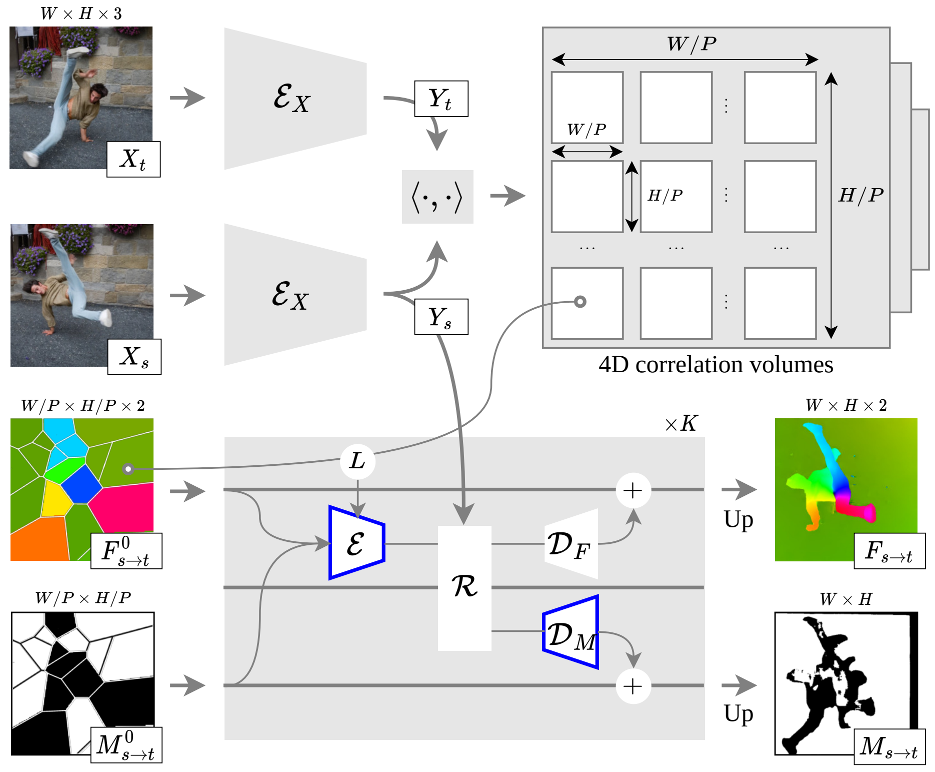
Dense Optical Tracking: Connecting the Dots

Appendix

A. Detailed architecture

Implementation details for our optical flow module are presented in Figure A. The two major differences with the original RAFT network architecture [60] are highlighted in the figure. First, we use a stride 1 instead of 2 in the first convolutional layer of the frame encoder ($\mathcal{E}_X$), such that the spatial resolution is decreased by a factor P=4 instead of P=8. We find that this has a significant impact on performance, see ablation studies in Table 4 of the paper and also additional results in Appendix D. Second, our approach not only predicts a dense flow field, but also a visibility mask. We thus adapt the encoder ($\mathcal{E}$) and add a new decoder ($\mathcal{D}_M$) to take into account this new modality in the refinement process.



(a) Overview.

Residual encoder			
Conv7 × 7	3	64	1
Res3 × 3	64	64	1
Res3 × 3	64	96	2
Res3 × 3	96	128	2
Conv1 × 1	128	256	1

(b) Frame encoder ($\mathcal{E}_X$).

Flow and mask encoder			
Conv7 × 7	2+1	128	1
Conv3 × 3	128	64	1
Correlation encoder			
Conv1 × 1	324	256	1
Conv3 × 3	256	192	1
Combination of outputs from both			
Conv3 × 3	64+192	125	1

(c) Joint flow and mask encoder ($\mathcal{E}$).

Convolutional gated recurrent unit			
GRU5 × 5	128+256	128	1

(d) Recurrent unit ($\mathcal{R}$).

