. Our DPM model consistently yields lower absolute relative errors on challenging benchmarks—delivering an average reduction of roughly 17.5% in absolute relative error.

Category	Method	Sintel		Bonn		KITTI		KITTI (crop)	
		Abs Rel ↓	$\delta < 1.25$ ↑	Abs Rel ↓	$\delta < 1.25$ ↑	Abs Rel ↓	$\delta < 1.25$ ↑	Abs Rel ↓	$\delta < 1.25$ ↑
1-frame	Marigold	0.532	51.5	**0.091**	**93.1**	0.149	79.6	—	—
	DepthAnythingV2	**0.367**	**55.4**	0.106	92.1	**0.140**	**80.4**	—	—
Video depth	NVDS	0.408	48.3	0.167	76.6	0.253	58.8	—	—
	ChronoDepth	0.687	48.6	0.100	91.1	0.167	75.9	—	—
	DepthCrafter	**0.292**	**69.7**	**0.075**	**97.1**	**0.110**	**88.1**	—	—
Joint D&P	Robust-CVD	0.703	47.8	—	—	—	—	—	—
	CasualSAM	0.387	54.7	0.169	73.7	0.246	62.2	—	—
	MonST3R	0.335	**58.5**	**0.063**	**96.4**	**0.104**	**89.5**	0.111	87.2
	DPM	**0.311**	58.0	0.064	94.8	0.128	81.0	**0.097**	**89.1**

Table 2. **Video Depth Evaluation:** Performance Comparison between single-frame, video depth, and optimization-based models on Sintel, Bonn, and KITTI datasets. Our approach demonstrates robust performance across all datasets.

		L_{rel}				Object Pose Error	
Dataset	Method	$P_1(t_1)$	$P_2(t_1)$	$P_1(t_2)$	$P_2(t_2)$	RPE rot	RPE trans
Kub.-F	MonST3R	0.047	0.095	0.122	0.039	56.1	0.504
	Ours	**0.041**	**0.047**	**0.049**	**0.035**	**33.7**	**0.053**
Kub.-G	MonST3R	0.061	0.154	0.188	0.062	N/A	
	Ours	**0.057**	**0.071**	**0.079**	**0.058**		
Waymo	MonST3R	0.197	0.221	0.249	0.178	N/A	
	Ours	**0.068**	**0.065**	**0.067**	**0.065**		

Table 3. **Dynamic reconstruction**. We compare our method with MonST3R [80]+RAFT [54] on two tasks: relative point cloud error and object pose tracking. MonST3R struggles with predicting motion whereas our method explicitly trained on this task consistently performs better across datasets and metrics. $P_k(t_k)$ denotes $P_k(t_k, \pi_1)$ for brevity.

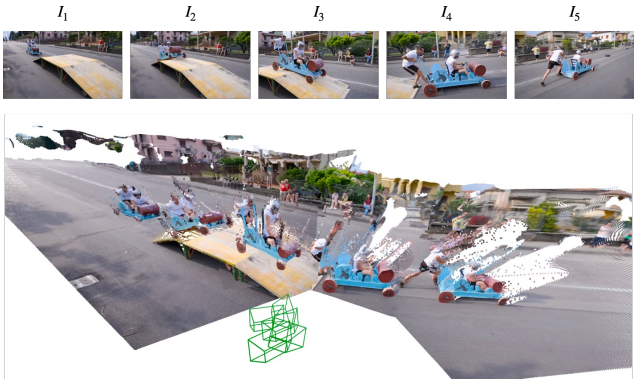

Figure 6. **Camera tracking:** the explicit modeling of a dynamic point cloud allows to recover camera motion by directly aligning point clouds with matching timestamp.

			Scene Flow		Object Flow	
Dataset	Method	Input	Forward	Backward	Forward	Backward
Kub.-F	MonST3R	RGB	0.113	0.088	0.111	0.084
	RAFT-3D	RGBD	**0.051**	**0.054**	N/A	N/A
	Ours	RGB	0.081	0.070	**0.033**	**0.029**
Kub.-G	MonST3R	RGB	0.171	0.177	0.175	0.151
	RAFT-3D	RGBD	4.067	4.084	N/A	N/A
	Ours	RGB	**0.104**	**0.106**	**0.059**	**0.050**
Waymo	MonST3R	RGB	0.182	0.180	0.180	0.195
	RAFT-3D	RGBD	0.150	0.145	N/A	N/A
	Ours	RGB	**0.051**	**0.051**	**0.030**	**0.032**

Table 4. **3D End-Point Error (EPE) for Scene Flow and Object Flow:** Despite relying solely on RGB input, our method performs comparably to RAFT-3D [54], which utilizes ground truth depth (D), in Kubric-F and surpasses it in Kubric-G and Waymo. For Object Flow estimation, RAFT-3D is incapable of performing the task, whereas our method achieves the best results across all datasets.

frames at random within a set margin. The results show that DPM outperforms MonST3R on all datasets except Bonn.

Video depth evaluation. Next, in Tab. 2, we consider depth prediction for longer video sequences. To fuse pairwise predictions from our network, we use a bundle adjustment strategy similar to that of MonST3R, but with a key modification: we remove the optical flow loss. This eliminates the overhead associated with computing optical flow using an additional model, resulting in a more efficient pipeline while still delivering high-quality depth predictions. We evaluate this approach on Sintel, Bonn, and KITTI datasets and show competitive performance with competitors including

Figure 7. **Rigid object motion:** 3D bounding box object tracking by masking and aligning the predicted dynamic point clouds.

