Supplemental Material

A. About the Design Choice

In terms of integrating LIIF [2] into RAFT [9], designing the output $\mathcal{O}$ as motion fields may seem more natural. Instead of directly estimating motion vector, however, we define f_θ as the upsampling layer to produce the upsampling mask of convex combination. This has several advantages over directly producing optical flow with it.

If we define $\mathcal{O}$ as motion vector and produce high-resolution optical flow, e.g. $H \times W \times 2$, it requires $H \times W$ times inference of the MLP (f_θ), which is computationally intractable, especially on embedded devices. In addition, RAFT generates $\Delta \mathbf{f}_i$ by every GRU iteration and accumulates it, which means f_θ should be used by every iteration to estimate it. Instead, by defining $\mathcal{O}$ as the upsampling mask, f_θ runs only after the final GRU iteration and helps avoid computational cost proportional to the number of iterations, which could be incurred otherwise.

B. Implementation and Network Details

B.1. Neural Implicit Flow Upsampler

For the neural implicit flow upsampler, as mentioned in Sec.3.1, we apply positional encoding function [4, 10] ψ to the relative coordinates.

$$\psi(x) = (\sin(\omega_1 x), \cos(\omega_1 x), \cdots, \sin(\omega_L x), \cos(\omega_L x)), \tag{1}$$

where the frequency parameters are initialized as $\omega_l = 2e^l, l \in \{1, 2, ..., L\}$ and optimized during the training. L is set to 24 for our experiments. We also apply cell decoding [2] to leave a slight room for the output to vary with the target resolution. We do not adopt feature unfolding and local ensemble [2] for AnyFlow since they incur significant computational costs. Table 1 describes the network architecture of the implicit upsampler, f_θ.

B.2. Multi-scale Feature Warping

For multi-scale feature warping, we employ three 1×1 convolution layers and two PixelShuffle layers, as detailed in Eqns. 3-5 of the manuscript. Table 2 outlines the architectures of each 1×1 convolution layer, as well as the subsequent PixelShuffle layers [8].

The convolution layer in Table 2 (a) processes the input pair of half-scale feature maps, $[F_1^{1/2}, F_2^{1/2}]$, concatenated along the channel dimension. The layer in Table 2 (b) processes the input pair of quarter-scale feature maps. Lastly, the final layer in Table 2 (c) processes the outputs from the previous layers, $F^{1/2'}$ and $F^{1/4'}$.

Neural Implicit Flow Upsampler

layers	channel out	channel in
Linear, ReLU	256	180
Linear, ReLU	256	256
Linear, ReLU	256	256
Linear	144	256

Table 1. Network details of the neural implicit flow upsampler.

layers	channel out	channel in
1x1 Conv	16	64×2
PixelShuffle (scale=$1/4$)	256	16

(a)

layers	channel out	channel in
1x1 Conv	32	96×2
PixelShuffle (scale=$1/2$)	128	32

(b)

layers	channel out	channel in
1x1 Conv	128	374

(c)

Table 2. Network details of the multi-scale feature warping.

B.3. Dynamic Lookup with Region Encoding

We follow the general framework of RAFT [9] and each common module has the same architecture as that in RAFT. To enable dynamic lookup, we set the output channel to 3, where the first two channels are for the residual flow and the last channel is for the residual radius. As the update block in RAFT takes as input the accumulated flow to produce the residual flow for the next iteration, we also feed the accumulated radius to the update block by concatenating it with the flow.

For the region encoding, we use 2-layer MLP for the encoding function g_ϕ with an intermediate dimension of 12: g_ϕ =[Linear(12, 10), ReLU, Linear(1, 12)]. We adopt respective encoding function for each correlation volume in multiple scales.

C. Analysis of Neural Implicit Flow Upsampler

In this section, we perform further ablation studies about the neural implicit flow upsampler. The usage of it enables us to adopt several strategies for AnyFlow. Since it can generate optical flow in higher-resolution than that of input images, we perform multi-scale training. As mentioned in Sec. 3.4, we randomly downsample inputs, generate output in the target size, and then compute the loss function. Also, it can be used to improve the performance of the multi-scale warping method, which was verified in the manuscript. We also compare the effects of hyperparameters, n, and the effect of positional encoding. Note that we adopt a *dynamic lookup* strategy for all the results in this section.

In the top section of Table 3, we analyze the effect of multi-scale training. As shown in the table, the performance is severely degraded without multi-scale training. Since the implicit upsampler takes as inputs 2D position information, i.e. relative position and its encoding, it needs to learn the mapping from the position inputs to the outputs. Therefore, the network should encounter diverse ranges of position information during training and multi-scale training is required to satisfy this condition. Without it, the network only takes fixed positional inputs during training and lacks the ability to generalize, which degrades the overall performance. In addition, image downsampling decreases the motion ranges and trains the network to generalize well on diverse motion ranges. Therefore, multi-scale training is beneficial for designing the robust network which can generalize well on various scenarios.

In the middle section, we compare the results of different n. In this experiment, we adopt the bilinear interpolation for the multi-scale feature warping (MS *bilin.*). As mentioned in Sec. 3.1, each output $\mathcal{O}(x_q)$ represents the n-times upsampling weights for each query point x_q. That is, as n decreases, the number of samples that is required to produce the output in the target resolution quadratically grows. As shown in the table, we empirically found that AnyFlow shows the best results when n is set to 4. When comparing $n = 8$ with $n = 4$, smaller n increases the sampling granularity within each pixel and enables the network to learn the diverse position inputs during the training. However, there exists an accuracy-efficiency trade-off when deciding the value of n. As mentioned in Sec. A, the implicit upsampler is only used at the final iteration and it does not have a critical effect on final efficiency during *inference*. However, it does for *training*, where the upsampler should be used at every iteration to compute the loss functions. Therefore, a smaller n affects the training efficiency, increasing the training time and memory.

In the last section, we evaluate the effect of the positional encoding. As the results show, AnyFlow benefits from the usage of positional encoding. Even though optical flow generally consists of locally smooth motion fields, learning high-frequency details through the position encoding is useful to learn clear and accurate boundaries. Furthermore, feeding the high-dimensional inputs by encoding the positions helps the network model the complex mapping from the position information to the outputs, instead of only feeding the 2D coordinates.