Flow / mask decoders			
Conv3 × 3	128	256	1
Conv3 × 3	256	2/1	1
Operation	In. dim.	Out. dim.	Stride

(e) Flow and mask decoders ($\mathcal{D}_F$ / $\mathcal{D}_M$).

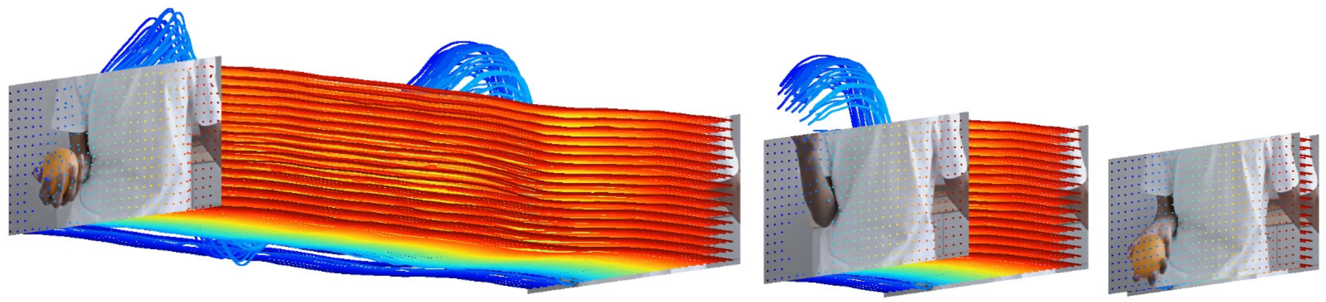
Figure A. **Optical flow module.** Our approach builds upon RAFT [60] and refines a dense flow field $F_{s\to t}$ and a visibility mask $M_{s\to t}$ from coarse estimates $F^0_{s\to t}$ and $M^0_{s\to t}$ using frames X_s and X_t. We first use a frame encoder ($\mathcal{E}_X$), implemented as a convolutional neural network with residual connections that extracts features Y_s and Y_t in $\mathbb{R}^{W/P \times H/P \times D}$ (P=4 and D=256 in practice) from input frames. We then compute the correlation between features for all pairs of source and target positions at different feature resolution levels, yielding a 4D correlation volume for each resolution level. More precisely, the size of the volume at the finest level is $W/P \times H/P \times W/P \times H/P$, and coarser levels are obtained by applying average pooling on the feature maps. The refinement process is composed of K iterations of the following operations (K=4 in practice): A look-up operation (L) in the cost volumes using the flow estimate from the previous iteration; The joint encoding ($\mathcal{E}$) of the flow, mask and correlation information; A recurrent unit ($\mathcal{R}$) which is an adaptation of a GRU cell [12] using convolutions; Two decoders ($\mathcal{D}_F$ and $\mathcal{D}_M$) to update the flow and the mask, respectively. We note that the model parameters are shared across all iterations. We finally upsample estimates from the K$^{\text{th}}$ iteration, using for each position at the fine resolution a linear combination of the values for the 9 nearest neighbors at the coarse resolution. Please refer to RAFT [60] for further details.