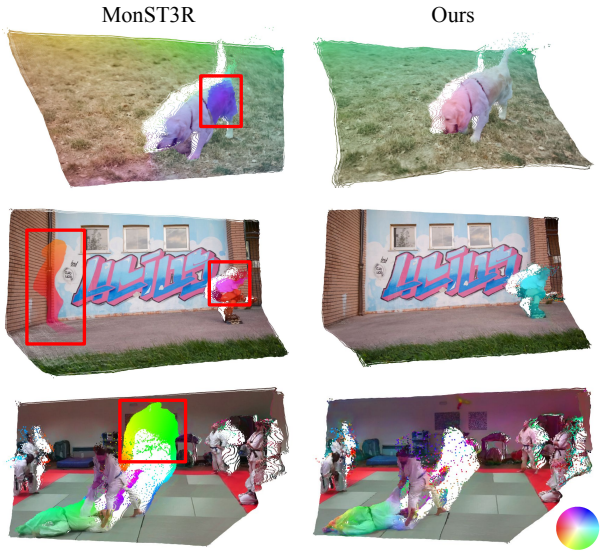

Figure 8. **Visualization of scene flow estimation results** for MonST3R (left) and our method (right). Red boxes highlight errors in MonST3R's predictions. **Row 1:** MonST3R exhibits unnatural artifacts and incorrect flow estimation. **Row 2:** Incorrect flow direction. **Row 3:** Poor handling of disocclusions due to MonST3R using 2D optical flow for warping. Our method consistently produces more accurate scene flow predictions across all cases.

MonST3R.

4.2. Dynamic reconstruction

We evaluate our method on dynamic reconstruction using a test set consisting of 250 held-out clips from our MOVi-G dataset, as well as on the public Kubric MOVi-F dataset and Waymo Open. These two variants of Kubric are fairly different: MOVi-G has complex camera motion and a mix of static and dynamic objects, whereas MOVi-F has simple linear motion and also a mix of dynamic and static objects with varying amounts of motion blur.

We primarily compare against MonST3R [80], which is trained with dynamic content but only predicts two of our four point maps (Fig. 3). To approximate the remaining point maps, MonST3R needs to use the pixel associations given by optical flow prediction, which is limited for pixels that are visible in both images.

Dynamic Point Maps prediction. We first show that we can predict our dynamic point maps well by using a fine-tuned version of DUSt3R. To this end, let $P, \hat{P} \in \mathbb{R}^{3\times HW}$ be, respectively, ground-truth and predicted maps. Furthermore, let $M \subset \{1, \dots, HW\}$ be a mask representing a subset of image pixels. Before computing the loss, we normalize the stacking P of point maps $P_1(t_1, \pi_1)$ and $P_2(t_2, \pi_1)$ by the median 2-norm of its valid points $Z_{\text{med}} = \text{median}(\{\|P_{:,i}\|_2 \mid i \in M\})$. We do the same for $\hat{P}$. Then, the *relative error* is

$$L_{\text{rel}}(\hat{P}, P|M, s) = \frac{1}{|M|} \sum_{i\in M} \frac{\|\hat{P}_{:,i} - P_{:,i}\|}{\|P_{:,i}\|}.$$

Table 3 report this metric for the four point map predictions. Our method outperforms MonST3R both on synthetic Kubric and real-world Waymo datasets. On the challenging Kubric-G dataset, the performance of MonST3R significantly degrades when making cross-time predictions ($P_2(t_1)$ and $P_1(t_2)$), highlighting the effectiveness of our representation for dynamic reconstruction.

Scene Flow. DPM allows us to infer 3D point correspondences, or dense scene flow. We assess this capability using standard metrics, described in the Appendix B.2.

Table 4 presents the quantitative evaluation of 3D End-Point Error (EPE), defined as the average Euclidean distance between predicted and ground-truth 3D displacement vectors, for Scene Flow (overall 3D motion including camera movement) and Object Flow (object-only motion from a fixed viewpoint) Evaluations are conducted on the Kubric-F, Kubric-G, and Waymo datasets for three methods: MonST3R [80] and RAFT-3D [56], a scene flow method.

Notably, RAFT-3D leverages ground truth depth (RGBD input), giving it an inherent advantage. Despite this, our RGB-only method achieves comparable performance to RAFT-3D on Kubric-F and surpasses it significantly on Kubric-G and Waymo. We further observe that RAFT-3D,

being a specialized Scene Flow model, cannot perform Object Flow estimation—a closely related task—which highlights the greater flexibility and broader applicability of our method. Compared to MonST3R, on average, our approach achieves a lower error across all datasets.

Figure 8 qualitatively demonstrates scene flow estimation, highlighting errors by MonST3R with red boxes. Specifically, Row 1 shows ghosting artifacts and incorrect motion predictions; Row 2 illustrates incorrect flow direction; and Row 3 demonstrates MonST3R's poor handling of disocclusions due to reliance on 2D optical flow-based warping.