Method	Sintel		KITTI	
	Clean	Final	F1-epe	F1-all
w/o m.s. training	1.31	**2.57**	4.39	14.10
w/ m.s. training	**1.17**	2.58	**3.95**	**13.01**
MS *bilin.*, $n = 8$	1.22	2.60	4.17	13.44
MS *bilin.*, $n = 4$	**1.20**	**2.58**	**4.01**	**13.12**
MS *bilin.*, $n = 2$	1.25	2.71	4.22	13.29
w/o p.e.	1.21	2.62	4.21	13.27
w/ p.e.	**1.17**	**2.58**	**3.95**	**13.01**

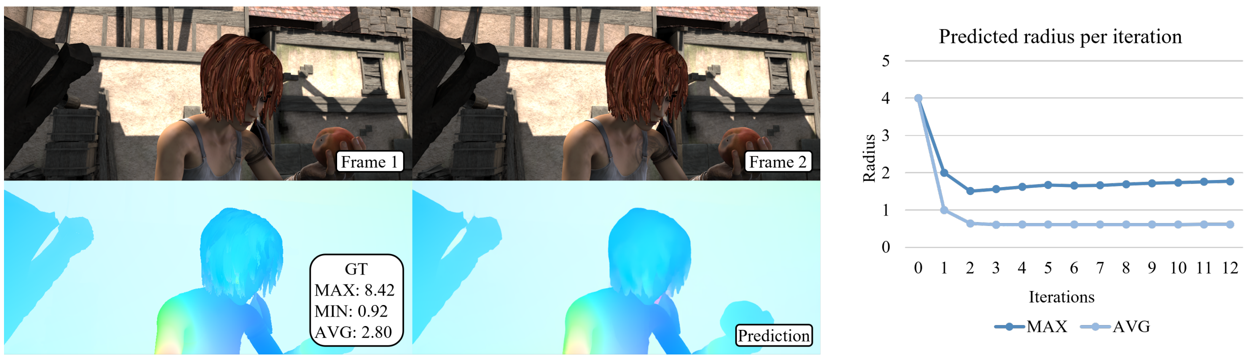
Table 3. Ablation experiments for the neural implicit flow upsampler. m.s. denotes multi-scale training strategy, and p.e. denotes the positional encoding. The training is performed on FlyingChairs [3] and FlyingThings [6].

D. Analysis of Dynamic Lookup

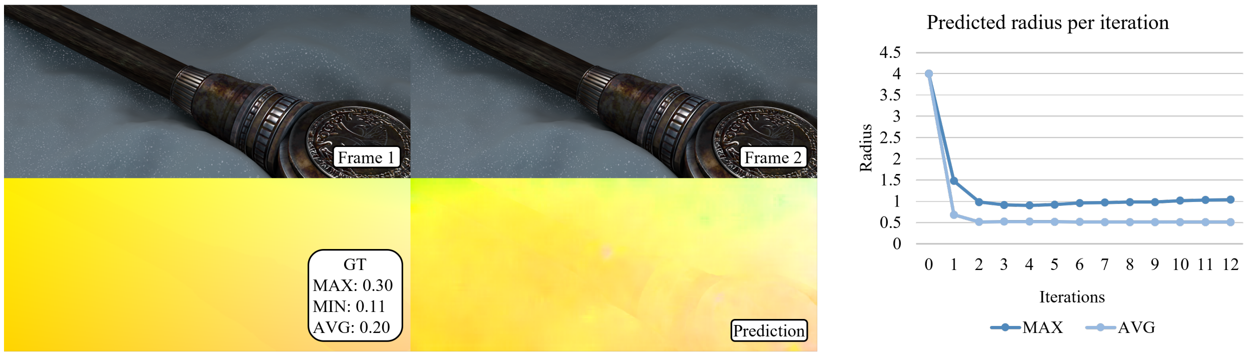
In this section, we analyze how the network predicts the radius depending on the motion ranges and input images. Our desired behavior is that the network increases the radius to capture large displacements when the input pair contains large motions. On the other hand, the network should reduce the radius to focus on small areas and produce precise estimations when the input pair mostly contains small motions.

In Fig. 1, we describe maximum, average and minimum values of norm of ground-truth flow in each image, and visualize the changes of the predicted radius by iterations. MAX denotes the maximum value of predicted radius across all pixels, and AVG denotes the averaged one over all pixels. We set the initial radius, r_0, as 4px for all examples.

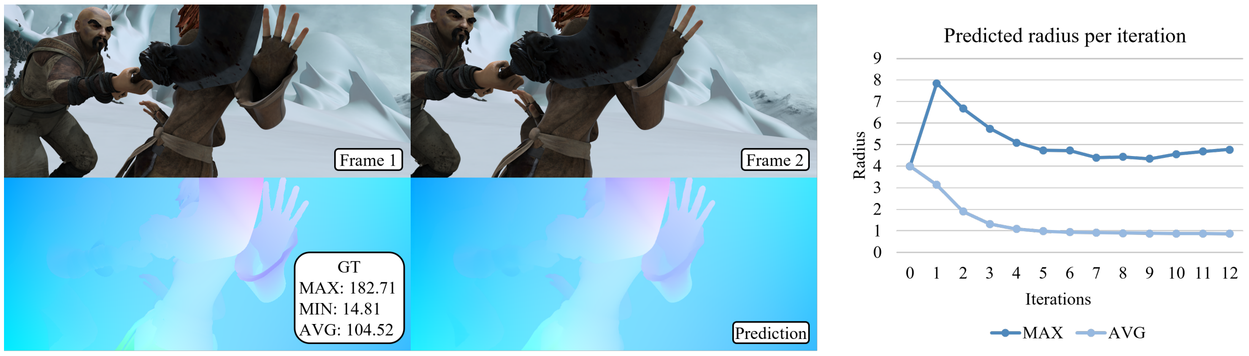
Example (a) contains motions within a range between 0.92 and 8.42, and example (b) contains motions within a range between 0.11 and 0.30. Both cases contain mostly small motions. When the input images contain small motions, as shown in the right columns, the predicted radius decreases as the number of iterations grows. As the initial radius is set to 4, the receptive field of each grid for