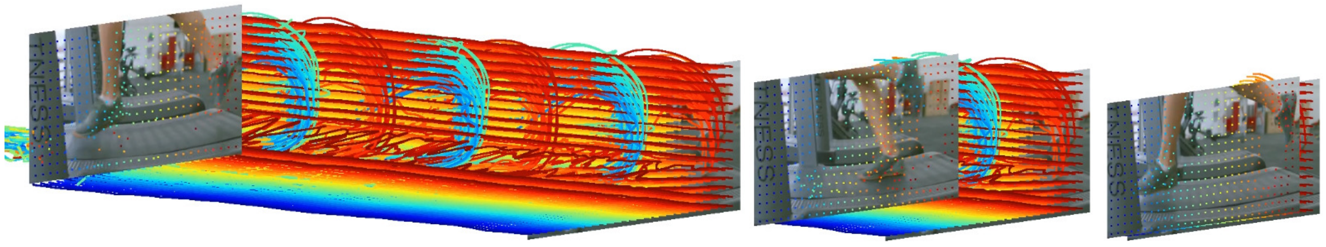
B. Interpolation strategy

Let N represent the number of initial tracks, and (H, W) denote the spatial resolution after interpolating these tracks. In a naive implementation of nearest-neighbor interpolation, the memory complexity grows quadratically in NHW, as it requires computing the distance between every pixel and every track. This can lead to significant memory issues, particularly when dealing with high resolutions. To address this challenge, we propose an efficient implementation relying on PyTorch3D [3] which comes with custom CUDA kernels, specifically optimized for this kind of operations.

C. Space-time visualizations of motion

Qualitative results in Figure C show that DOT produces accurate dense tracks even in challenging settings, such as when an object exits and re-enters the frame, when multiple objects occlude each other, or when facing extreme camera motions.


(a) Throwing an orange


(b) Running at the gym

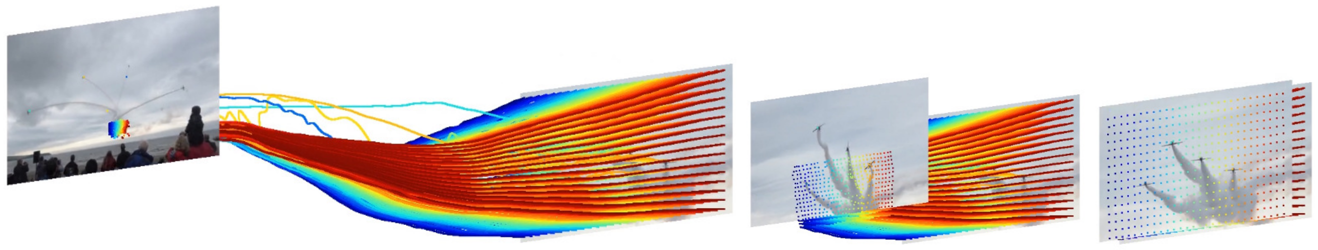

(c) Airplane parade

Figure C. **Space-time visualizations.** We track all pixels from the first frame of different videos, and show the trajectory for a subset of points, laid on a regular grid in the first frame. These trajectories are displayed in 3D, by using time as an additional dimension (progressing from right to left in the figure) alongside the two spatial dimensions. Each point on the grid is uniquely represented by a distinct color. Our method is able to track objects accurately, even when they go out of the frame multiple times like in video (a). DOT is able to cope with intra-object occlusions, as in video (b) where the left leg occludes the right one repeatedly. It is also robust to extreme camera motions, as illustrated in video (c) where points, initially covering the whole frame, are concentrated in a small regions in the last frame. Please zoom in for details. See also the videos in the project webpage.

[3] https://github.com/facebookresearch/pytorch3d/

D. Effect of the patch size on model performance and speed

We evaluate the impact of the patch size in the optical flow refinement module of our approach on the trade-off between performance and speed in Figure D. This involves setting the number N of initial point tracks to different values. We find that having a patch size of P=4 instead of P=8 is almost always beneficial, with a great boost in performance at the cost of a small reduction in inference speed. We note that a similar study conducted in CoTracker [33] yields the same conclusions. It is only for very small numbers of initial tracks that the larger patch size reaches better trade-offs, for example, with better performance in terms of EPE at a similar speed for $(P, N) = (8, 512)$ compared to $(P, N) = (4, 256)$.