Object tracking. We measure the performance of tracking dynamic objects by estimating the relative pose transformation between times t_0 and t_1. We estimate the relative rotation R and translation t that align $P_1(t_1, \pi_1)$ to $P_1(t_2, \pi_1)$ using the Umeyama algorithm [59]. We obtain the reference transformations $\hat{R}$ and $\hat{t}$ from ground truth bounding boxes and compute the geodesic distance $\text{dist}(R, \hat{R}) = \arccos((\text{tr}(R^T \hat{R}) - 1)/2)$ and the L_2 distance between object center translations $\text{dist}(t, \hat{t}) = \|t - \hat{t}\|_2$.

Table 3 shows that MonST3R fails to accurately predict future object locations because it lacks training for cross-time predictions and depends only on optical flow for correspondences. While rotation errors remain high for both methods during large object motions, our approach reduces these errors by 40% compared to MonST3R.

Camera pose. We report in Table 5 pose evaluation results in the TUM-dynamics dataset using the same optimisation protocol as in MonST3R. We show improved performance over MonST3R and comparable results to CasualSAM.

4.3. Qualitative results

We test DPM qualitatively on video clips recorded with a consumer camera by running the network on image pairs (see also the supplemental material). Figure 1 shows dynamic reconstructions for clips in which both an object and the camera are moving. We observe that our method can disentangle the two motions, particularly by removing the camera motion. Figure 6 visualizes the recovered camera poses and dynamic point cloud on a DAVIS sequence. We also visualize the results of applying DPM to solve downstream tasks in other examples, such as motion segmentation in Fig. 4 and point correspondence in Fig. 5.

Method	ATE ↓	RPE trans ↓	RPE rot ↓
Robust-CVD	0.189	0.071	3.681
CasualSAM	**0.045**	<u>0.020</u>	<u>0.841</u>
DUST3R w/ mask†	0.127	0.062	3.099
MonST3R	0.074	0.019	0.905
DPM	<u>0.056</u>	**0.014**	**0.836**

Table 5. Comparison of pose metrics on TUM-dynamics.

5. Downstream Applications

We now illustrate the ability of DPMs to easily solve a variety of 4D reconstruction tasks. We refer the reader to Appendix A.4 for formal derivations.

Firstly, we consider *4D reconstruction*. The point maps $P_i(t_i, \pi_1)$ provide a 3D reconstruction of each image I_i in a sequence where the viewpoint is fixed to π_1. In this way, we obtain a 4D 'animation' of the scene where the intrinsic motion of the objects is isolated from the camera motion. An example is shown in Fig. 6.

Secondly, we consider *motion segmentation*. Given two (or more) images I_1 and I_2, we can simply compare point maps $P_1(t_1, \pi_1)$ and $P_1(t_2, \pi_2)$ to determine which 3D points from image I_1 have moved (independently of the camera motion), and correspondingly with I_2. By thresholding the difference between the two sets of points, we can segment motion, as shown in Fig. 4.

Thirdly, we consider *point correspondence*. A key property of DPMs is their ability to restore invariance of the point maps despite in-scene camera motion, which allows matching them. Namely, given a pixel $\boldsymbol{u}_1$ in image I_1, we can describe it via the 3D point $\boldsymbol{p}_1 = P_1(t_1, \pi_1)(\boldsymbol{u}_1)$. To find the matching point in image I_2, we compare $\boldsymbol{p}_1$ to all points $\boldsymbol{p}_2(\boldsymbol{u}_2) = P_1(t_2, \pi_2)(\boldsymbol{u}_2)$ and take the pixel $\boldsymbol{u}_2$ that minimizes the distance $\|\boldsymbol{p}_2(\boldsymbol{u}_2) - \boldsymbol{p}_1\|$ as the match. An example of this process is shown in Fig. 5.

Fourthly, we consider *camera tracking*. One way to recover the camera motion while ignoring dynamic distractors is to match point clouds $P_1(t_1, \pi_1)$ and $P_2(t_1, \pi_2)$ using Procrustes analysis, where the time t_1 is fixed, but the camera π_i varies. This effectively reduces the problem to standard camera tracking in a static scene by undoing the passing of time. An example is shown in Fig. 6.

Lastly, we consider *object tracking*. In this case, given a mask M for the object of interest in image I_1, one can simply observe points $M \odot P(t_i, \pi_1)$ to infer the motion of the object independently of the camera motion. If the object is rigid, it is straightforward to fit a rigid transformation to these points. An example is shown in Fig. 7.

6. Conclusions

We have presented Dynamic Point Maps, a generalization of point maps that, by incorporating both viewpoint and time invariance, facilitates solving several challenging 4D reconstruction tasks in a single neural network evaluation. We show that this representation can help to developing more powerful 3D/4D vision models. In particular, when used to extend DUSt3R, it consistently matches or surpasses state-of-the-art in video depth estimation, dynamic point cloud reconstruction, and scene flow estimation.

Acknowledgments. The authors of this work were supported by ERC 101001212-UNION.

References

[1] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building rome in a day. In *Proc. ICCV*, 2009. 3

[2] Ijaz Akhter, Yaser Sheikh, Sohaib Khan, and T. Kanade. Nonrigid structure from motion in trajectory space. In *Proc. NeurIPS*, 2008. 3

[3] Ijaz Akhter, Yaser Sheikh, Sohaib Khan, and T. Kanade. Trajectory Space: A dual representation for nonrigid structure from motion. *arXiv*, 2011. 3

[4] Christian Bailer, Bertram Taetz, and Didier Stricker. Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation. In *Proc. ICCV*, 2015. 3

[5] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: speeded up robust features. In *Proc. ECCV*, 2006. 3

[6] Christoph Bregler, Aaron Hertzmann, and Henning Biermann. Recovering non-rigid 3D shape from image streams. In *Proc. CVPR*, 2000. 3

[7] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *Proc. ECCV*, 2004. 3

[8] Thomas Brox, Christoph Bregler, and Jitendra Malik. Large displacement optical flow. In *Proc. CVPR*, 2009.