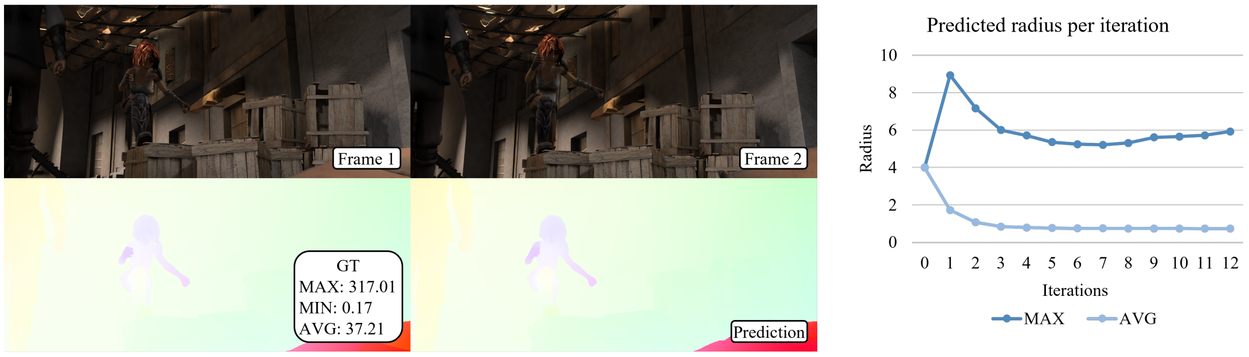
(a)



(b)



(c)



(d)

Figure 1. Examples on the Sintel [1] dataset. We describe motion ranges in the ground-truth of each example and the changes of radius predicted by AnyFlow in each iteration using the dynamic lookup strategy. The training is performed on FlyingChairs [3] and FlyingThings [6].

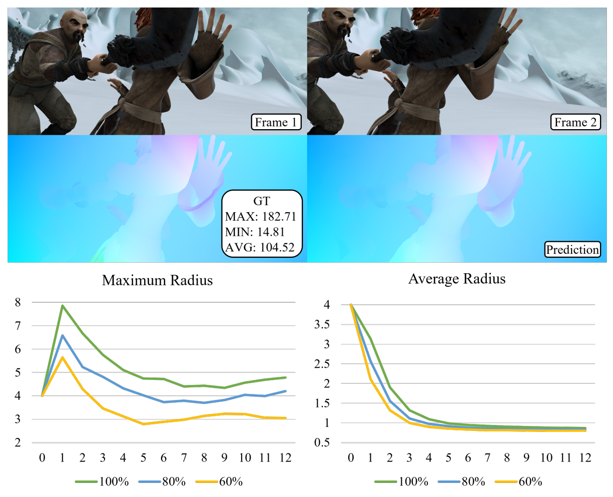


Figure 2. Comparisons of the predicted radius in each iteration as the image resolution decreases.

the correlation lookup becomes $(2 \times 4 \times 8)^2$ area because the correlation sampling is performed on $1/8$-sized feature maps. Therefore, the initial lookup already covers every motion range and the network tries to reduce the radius to concentrate more on real correspondence from the second iteration. In addition, example (a) contains relatively larger motions than those in (b), and the average radius is predicted as larger for (a) than that in (b).

On the other hand, examples (c) and (d) contain large displacements, e.g. larger than 100px, and the network predicts much larger radius for these inputs. As the initial radius is set as 4, the receptive fields for the correlation lookup do not cover all the regions that contain expected correspondence. Therefore, the network tries to increase the radius to find the correspondence.

Fig. 2 shows the maximum and average radius, as the network takes as input the downsampled images. We feed a single pair of images downsampled by different scale factors and visualize the radius changes. As shown in the figure, the network tends to predict smaller radius as the downsample factor increases. This demonstrates the ability of AnyFlow to generalize well on diverse resolutions and perform robust estimation.

E. Runtime Analysis

In Fig. 3, we report end-to-end point error (EPE) on the Sintel clean as a function of runtime. The runtimes are estimated for inferencing one 1024×436 image using an NVIDIA V100 GPU. As AnyFlow is an iterative approach, naive comparisons of runtimes under a fixed number of iterations do not provide meaningful information. Therefore, we analyze how long AnyFlow takes to reach the target accuracy and how many iterations it requires. We compare it with RAFT [9] and GMA [5], and report EPE and runtimes for a total of 32 iterations.

Even though AnyFlow with the region encoding (denoted as R.E.) takes more time for running a single iteration than that of RAFT, it only requires 5 iterations to achieve a lower EPE than the best EPE of RAFT. After 7^{th} iteration, it achieves better accuracy than the best accuracy of GMA [5]. After the 15^{th} iteration, AnyFlow (R.E.) achieves 1.10 EPE in 0.183s. Since the RAFT uses 32 iterations to achieve the best accuracy on Sintel, which takes 0.195s for the inference. AnyFlow (R.E.) achieves much better results (1.10 vs 1.47) as well as enables faster inference.

We also report the results of AnyFlow (dynamic). Since the region encoding takes more time to encode regional correlation, AnyFlow (dynamic) shows better efficiency. It takes less time for 32 iterations than GMA [5] and achieves better results at the same time. Only after the 9^{th} iteration, it achieves 1.27 EPE in 0.086s and outperforms GMA.

F. Qualitative Results on KITTI

We visualize the optical flow predictions of AnyFlow, RAFT [9] and GMA [5] on KITTI test images in Fig. 4 and Fig. 5. As shown in Fig. 4, AnyFlow shows more accurate shapes and boundaries of objects with preserving details. Fig. 5 shows the examples that contain small moving objects and small motions. Since the person in the scenes is far from the camera, it is difficult for the other methods to detect it. As AnyFlow benefits from high-resolution feature maps thanks to the multi-scale warping strategy, it can more precisely estimate small objects in the real-world dataset. These results further demonstrate the ability of AnyFlow that can generalize well on real world-scenes, KITTI.

In Table 4, we also compare our results with DIP [11]. In Table 1 in the manuscript, we only compare F1-all, which includes all regions for computing the metric. Even though DIP shows better F1-all than ours, we achieve lower F1-fg, further demonstrating that AnyFlow performs precise estimation of especially the foreground objects.