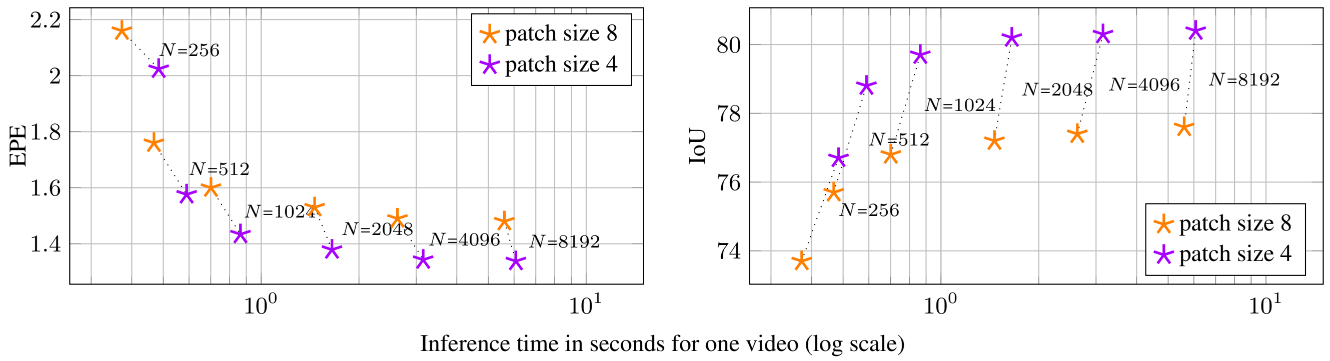


Figure D. **Effect of the patch size** on flow reconstruction, occlusion prediction and speed on the CVO (*Final*) dataset. We report the end point error (EPE) of flows, the intersection over union (IoU) of occluded regions, and the inference time (in seconds) when setting the number N of initial tracks to different values in $[256, 512, 1024, 2048, 4096, 8192]$.

E. Effect of the number of tracks on initial and final motion estimates

We show in Figure E some qualitative samples for different numbers of initial tracks. We see that final motion estimates produced by our method consistently improve over initial ones obtained through nearest-neighbor interpolation of tracks. Our refinement process not only enhances spatial smoothness but also produces motions that better match object edges like the border between the umbrella and the background or the part that sticks out in the middle.