[9] Andrés Bruhn, Joachim Weickert, and Christoph Schnörr. Lucas/kanade meets horn/schunck: Combining local and global optic flow methods. *IJCV*, 61(3), 2005. 3

[10] Hongkai Chen, Zixin Luo, Jiahui Zhang, Lei Zhou, Xuyang Bai, Zeyu Hu, Chiew-Lan Tai, and Long Quan. Learning to match features with seeded graph matching network. In *Proc. CVPR*, 2021. 3

[11] Mingyu Chen, G. Al-Regib, and B. Juang. Trajectory triangulation: 3D motion reconstruction with $\ell 1$ optimization. In *Proc. ICASSP*, 2011. 3

[12] DeTone Daniel, Malisiewicz Tomasz, and Rabinovich Andrew. SuperPoint: self-supervised interest point detection and description. *arXiv*, 1712.07629, 2017. 3

[13] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. FlowNet: Learning optical flow with convolutional networks. In *Proc. ICCV*, 2015. 3, 5

[14] Yilun Du, Yinan Zhang, Hong-Xing Yu, Joshua B. Tenenbaum, and Jiajun Wu. Neural radiance flow for 4d view synthesis and video processing. In *Proc. ICCV*, 2021. 3

[15] O. D. Faugeras, F. Lustman, and G. Toscani. Motion and structure from motion from point and line matches. In *Proc. ICCV*, 1987. 3

[16] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. Towards internet-scale multi-view stereo. In *Proc. CVPR*. IEEE, 2010. 3

[17] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proc. ICCV*, 2021. 3

[18] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam Laradji, Hsueh-Ti (Derek) Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, Cengiz Oztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, and Andrea Tagliasacchi. Kubric: a scalable dataset generator. In *Proc. CVPR*, 2022. 5

[19] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000. 3

[20] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow: A retrospective. *Artif. Intell.*, 1-2, 1993. 3

[21] Zhaoyang Huang, Xiaoyu Shi, Chao Zhang, Qiang Wang, Ka Chun Cheung, Hongwei Qin, Jifeng Dai, and Hongsheng Li. FlowFormer: a transformer architecture for optical flow. In *Proc. ECCV*, 2022. 3

[22] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks. *arXiv preprint arXiv:1612.01925*, 2016. 3

[23] J. Janai, F. Güney, A. Ranjan, M. Black, and A. Geiger. Unsupervised learning of multi-frame optical flow with occlusions. In *Proc. ECCV*, 2018. 3

[24] Shihao Jiang, Dylan Campbell, Yao Lu, Hongdong Li, and Richard Hartley. Learning to estimate hidden motions with global motion aggregation. In *Proc. ICCV*, 2021. 3

[25] Wei Jiang, Eduard Trulls, Jan Hosang, Andrea Tagliasacchi, and Kwang Moo Yi. COTR: correspondence transformer for matching across images. *CoRR*, abs/2103.14167, 2021. 3

[26] Linyi Jin, Richard Tucker, Zhengqi Li, David Fouhey, Noah Snavely, and Aleksander Holynski. Stereo4d: Learning how things move in 3d from internet stereo videos. In *arXiv preprint*, 2024. 3

[27] Nikita Karaev, Ignacio Rocco, Ben Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. CoTracker: It is better to track together. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2024. 5

[28] Alex Kendall and Yarin Gal. What uncertainties do we need in Bayesian deep learning for computer vision? *Proc. NeurIPS*, 2017. 5

[29] Skanda Koppula, Ignacio Rocco, Yi Yang, Joe Heyward, João Carreira, Andrew Zisserman, Gabriel Brostow, and Carl Doersch. TAPVid-3D: a benchmark for tracking any point in 3D. *arXiv*, 2407.05921, 2024. 3

[30] Suryansh Kumar, Yuchao Dai, and Hongdong Li. Monocular dense 3D reconstruction of a complex dynamic scene from two perspective frames. In *Proc. ICCV*, 2017. 3

[31] Jiahui Lei, Yijia Weng, Adam Harley, Leonidas Guibas, and Kostas Daniilidis. MoSca: dynamic gaussian fusion from casual videos via 4d motion scaffolds. *arXiv*, 2405.17421, 2024. 2

[32] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proc. CVPR*, 2021. 3

[33] Zhengqi Li, Qianqian Wang, Forrester Cole, Richard Tucker, and Noah Snavely. DynIBaR: Neural dynamic image-based rendering. In *Proc. CVPR*, 2023. 3

[34] Philipp Lindenberger, Paul-Edouard Sarlin, and Marc Pollefeys. LightGlue: local feature matching at light speed. In *Proc. ICCV*, 2023. 3

[35] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proc. of the 7th International Joint Conference on Artificial Intelligence*, 1981. 3

[36] David Novotný, Diane Larlus, and Andrea Vedaldi. Learning 3D object categories by looking around them. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2017. 5