G. Online Benchmarks

In Fig. 6 and Fig. 7, we demonstrate the results on the test set of Sintel [1] and KITTI [7]. The results are the same as the ones we present in Table 1 in the manuscript.

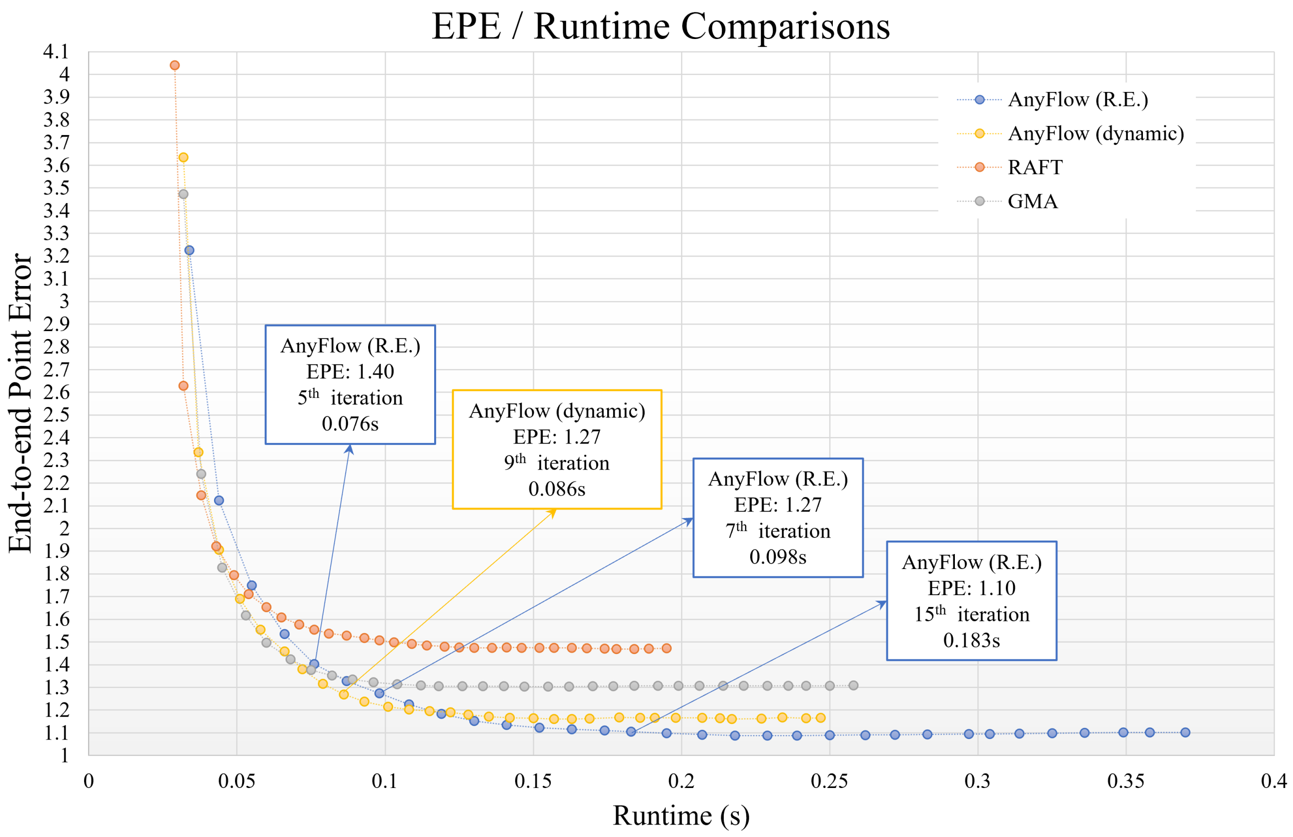


Figure 3. End-to-end point error (EPE) and runtime comparisons with RAFT [9] and GMA [5]. We report the data points for a total of 32 iterations.

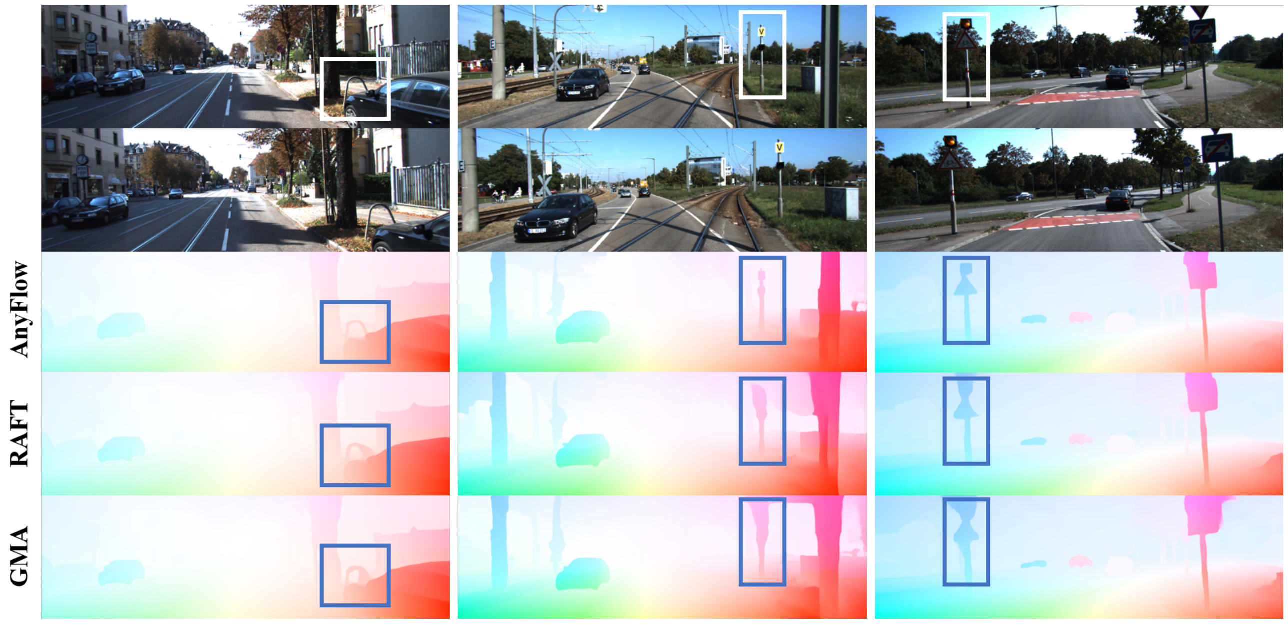


Figure 4. Qualitative results of AnyFlow, RAFT [9] and GMA [5] on KITTI test images. AnyFlow shows clearer and more accurate shape of objects with detail preservation.