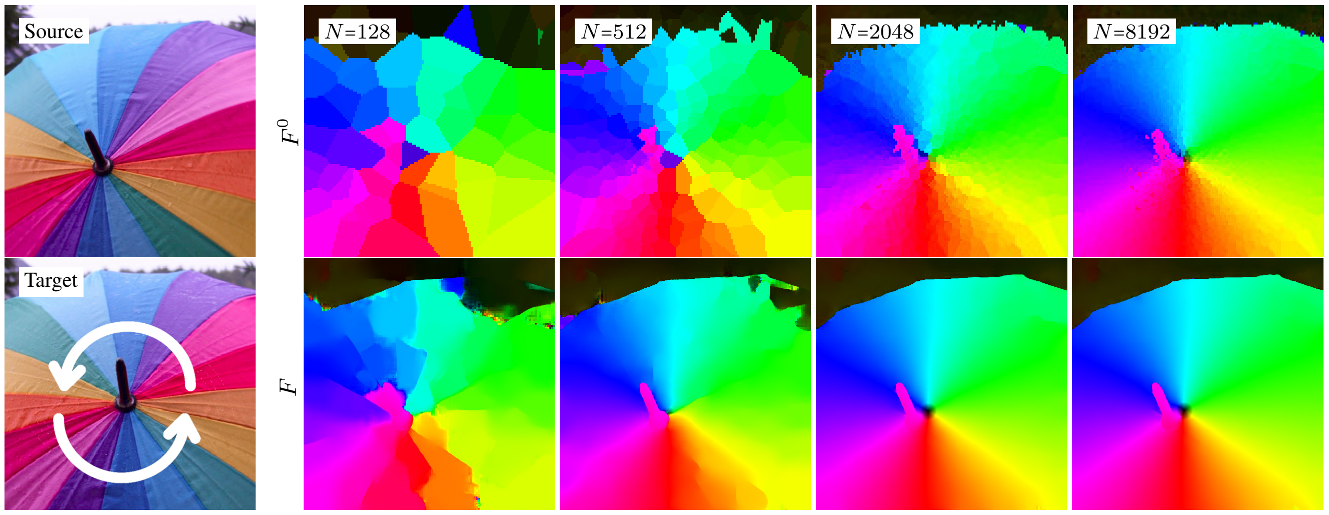


Figure E. **Effect of the number of tracks.** We visualize the initial flow between source and target frames (F^0) obtained by applying nearest-neighbor interpolation to N tracks extracted with an off-the-shelf model [33], and the final flow (F) refined by our approach. We represent motion directions using distinctive colors. We set the number N of initial tracks to different values in $[128, 512, 2048, 8192]$. The scene is composed of an umbrella undergoing a ~180° rotation (see the white arrows superimposed on the target frame) and a small translation, which is the reason for the nonzero flow associated with the tip of the umbrella.

F. Effect of the method used to extract sparse correspondences

We show in Figure F correspondences between source and target frames obtained by various methods. Our approach refines dense motions (optical flow and visibility mask) from these correspondences. Local feature matching methods, like SIFT [43] or SuperPoint [15] with LightGlue [42], excel in detecting salient feature points but struggle with textureless objects (*e.g.*, the red pot) or regions with motion blur (*e.g.*, the shoe). The resulting flow for these objects is often far from the ground truth. Moreover, as these methods only associate visible points, they offer limited assistance in predicting occlusions. Point tracking methods such as PIPs++[72] and CoTracker[33] exhibit robust performance, predicting correspondences across the entire image. They consider all frames between source and target time steps, tracking points even under occlusion, providing informative estimates for our method. Among these, we adopt CoTracker as it yields higher quality correspondences.