[37] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B. Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *iccv*, 2021. 3

[38] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B. Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. Hypernerf: a higher-dimensional representation for topologically varying neural radiance fields. *Proc. SIGGRAPH*, 40(6), 2021. 3

[39] Chiara Plizzari, Shubham Goel, Toby Perrett, Jacob Chalk, Angjoo Kanazawa, and Dima Damen. Spatial cognition from egocentric video: Out of sight, not out of mind. *arXiv*, 2404.05072, 2024. 2

[40] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural radiance fields for dynamic scenes. In *Proc. CVPR*, 2021. 3

[41] Rene Ranftl, Vibhav Vineet, Qifeng Chen, and Vladlen Koltun. Dense monocular depth estimation in complex dynamic scenes. In *Proc. CVPR*, 2016. 3

[42] Zhile Ren, Orazio Gallo, Deqing Sun, Ming-Hsuan Yang, Erik B. Sudderth, and Jan Kautz. A simple and effective fusion approach for multi-frame optical flow estimation. In *Proc. ECCV Workshop*, 2018. 3

[43] Chris Russell, Rui Yu, and Lourdes Agapito. Video pop-up: Monocular 3D reconstruction of dynamic scenes. In *Proc. ECCV*, 2014. 3

[44] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. SuperGlue: learning feature matching with graph neural networks. In *Proc. CVPR*, 2020. 3

[45] Frederik Schaffalitzky and Andrew Zisserman. Multi-view matching for unordered image sets, or "How do I organize my holiday snaps?". In *Proc. ECCV*, 2002. 3

[46] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proc. CVPR*, 2016. 3

[47] Xiaoyu Shi, Zhaoyang Huang, Weikang Bian, Dasong Li, Manyuan Zhang, Ka Chun Cheung, Simon See, Hongwei Qin, Jifeng Dai, and Hongsheng Li. VideoFlow: exploiting temporal cues for multi-frame optical flow estimation. In *Proc. ICCV*, 2023. 3

[48] Xiaoyu Shi, Zhaoyang Huang, Dasong Li, Manyuan Zhang, Ka Chun Cheung, Simon See, Hongwei Qin, Jifeng Dai, and Hongsheng Li. FlowFormer++: masked cost volume autoencoding for pretraining optical flow estimation. In *Proc. CVPR*, 2023. 3

[49] Yan Shi, Jun-Xiong Cai, Yoli Shavit, Tai-Jiang Mu, Wensen Feng, and Kai Zhang. ClusterGNN: cluster-based coarse-to-fine graph neural network for efficient feature matching. In *Proc. CVPR*, 2022. 3

[50] 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. *arXiv*, 2406.18717, 2024. 2

[51] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *Proc. CVPR*, 2018. 3

[52] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 5

[53] Zachary Teed and Jia Deng. DeepV2D: video to depth with differentiable structure from motion. In *Proc. ICLR*, 2020. 3

[54] Zachary Teed and Jia Deng. RAFT: recurrent all-pairs field transforms for optical flow. In *Proc. ECCV*, 2020. 3, 6

[55] Zachary Teed and Jia Deng. DROID-SLAM: deep visual SLAM for monocular, stereo, and RGB-D cameras. In *Proc. NeurIPS*, 2021. 3

[56] Zachary Teed and Jia Deng. RAFT-3D: scene flow using rigid-motion embeddings. In *Proc. CVPR*, 2021. 3, 7

[57] L. Torresani, A. Hertzmann, and C. Bregler. Learning non-rigid 3D shape from 2D motion. In *Proc. NeurIPS*, 2004. 3

[58] MJ Tyszkiewicz, P Fua, and E Trulls. DISK: learning local features with policy gradient. In *Proc. NeurIPS*, 2020. 3

[59] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(4), 1991. 8

[60] Benjamin Ummenhofer, Huizhong Zhou, Jonas Uhrig, Nikolaus Mayer, Eddy Ilg, Alexey Dosovitskiy, and Thomas Brox. DeMoN: depth and motion network for learning monocular stereo. In *Proc. CVPR*, 2017. 3

[61] Jack Valmadre and Simon Lucey. General trajectory prior for non-rigid reconstruction. In *Proc. CVPR*, 2012. 3

[62] Bo Wang, Jian Li, Yang Yu, Li Liu, Zhenping Sun, and Dewen Hu. SceneTracker: long-term scene flow estimation network. *arXiv*, 2403.19924, 2024. 3

[63] Chaoyang Wang, Ben Eckart, Simon Lucey, and Orazio Gallo. Neural trajectory fields for dynamic novel view synthesis. *arXiv.cs*, abs/2105.05994, 2021. 3

[64] He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J. Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. In *Proc. CVPR*, pages 2642–2651, 2019. 5

[65] Jianyuan Wang, Yiran Zhong, Yuchao Dai, Stan Birchfield, Kaihao Zhang, Nikolai Smolyanskiy, and Hongdong Li. Deep two-view structure-from-motion revisited. In *Proc. CVPR*, 2021. 3

[66] Jianyuan Wang, Nikita Karaev, Christian Rupprecht, and David Novotny. VGGSfM: visual geometry grounded deep structure from motion. In *Proc. CVPR*, 2024. 3