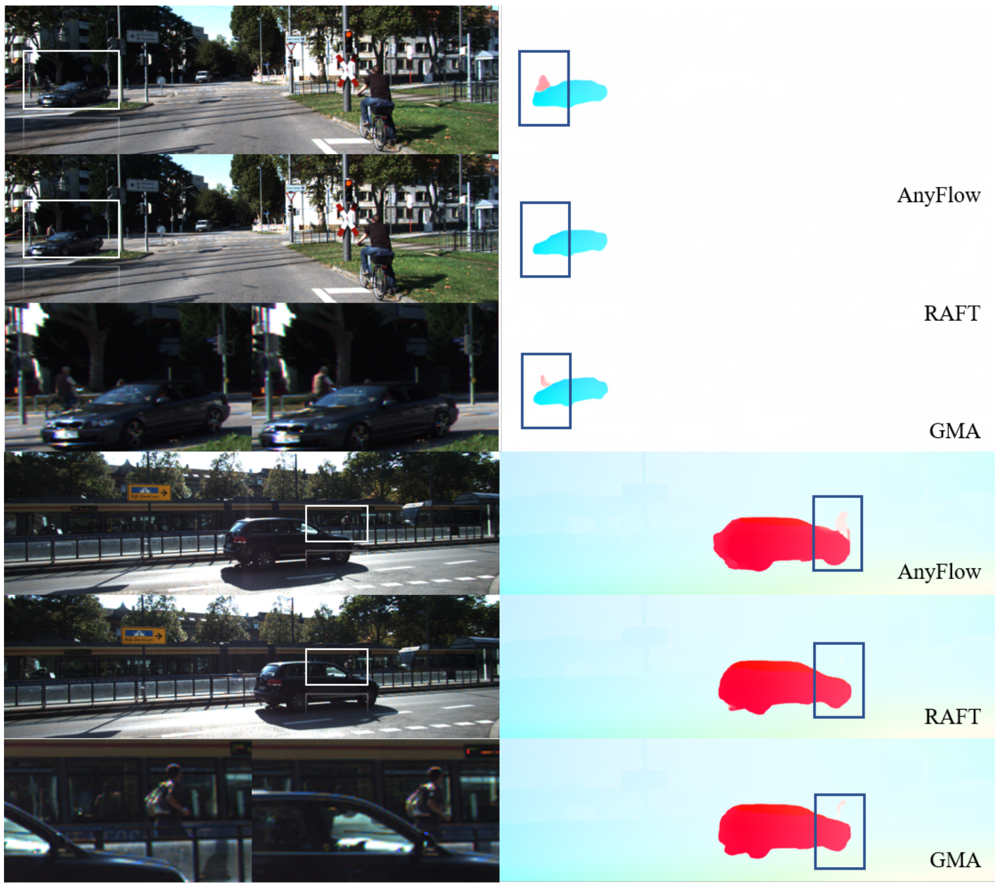
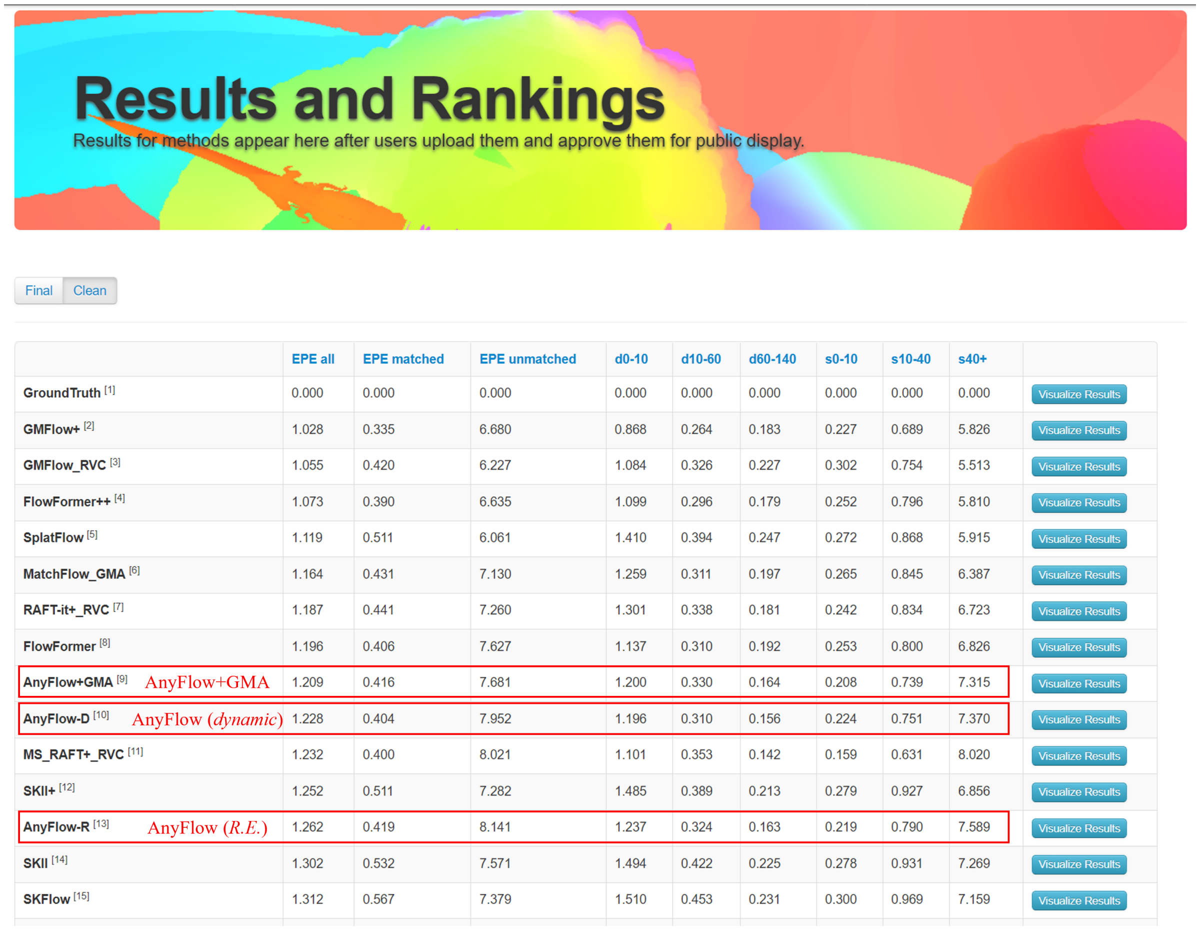