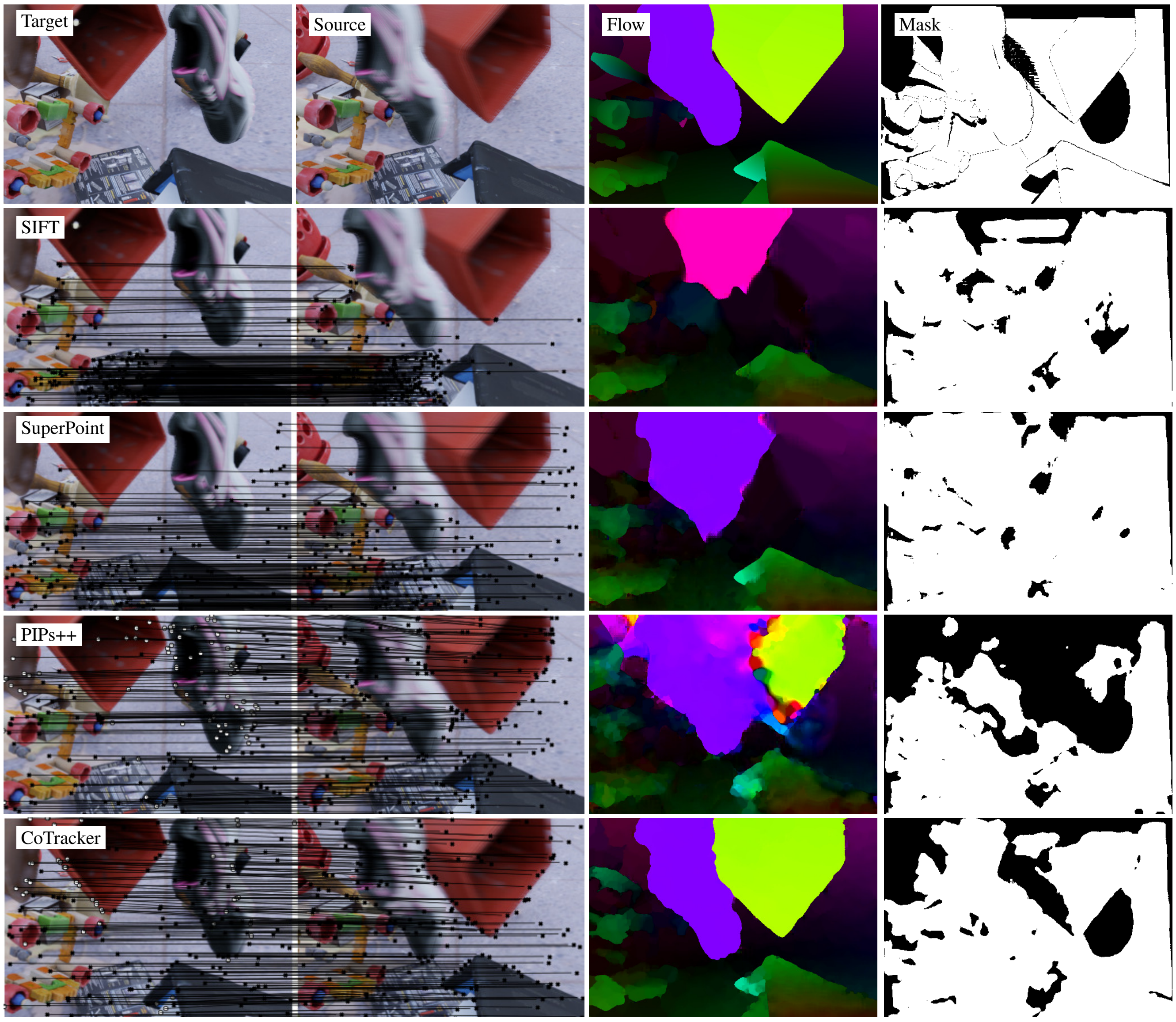


Figure F. **Effect of the method used to extract sparse correspondences.** We show the flow and visibility mask produced by DOT on the CVO (*Final*) test set when fed with N=1024 correspondences from different methods. For clarity, we only show correspondences for 256 pairs of points for each method (×: visible, ○: occluded). For fair comparison, we use the same densification model for all methods and do not do in-domain training. We represent motion directions in the flow using distinctive colors and visible regions in the mask in white.

G. Robustness to appearance changes

We make DOT robust to appearance changes by training on data featuring shadows and motion blur effects. CoTracker and RAFT, trained on the same data, are less robust to such changes, as shown in Figure G.