[67] Qianqian Wang, Vickie Ye, Hang Gao, Jake Austin, Zhengqi Li, and Angjoo Kanazawa. Shape of motion: 4D reconstruction from a single video. *arXiv*, 2407.13764, 2024. 3

[68] Qianqian Wang, Yifei Zhang, Aleksander Holynski, Alexei A Efros, and Angjoo Kanazawa. Continuous 3d perception model with persistent state. In *arXiv preprint*, 2025. 3

[69] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. DUSt3R: Geometric 3D vision made easy. In *Proc. CVPR*, 2024. 1, 3, 5

[70] Xingkui Wei, Yinda Zhang, Zhuwen Li, Yanwei Fu, and Xiangyang Xue. DeepSFM: structure from motion via deep bundle adjustment. In *Proc. ECCV*, 2020. 3

[71] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4D gaussian splatting for real-time dynamic scene rendering. In *Proc. CVPR*, 2023. 3

[72] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *Proc. CVPR*, 2021. 3

[73] Yuxi Xiao, Qianqian Wang, Shangzhan Zhang, Nan Xue, Sida Peng, Yujun Shen, and Xiaowei Zhou. SpatialTracker: tracking any 2d pixels in 3d space. *arXiv*, 2404.04319, 2024. 3

[74] Haofei Xu, Jing Zhang, Jianfei Cai, Hamid Rezatofighi, and Dacheng Tao. GMFlow: learning optical flow via global matching. In *Proc. CVPR*, 2022. 3

[75] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In *Proc. CVPR*, 2024. 5

[76] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything V2. *arXiv*, 2406.09414, 2024. 5

[77] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. In *Proc. CVPR*, 2024. 3

[78] Zeyu Yang, Hongye Yang, Zijie Pan, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. In *Proc. ICLR*, 2024. 3

[79] Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. LIFT: learned invariant feature transform. In *Proc. ECCV*, 2016. 3

[80] Junyi Zhang, Charles Herrmann, Junhwa Hur, Varun Jampani, Trevor Darrell, Forrester Cole, Deqing Sun, and Ming-Hsuan Yang. MonST3R: a simple approach for estimating geometry in the presence of motion. *arXiv*, 2410.03825, 2024. 2, 3, 4, 6, 7

Dynamic Point Maps: A Versatile Representation for Dynamic 3D Reconstruction

Supplementary Material

A. Theory of Dynamic Point Maps

We discuss more formally how Dynamic Point Maps is defined and how it can be used to solve various 4D reconstruction tasks.

A.1. Monocular point maps

We represent the image I as a $3 \times HW$ matrix by stacking the spatial dimensions, where 3 is the number of color channels. We assume that the image is taken by a pinhole camera. Hence, a 3D point $\boldsymbol{p} = (x, y, z) \in \mathbb{R}^3$ expressed in the reference frame of this camera projects to the image pixel $\boldsymbol{u} = (u_x, u_y, 1)$ such that

$$\boldsymbol{u}\lambda = K\boldsymbol{p},$$

where $K \in \mathbb{R}^{3\times3}$ is the camera's calibration matrix containing the camera's intrinsic parameters, and $\lambda > 0$ is the depth of the point.

The point map P is the collection of 3D points corresponding to each pixel, which we can write as a matrix $P \in \mathbb{R}^{3\times HW}$. Denote by $U \in \mathbb{R}^3$ the grid of pixels in homogeneous coordinates (this matrix is fixed). Then we can write

$$U \operatorname{diag} \Lambda = KP,$$

where $\Lambda \in \mathbb{R}_+^{HW}$ contains the depth values.

For monocular prediction, we can task a neural network Φ with mapping the image I to the corresponding 3D point cloud P, i.e., $P = \Phi(I)$. This problem is related to monocular depth estimation, in which a neural network $\Lambda = \Phi_{\text{depth}}(I)$ is tasked with associating pixels to depth values, but it provides more information. In fact, to reconstruct the 3D points P from the depth Λ, we also require knowledge of the camera intrinsics K, so that the 3D points can be recovered as $P = K^{-1}U \operatorname{diag}(\Lambda) = K^{-1}U \operatorname{diag}(\Phi_{\text{depth}}(I))$. Conversely, knowledge of the point map P allows us to *infer* depth and intrinsics by solving the equation $U\Lambda = KP = K\Phi(I)$ for Λ and K.

A.2. Binocular point maps (DUSt3R)

Next, we extend the case above to consider a pair of images I_1 and I_2. Each image is taken by a camera with different intrinsics K_1 and K_2 and, most importantly, different viewpoints π_1 and π_2.

Let the symbols $\boldsymbol{p}(\pi_1)$ and $\boldsymbol{p}(\pi_2)$ denote the coordinates of a certain 3D point $\boldsymbol{p}$ expressed in the reference frames of the first and second cameras, respectively. Following DUSt3R, we task a neural network Φ with predicting the pair of point clouds $(P_1(\pi_1), P_2(\pi_1))$ from the pair of images (I_1, I_2):

$$(P_1(\pi_1), P_2(\pi_1)) = \Phi(I_1, I_2). \tag{3}$$

As above, knowledge of $P_1(\pi_1)$ allows us to recover the intrinsics K_1 of the first camera from

$$U_1\Lambda_1 = K_1P_1(\pi_1).$$

K_2 can be recovered by calling the network a second time with the two arguments swapped. More interestingly, however, the second point cloud $P_2(\pi_1)$ contains the 3D points that correspond to the pixels of the second image I_2, but *still expressed in the reference frame π_1 of the first camera*. This means that the extrinsics K_2 and the relative rigid motion $(R^*, \boldsymbol{t}^*) \in SE(3)$ from the second camera to the first can be recovered from the analogous equation

$$U_2\Lambda_2 = K_2\,(R^*)^{-1}(P_2(\pi_1) - \boldsymbol{t}^*).$$

Later, we will discuss an alternative method based on matching point clouds.