Figure 5. Qualitative results of AnyFlow, RAFT [9] and GMA [5] on KITTI test images. AnyFlow detects small objects and small motions well, where the other methods fail to detect.

Training	C + T + S + K + H	
Method	KITTI (test)	
	F1-fg	F1-all
DIP [11]	5.96	**4.21**
AnyFlow (*dynamic*)	**5.76**	4.41

Table 4. Comparisons of F1-fg and F1-all with DIP [11] on the KITTI test images.



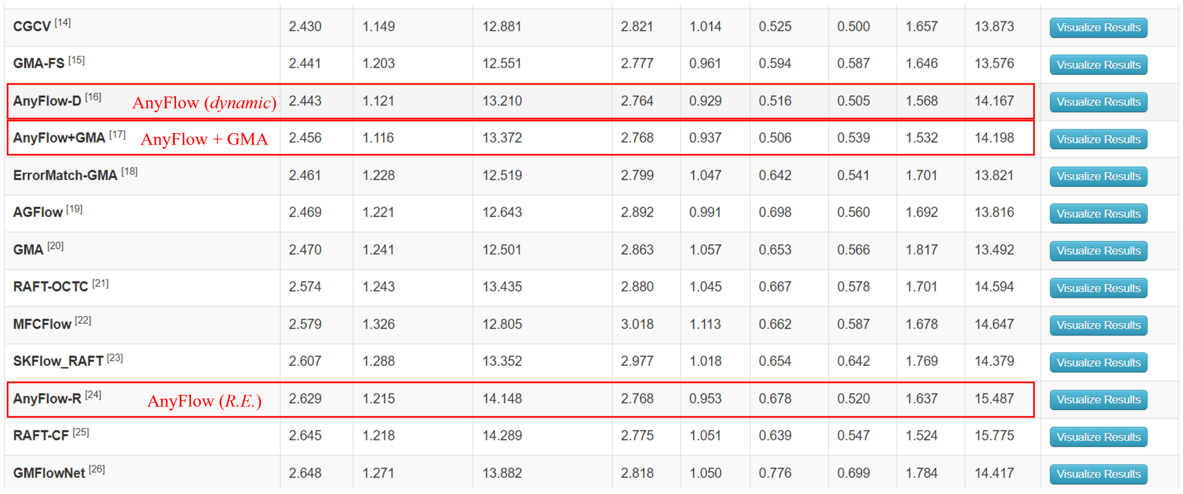
Results and Rankings

Results for methods appear here after users upload them and approve them for public display.

Final | Clean

	EPE all	EPE matched	EPE unmatched	d0-10	d10-60	d60-140	s0-10	s10-40	s40+	
GroundTruth [1]	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	Visualize Results
GMFlow+ [2]	1.028	0.335	6.680	0.868	0.264	0.183	0.227	0.689	5.826	Visualize Results
GMFlow_RVC [3]	1.055	0.420	6.227	1.084	0.326	0.227	0.302	0.754	5.513	Visualize Results
FlowFormer++ [4]	1.073	0.390	6.635	1.099	0.296	0.179	0.252	0.796	5.810	Visualize Results
SplatFlow [5]	1.119	0.511	6.061	1.410	0.394	0.247	0.272	0.868	5.915	Visualize Results
MatchFlow_GMA [6]	1.164	0.431	7.130	1.259	0.311	0.197	0.265	0.845	6.387	Visualize Results
RAFT-it+_RVC [7]	1.187	0.441	7.260	1.301	0.338	0.181	0.242	0.834	6.723	Visualize Results
FlowFormer [8]	1.196	0.406	7.627	1.137	0.310	0.192	0.253	0.800	6.826	Visualize Results
AnyFlow+GMA [9] AnyFlow+GMA	1.209	0.416	7.681	1.200	0.330	0.164	0.208	0.739	7.315	Visualize Results
AnyFlow-D [10] AnyFlow (*dynamic*)	1.228	0.404	7.952	1.196	0.310	0.156	0.224	0.751	7.370	Visualize Results
MS_RAFT+_RVC [11]	1.232	0.400	8.021	1.101	0.353	0.142	0.159	0.631	8.020	Visualize Results
SKII+ [12]	1.252	0.511	7.282	1.485	0.389	0.213	0.279	0.927	6.856	Visualize Results
AnyFlow-R [13] AnyFlow (*R.E.*)	1.262	0.419	8.141	1.237	0.324	0.163	0.219	0.790	7.589	Visualize Results
SKII [14]	1.302	0.532	7.571	1.494	0.422	0.225	0.278	0.931	7.269	Visualize Results
SKFlow [15]	1.312	0.567	7.379	1.510	0.453	0.231	0.300	0.969	7.159	Visualize Results

