

# 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  $\mathbf{p} = (x, y, z) \in \mathbb{R}^3$  expressed in the reference frame of this camera projects to the image pixel  $\mathbf{u} = (u_x, u_y, 1)$  such that

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

where  $K \in \mathbb{R}^{3 \times 3}$  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

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

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

For monocular prediction, we can task a neural network  $\Phi$  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  $\Lambda$ , we also require knowledge of the camera intrinsics  $K$ , so that the 3D points can be recovered as  $P = K^{-1}U \text{diag}(\Lambda) = K^{-1}U \text{diag}(\Phi_{\text{depth}}(I))$ . 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  $\Lambda$  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  $\pi_1$  and  $\pi_2$ .

Let the symbols  $\mathbf{p}(\pi_1)$  and  $\mathbf{p}(\pi_2)$  denote the coordinates of a certain 3D point  $\mathbf{p}$  expressed in the reference frames of the first and second cameras, respectively. Following DUST3R, we task a neural network  $\Phi$  with predicting the pair

of point clouds  $(P_1(\pi_1), P_2(\pi_1))$  from the pair of images  $(I_1, I_2)$ :

$$(P_1(\pi_1), P_2(\pi_1)) = \Phi(I_1, I_2). \quad (3)$$

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  $\pi_1$  of the first camera*. This means that the extrinsics  $K_2$  and the relative rigid motion  $(R^*, \mathbf{t}^*) \in SE(3)$  from the second camera to the first can be recovered from the analogous equation

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

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  $\mathbf{u}_i$  in image  $I_1$  is the best match for pixel  $\mathbf{u}_j$  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  $\pi_1$ .

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

$$C(A, B) = \underset{C}{\operatorname{argmin}} \|A - BC\| \quad (4)$$

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  $\mathbf{p}$  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  $\mathbf{p}$  are a function of both viewpoint and time, i.e.,  $\mathbf{p}(t_i, \pi_j)$ .

Given the two images, we have four possible combinations of these reference parameters. Two, i.e.,  $(t_1, \pi_1)$  and  $(t_2, \pi_2)$ , correspond to the viewpoint and timestamp of images  $I_1$  and  $I_2$ , respectively. The other two, i.e.,  $(t_1, \pi_2)$  and  $(t_2, \pi_1)$ , 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  $\Phi$  with predicting four quantities, with the other four obtained by swapping the inputs. These four predictions are:

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

Note that all these predictions refer the point clouds to the reference frame  $\pi_1$  of the first camera. We obtain the four complementary predictions for  $\pi_2$  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  $\Phi$  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  $(R^*, \mathbf{t}^*) \in SE(3)$  from the second camera to the first can be recovered from the analogous equation  $U_2 \Lambda_2 = K_2 (R^*)^{-1} (P_2(t_1, \pi_1) - \mathbf{t}^*)$ . 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  $\mathbf{u}_i$  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 im-

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	✓	—	—	✓
ScanNet++	No	✓	✓	✓	✓
(c) 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.

ages  $I_1$  and  $I_2$  are:

$$\begin{aligned} M_1 &= M(P_1(t_1, \pi_1), P_1(t_2, \pi_1)), \\ M_2 &= M(P_2(t_1, \pi_1), P_2(t_2, \pi_1)). \end{aligned}$$

**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  $(R^*, \mathbf{t}^*) \in SE(3)$  from the second camera to the first can be recovered from the matching points as

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

where the subtraction by  $\mathbf{t}$  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  $(t_1, \pi_1)$  can be recovered as (see Figure 7):

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

## 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.

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

Table 7. Dynamic reconstruction.

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

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