The point maps also encode correspondences between images I_1 and I_2, as it is immediate to determine which 3D points are the same by checking for equality of coordinates, given that these are expressed in the same reference frame. Specifically, to find out which pixel $\boldsymbol{u}_i$ in image I_1 is the best match for pixel $\boldsymbol{u}_j$ in image I_2, one simply minimizes the distance between the corresponding 3D points $\|[P_1(\pi_1)]_{:,i} - [P_2(\pi_1)]_{:,j}\|$, which is meaningful as they are both expressed in the same reference frame π_1.

It is useful to express these correspondences using a matrix notation. Hence, given two set of d-dimensional point descriptors $A^{d\times HW}$ and $B^{d\times HW}$, we define

$$C(A, B) = \operatorname*{argmin}_C \|A - BC\| \tag{4}$$

where $C \in \{0, 1\}^{HW\times HW}$ is a square binary matrix with exactly one unitary entry along each row, i.e., $C\mathbf{1} = \mathbf{1}$. Hence, the correspondence matrix from image I_2 back to image I_1 is simply:

$$C_{12} = C(P_1(\pi_1), P_2(\pi_2)).$$

A.3. Dynamic point maps

The arguments above break if physical points can move over time and if the two shots I_1 and I_2 are not taken simultaneously. In this case, a physical point $\boldsymbol{p}$ will not, in general, be found at the same location in the two shots. Hence, compensating for the camera viewpoint is insufficient to establish correspondences.

Our solution is to parametrize points with respect to both *viewpoint* and *time*. Images I_1 and I_2 come with timestamps t_1 and t_2, so the coordinates of a physical point $\boldsymbol{p}$ are a function of both viewpoint and time, i.e., $\boldsymbol{p}(t_i, \pi_j)$.

Given the two images, we have four possible combinations of these reference parameters. Two, i.e., (t_1, π_1) and (t_2, π_2), correspond to the viewpoint and timestamp of images I_1 and I_2, respectively. The other two, i.e., (t_1, π_2) and (t_2, π_1), correspond to swapping the viewpoint and timestamp between the two images.

Because there are two point clouds P_1 and P_2, the first corresponding to the pixels in image I_1 and the second to those in image I_2, and each is expressible in any of these references, there are a total of eight different outputs. Since the roles of images I_1 and I_2 are symmetric, it suffices to task the network Φ with predicting four quantities, with the other four obtained by swapping the inputs. These four predictions are:

$$(P_1(t_1, \pi_1), P_1(t_2, \pi_1), P_2(t_1, \pi_1), P_2(t_2, \pi_1)) = \Phi(I_1, I_2). \tag{5}$$

Note that all these predictions refer the point clouds to the reference frame π_1 of the first camera. We obtain the four complementary predictions for π_2 by swapping the network's inputs.

Special cases. If the images are taken at the same time, then $t_1 = t_2$, the predictions $P_1(t_1, \pi_1) = P_1(t_2, \pi_1)$ and $P_2(t_1, \pi_1) = P_2(t_2, \pi_1)$ collapse, and model Eq. (5) reduces to Eq. (3) explored by DUSt3R. Furthermore, if $\pi_1 = \pi_2$ as well, then this model reduces to monocular point prediction which, as we have seen above, is related to but more informative than depth prediction.

A.4. Using DPM to solve 3D and 4D tasks

In this section, we show how the output of the network Φ can be used to solve a number of basic 3D and 4D problems.

Recovering the camera intrinsics and extrinsics. The camera intrinsics K_1 can be recovered immediately from equation $U_1 \Lambda_1 = K_1 P_1(t_1, \pi_1)$. The camera extrinsics K_2 and the relative rigid motion $(R^*, \boldsymbol{t}^*) \in SE(3)$ from the second camera to the first can be recovered from the analogous equation $U_2 \Lambda_2 = K_2 \, (R^*)^{-1} (P_2(t_1, \pi_1) - \boldsymbol{t}^*)$. These are the same equations for static point maps, which is possible because we are fixing the time to t_1.

Performing motion segmentation. To tell whether pixel $\boldsymbol{u}_i$ in image I_1 corresponds to a physical points that moves with respect to the camera, we can simply check if its coordinates $[P_1(t_1, \pi_1)]_{:,i}$ and $[P_1(t_2, \pi_1)]_{:,i}$ differ or not (see Fig. 4). We introduce the following compact notation: given point descriptors $A, B \in \mathbb{R}^{d \times HW}$, we define the mask $M(A, B) \in \{0, 1\}^{HW}$ as the vector $[M(A, B)]_i = \chi(\|A_{:,i} - B_{:,i}\| > \epsilon)$, where $\epsilon \geq 0$ is a threshold. Then, the motion masks in images I_1 and I_2 are:

$$M_1 = M(P_1(t_1, \pi_1), P_1(t_2, \pi_1)),$$
$$M_2 = M(P_2(t_1, \pi_1), P_2(t_2, \pi_1)).$$