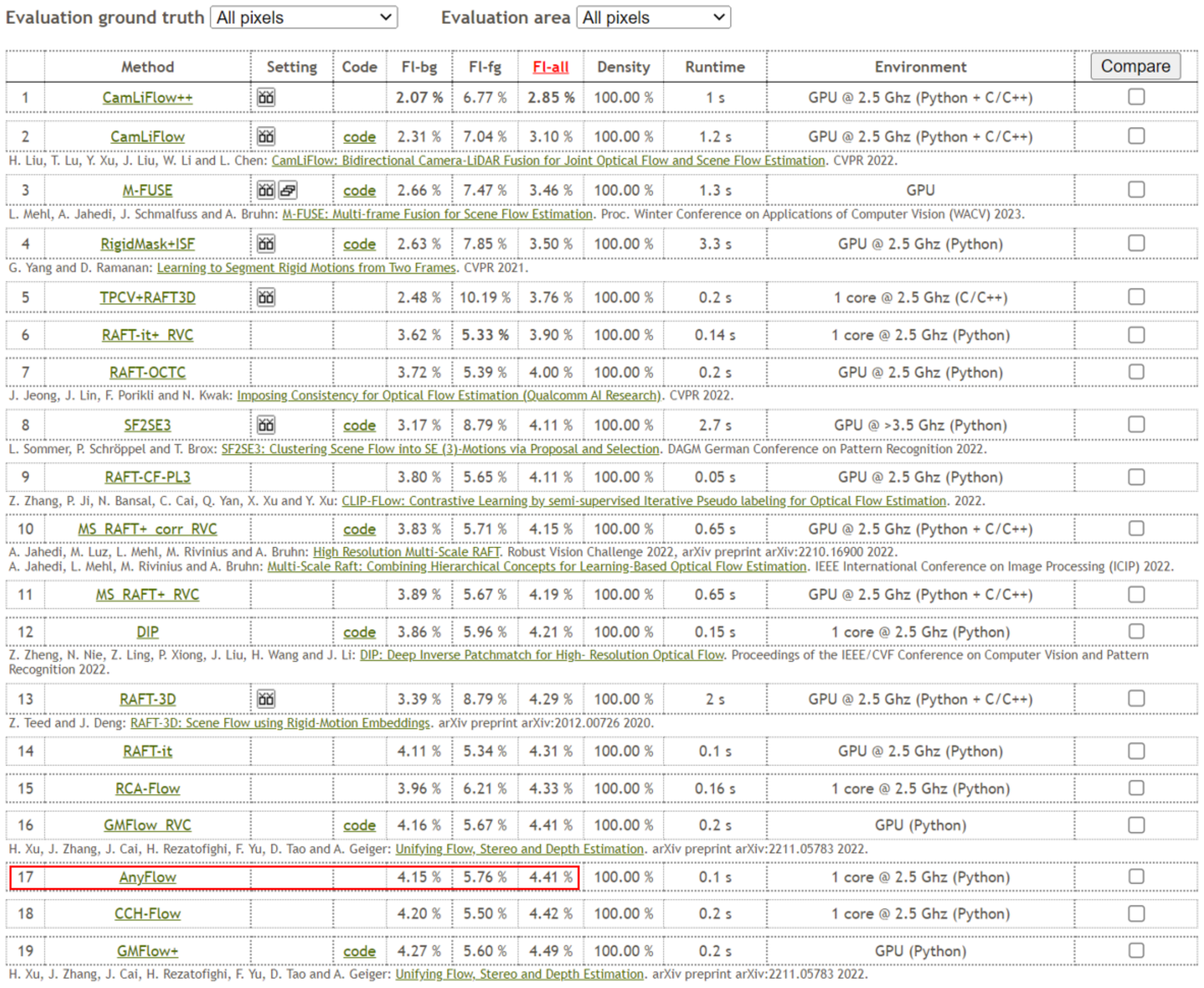
(a) Results on Sintel clean dataset.



CGCV [14]	2.430	1.149	12.881	2.821	1.014	0.525	0.500	1.657	13.873	Visualize Results
GMA-FS [15]	2.441	1.203	12.551	2.777	0.961	0.594	0.587	1.646	13.576	Visualize Results
AnyFlow-D [16] AnyFlow (*dynamic*)	2.443	1.121	13.210	2.764	0.929	0.516	0.505	1.568	14.167	Visualize Results
AnyFlow+GMA [17] AnyFlow + GMA	2.456	1.116	13.372	2.768	0.937	0.506	0.539	1.532	14.198	Visualize Results
ErrorMatch-GMA [18]	2.461	1.228	12.519	2.799	1.047	0.642	0.541	1.701	13.821	Visualize Results
AGFlow [19]	2.469	1.221	12.643	2.892	0.991	0.698	0.560	1.692	13.816	Visualize Results
GMA [20]	2.470	1.241	12.501	2.863	1.057	0.653	0.566	1.817	13.492	Visualize Results
RAFT-OCTC [21]	2.574	1.243	13.435	2.880	1.045	0.667	0.578	1.701	14.594	Visualize Results
MFCFlow [22]	2.579	1.326	12.805	3.018	1.113	0.662	0.587	1.678	14.647	Visualize Results
SKFlow_RAFT [23]	2.607	1.288	13.352	2.977	1.018	0.654	0.642	1.769	14.379	Visualize Results
AnyFlow-R [24] AnyFlow (*R.E.*)	2.629	1.215	14.148	2.768	0.953	0.678	0.520	1.637	15.487	Visualize Results
RAFT-CF [25]	2.645	1.218	14.289	2.775	1.051	0.639	0.547	1.524	15.775	Visualize Results
GMFlowNet [26]	2.648	1.271	13.882	2.818	1.050	0.776	0.699	1.784	14.417	Visualize Results

(b) Results on Sintel final dataset.

Figure 6. Screenshots for Sintel results on the online benchmarks.