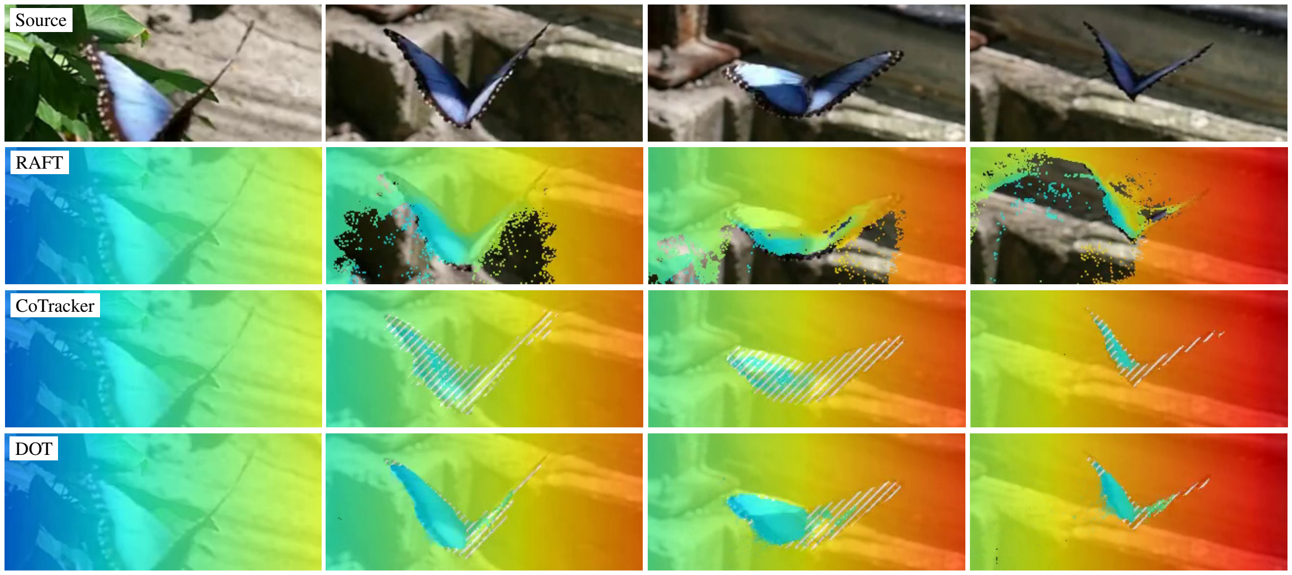


Figure G. **Robustness to appearance changes.** We track all points in the first frame of the "butterfly" video from the DAVIS dataset. RAFT loses part of the wing. CoTracker struggles with visibility.

H. Data curation pipeline

We have found that some videos from the CVO test set have erroneous optical flow ground truths due to objects being too close to the simulated camera. We have thus performed a systematic visual test for all the videos by showing simultaneously source and target frames, and the corresponding optical flow maps, as illustrated in Figure H. We have identified 25 corrupted samples out of more than 500 videos in the *Clean* and *Final* test sets but have not found any in our *Extended* set. We filter out these few problematic samples when comparing different methods in our experiments.

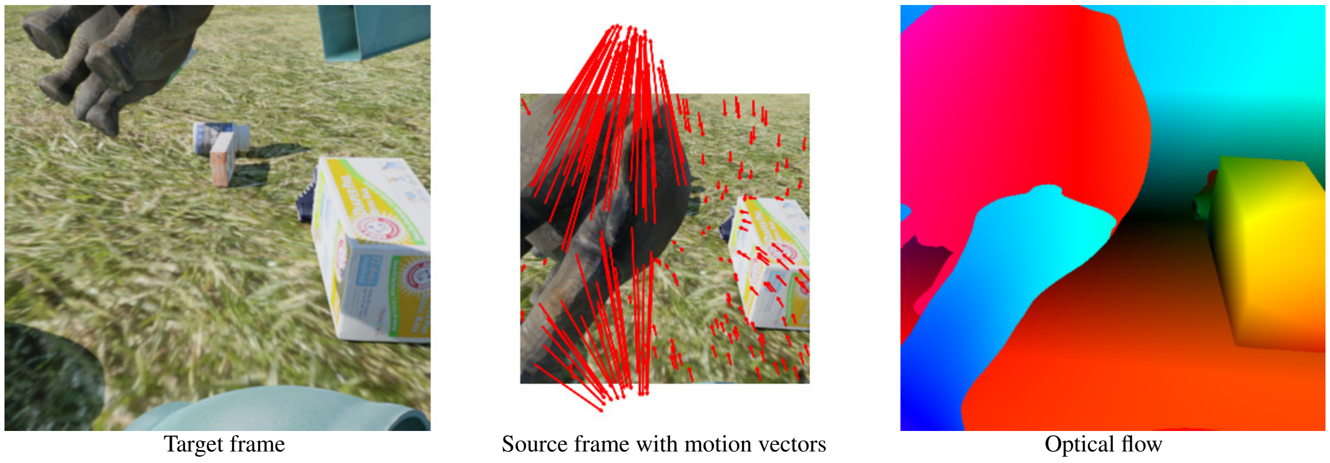


Figure H. **Data curation pipeline on the CVO dataset.** We identify samples with corrupted ground truth by visualizing source and target frames and the corresponding optical flow map side-by-side. In most cases, objects whose flow is incorrect have motion vectors pointing to very different directions. So we ease the verification process by showing a few motion vectors on top of the source frame. The corrupted part in the example presented here is the right hind leg of the toy.