	Dataset	Motion?	$P_1(t_1, \pi_1)$	$P_2(t_1, \pi_1)$	$P_1(t_2, \pi_1)$	$P_2(t_2, \pi_1)$
(a)	Kubric	Yes	✓	✓	✓	✓
	Waymo	Yes	✓	✓	✓	✓
	PointOdyssey	Yes	✓	✓	✓	✓
(b)	Spring	Yes	✓	—	—	✓
(c)	ScanNet++	No	✓	✓	✓	✓
	BlendedMVS	No	✓	✓	✓	✓
	MegaDepth	No	✓	✓	✓	✓

Table 6. Training datasets fall into 3 groups. (a) Kubric, Waymo, and PointOdyssey contain dynamic scenes and provide annotations for all 4 point maps. (b) Spring only supervises *same-time* point maps. (c) Finally, ScanNet++, BlendedMVS and MegaDepth contain static scenes.

Obtaining point correspondences. Using Eq. (4), we can map each pixel in image I_2 to the corresponding pixel image I_1 via the correspondence matrix: $C_{12} = C(P_1(t_1, \pi_1), P_2(t_1, \pi_1))$. Note that this also works for dynamic points because the network registers both viewpoint and timestamp, see Fig. 5.

Reconstructing the camera motion. The relative rigid motion $(R^*, \boldsymbol{t}^*) \in SE(3)$ from the second camera to the first can be recovered from the matching points as

$$\operatorname*{argmin}_{R, \boldsymbol{t}} \|(P_1(t_1, \pi_1) C_{12} - R P_2(t_1, \pi_2) - \boldsymbol{t}) \operatorname{diag}(\bar{M}_2)\|,$$

where the subtraction by $\boldsymbol{t}$ is broadcast to all points and $\bar{M}_2 = \mathbf{1} - M_2$ masks out dynamic pixels, see Figure 6.

Reconstructing rigid object motion. If M is the mask of a certain object in image I_1, then its rigid motion with respect to the reference frame (t_1, π_1) can be recovered as (see Figure 7):

$$\operatorname*{argmin}_{R, \boldsymbol{t}} \|(P_1(t_2, \pi_1) - R P_1(t_1, \pi_1) - \boldsymbol{t}) \operatorname{diag}(M)\|.$$

B. Experimental details

B.1. Dataset categories

The different datasets used for training and their type of supervision signal, is shown in Table 6.

B.2. Scene and object flow metrics

Scene Flow and Object Flow are defined based on the 3D displacement of points in a scene, with and without camera motion.

- **Scene Flow (SF)** captures the full 3D motion of points, incorporating both object and camera movement:

  - **Forward Scene Flow (SF-F):** $P_1(t_2, \pi_2) - P_1(t_1, \pi_1)$ describing how points at t_1 move to t_2 under a potentially moving camera.
  - **Backward Scene Flow (SF-B):** $P_2(t_1, \pi_1) - P_2(t_2, \pi_2)$ mapping how points at t_2 correspond back to t_1.
- **Object Flow (OF)** isolates object motion by assuming a fixed camera:
  - **Forward Object Flow (OF-F):** $P_1(t_2, \pi_1) - P_1(t_1, \pi_1)$ capturing how points move between frames when viewed from the same camera pose.
  - **Backward Object Flow (OF-B):** $P_2(t_1, \pi_1) - P_2(t_2, \pi_1)$ tracing the movement of points back in time while keeping the viewpoint unchanged.

C. Original MonST3R

Table 3 and Table 4 with the the original MonST3R checkpoint are shown in Table 7 and Table 8 respectively.

		L_{rel}				**Object Pose Error**	
Dataset	Method	$P_1(t_1)$	$P_2(t_1)$	$P_1(t_2)$	$P_2(t_2)$	RPE rot	RPE trans
Kub.-F	MonST3R	0.209	0.275	0.394	0.201	56.1	0.504
	Ours	**0.041**	**0.047**	**0.049**	**0.035**	**33.7**	**0.053**
Kub.-G	MonST3R	0.163	0.265	0.346	0.178	N/A	
	Ours	**0.057**	**0.071**	**0.079**	**0.058**		
Waymo	MonST3R	0.197	0.221	0.249	0.178	N/A	
	Ours	**0.068**	**0.065**	**0.067**	**0.065**		

Table 7. Dynamic reconstruction.

			Scene Flow		**Object Flow**	
Dataset	Method	Input	Forward	Backward	Forward	Backward
Kub.-F	MonST3R	RGB	0.321	0.241	0.334	0.215
	RAFT-3D	RGBD	**0.051**	**0.054**	N/A	N/A
	Ours	RGB	0.081	0.071	**0.033**	**0.029**
Kub.-G	MonST3R	RGB	0.334	0.279	0.310	0.265
	RAFT-3D	RGBD	4.067	4.084	N/A	N/A
	Ours	RGB	**0.104**	**0.106**	**0.059**	**0.050**
Waymo	MonST3R	RGB	0.161	0.135	0.108	0.102
	RAFT-3D	RGBD	0.150	0.145	N/A	N/A
	Ours	RGB	**0.051**	**0.053**	**0.020**	**0.020**

Table 8. 3D End-Point Error (EPE) for Scene Flow and Object Flow.