Evaluation ground truth: All pixels | Evaluation area: All pixels

	Method	Setting	Code	Fl-bg	Fl-fg	Fl-all	Density	Runtime	Environment	Compare
1	CamLiFlow++	▣		**2.07 %**	6.77 %	**2.85 %**	100.00 %	1 s	GPU @ 2.5 Ghz (Python + C/C++)	☐
2	CamLiFlow	▣	code	2.31 %	7.04 %	3.10 %	100.00 %	1.2 s	GPU @ 2.5 Ghz (Python + C/C++)	☐
H. Liu, T. Lu, Y. Xu, J. Liu, W. Li and L. Chen: CamLiFlow: Bidirectional Camera-LiDAR Fusion for Joint Optical Flow and Scene Flow Estimation. CVPR 2022.										
3	M-FUSE	▣ ▣	code	2.66 %	7.47 %	3.46 %	100.00 %	1.3 s	GPU	☐
L. Mehl, A. Jahedi, J. Schmalfuss and A. Bruhn: M-FUSE: Multi-frame Fusion for Scene Flow Estimation. Proc. Winter Conference on Applications of Computer Vision (WACV) 2023.										
4	RigidMask+ISF	▣	code	2.63 %	7.85 %	3.50 %	100.00 %	3.3 s	GPU @ 2.5 Ghz (Python)	☐
G. Yang and D. Ramanan: Learning to Segment Rigid Motions from Two Frames. CVPR 2021.										
5	TPCV+RAFT3D	▣		2.48 %	10.19 %	3.76 %	100.00 %	0.2 s	1 core @ 2.5 Ghz (C/C++)	☐
6	RAFT-it+_RVC			3.62 %	**5.33 %**	3.90 %	100.00 %	0.14 s	1 core @ 2.5 Ghz (Python)	☐
7	RAFT-OCTC			3.72 %	5.39 %	4.00 %	100.00 %	0.2 s	GPU @ 2.5 Ghz (Python)	☐
J. Jeong, J. Lin, F. Porikli and N. Kwak: Imposing Consistency for Optical Flow Estimation (Qualcomm AI Research). CVPR 2022.										
8	SF2SE3	▣	code	3.17 %	8.79 %	4.11 %	100.00 %	2.7 s	GPU @ >3.5 Ghz (Python)	☐
L. Sommer, P. Schröppel and T. Brox: SF2SE3: Clustering Scene Flow into SE (3)-Motions via Proposal and Selection. DAGM German Conference on Pattern Recognition 2022.										
9	RAFT-CF-PL3			3.80 %	5.65 %	4.11 %	100.00 %	0.05 s	GPU @ 2.5 Ghz (Python)	☐
Z. Zhang, P. Ji, N. Bansal, C. Cai, Q. Yan, X. Xu and Y. Xu: CLIP-FLow: Contrastive Learning by semi-supervised Iterative Pseudo labeling for Optical Flow Estimation. 2022.										
10	MS_RAFT+_corr_RVC		code	3.83 %	5.71 %	4.15 %	100.00 %	0.65 s	GPU @ 2.5 Ghz (Python + C/C++)	☐
A. Jahedi, M. Luz, L. Mehl, M. Rivinius and A. Bruhn: High Resolution Multi-Scale RAFT. Robust Vision Challenge 2022, arXiv preprint arXiv:2210.16900 2022. A. Jahedi, L. Mehl, M. Rivinius and A. Bruhn: Multi-Scale Raft: Combining Hierarchical Concepts for Learning-Based Optical Flow Estimation. IEEE International Conference on Image Processing (ICIP) 2022.										
11	MS_RAFT+_RVC			3.89 %	5.67 %	4.19 %	100.00 %	0.65 s	GPU @ 2.5 Ghz (Python + C/C++)	☐
12	DIP		code	3.86 %	5.96 %	4.21 %	100.00 %	0.15 s	1 core @ 2.5 Ghz (Python)	☐
Z. Zheng, N. Nie, Z. Ling, P. Xiong, J. Liu, H. Wang and J. Li: DIP: Deep Inverse Patchmatch for High- Resolution Optical Flow. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2022.										
13	RAFT-3D	▣		3.39 %	8.79 %	4.29 %	100.00 %	2 s	GPU @ 2.5 Ghz (Python + C/C++)	☐
Z. Teed and J. Deng: RAFT-3D: Scene Flow using Rigid-Motion Embeddings. arXiv preprint arXiv:2012.00726 2020.										
14	RAFT-it			4.11 %	5.34 %	4.31 %	100.00 %	0.1 s	GPU @ 2.5 Ghz (Python)	☐
15	RCA-Flow			3.96 %	6.21 %	4.33 %	100.00 %	0.16 s	1 core @ 2.5 Ghz (Python)	☐
16	GMFlow_RVC		code	4.16 %	5.67 %	4.41 %	100.00 %	0.2 s	GPU (Python)	☐
H. Xu, J. Zhang, J. Cai, H. Rezatofighi, F. Yu, D. Tao and A. Geiger: Unifying Flow, Stereo and Depth Estimation. arXiv preprint arXiv:2211.05783 2022.										
17	AnyFlow			4.15 %	5.76 %	4.41 %	100.00 %	0.1 s	1 core @ 2.5 Ghz (Python)	☐
18	CCH-Flow			4.20 %	5.50 %	4.42 %	100.00 %	0.2 s	1 core @ 2.5 Ghz (Python)	☐
19	GMFlow+		code	4.27 %	5.60 %	4.49 %	100.00 %	0.2 s	GPU (Python)	☐
H. Xu, J. Zhang, J. Cai, H. Rezatofighi, F. Yu, D. Tao and A. Geiger: Unifying Flow, Stereo and Depth Estimation. arXiv preprint arXiv:2211.05783 2022.										

Figure 7. Screenshot for KITTI results on the online benchmark.

References

[1] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *ECCV*, 2012. 3, 4

[2] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. In *CVPR*, pages 8628–8638, 2021. 1

[3] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, 2015. 2, 3

[4] Hanzhe Hu, Yinbo Chen, Jiarui Xu, Shubhankar Borse, Hong Cai, Fatih Porikli, and Xiaolong Wang. Learning implicit feature alignment function for semantic segmentation. In *ECCV*, 2022. 1

[5] Shihao Jiang, Dylan Campbell, Yao Lu, Hongdong Li, and Richard Hartley. Learning to estimate hidden motions with global motion aggregation. In *ICCV*, 2021. 4, 5, 6

[6] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *CVPR*, 2016. 2, 3

[7] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *CVPR*, 2015. 4

[8] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *CVPR*, 2016. 1

[9] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *ECCV*, 2020. 1, 4, 5, 6

[10] Xingqian Xu, Zhangyang Wang, and Humphrey Shi. Ultrasr: Spatial encoding is a missing key for implicit image function-based arbitrary-scale super-resolution. In *arXiv preprint arXiv:2103.12716*, 2021. 1

[11] Zihua Zheng, Ni Nie, Zhi Ling, Pengfei Xiong, Jiangyu Liu, Hao Wang, and Jiankun Li. Dip: Deep inverse patchmatch for high-resolution optical flow. In *CVPR*, 2022. 4, 6