GMFlow: Learning Optical Flow via Global Matching
Supplementary Material

Haofei Xu[1]* Jing Zhang[2] Jianfei Cai[1] Hamid Rezatofighi[1] Dacheng Tao[3,2]
[1]Department of Data Science and AI, Monash University, Australia
[2]The University of Sydney, Australia [3]JD Explore Academy, China
{xuhfgm,dacheng.tao}@gmail.com jing.zhang1@sydney.edu.au {jianfei.cai,hamid.rezatofighi}@monash.edu

A. More Comparisons

We present more comprehensive comparisons (as a supplement of Table 1 in the main paper) with all the possible combinations of flow estimation approaches in Table A. Our Transformer and softmax-based method is consistently better and has less parameters than other variants.

B. Computational Complexity

We analyze the computational complexities of core components in our framework below.

Global Matching. In our global matching formulation, we build a 4D correlation matrix $H \times W \times H \times W$ to model all pair-wise similarities between two features (with size $H \times W$, $1/8$ of the original image resolution). There exists an equivalent implementation should it become a bottleneck for high-resolution images. Note that the pixels in the first feature are *independent* and thus their flow predictions can be computed *sequentially*. Specifically, we can sequentially compute $K \times K$ correlation matrices (each with size $H/K \times W/K \times H \times W$), and finally merge the results for all pixels. Such a sequential implementation can save the memory consumption while having little influence on the overall inference time (see Table B), since the global matching operation only needs to compute once, and it's not a significant speed bottleneck in the full framework.

#splits	1×1	2×2	4×4	8×8
Time (ms)	52.57	52.64	52.90	59.45

Table B. **Inference time *vs*. number of splits for sequential global matching implementation.** The input image resolution is 448×1024, and the features are downsampled by $8\times$.

We note that our alternative sequential implementation is not applicable for previous cost volume and convolution-based approaches (*e.g.*, RAFT [3]), since the cost volume is used as an intermediate component for subsequent regression with convolutions, where all pixels in the spatial dimension are tightly coupled.

Transformer. We use shifted local window attention [1] in the Transformer implementation, where each local window size is $1/16$ of the original image resolution by default. The computational cost is usually acceptable for regular image resolutions (*e.g.*, 448×1024). Note that we can always switch to smaller windows size (*e.g.*, $1/32$, see Table 2b of the main paper) should it become a bottleneck.

Flow Propagation. Our default flow propagation scheme computes a global self-attention. The sequential implementation in global matching can also be adopted here. It's also possible to compute a local window self-attention only for less memory consumption by trading some accuracy in large motion (Table C). Such a local attention operation can be implemented efficiently with PyTorch's `unfold` function.

self-attn.	Sintel (train, final)			
	EPE	s_{0-10}	s_{10-40}	s_{40+}
global	**3.13**	0.80	3.87	**18.04**
local 3×3	3.31	0.79	3.75	20.22
local 5×5	3.21	**0.75**	**3.66**	19.69

Table C. **Global *vs*. local self-attention for flow propagation.**

Refinement. Although the feature resolution of our refinement architecture is higher ($1/4$), it is not a significant bottleneck since smaller local window ($1/32$ of the original image resolution) attention is used in the Transformer and matching is performed within a local window.

Overall, our GMFlow framework is general and flexible, and many concrete implementations are possible to meet specific needs.

C. More Visual Results

Flow Propagation. Our flow propagation scheme with

*Work done during Haofei's internship at JD Explore Academy

Method	feature enhancement	flow prediction	#convs	Sintel (train, clean)			Sintel (train, final)			Param (M)
				all	matched	unmatched	all	matched	unmatched	
variants	-	cost + conv	14	3.32	1.56	22.34	4.93	2.82	27.73	4.64
	Transformer	cost + conv	2	3.41	2.40	14.32	4.57	3.27	18.67	4.95
	Transformer	cost + conv	14	2.04	1.09	12.34	3.37	2.03	17.80	7.79
	conv	softmax	14	6.36	3.22	40.30	8.00	4.80	42.58	5.12
GMFlow (w/o prop.)	Transformer	softmax	0	2.28	**1.06**	15.54	3.44	**1.95**	19.50	4.20
GMFlow (w/ prop.)	Transformer	softmax	0	**1.89**	1.10	**10.39**	**3.13**	1.98	**15.52**	4.23

Table A. **Comparisons on different variants of flow estimation approaches.** Although the Transformer can also be used for feature enhancement in the cost volume and convolution-based approach (cost + conv), its performance heavily relies on a deep convolutional regressor (*e.g.*, 14 layers to catch up). In contrast, our softmax-based method is *parameter-free* (4.20M *vs*. 7.79M). The flow propagation (prop.) layer further improves ours performance in unmatched regions, while only introducing additional 0.03M parameters. Replacing the Transformer with convolutions for feature enhancement leads to significantly large performance drop, since convolutions are not able to model the mutual relationship between two features.

self-attention is quite effective for handling occluded and out-of-boundary pixels, as can be seen from Fig. A.

Prediction on DAVIS dataset. We test our pre-trained Sintel model on the DAVIS [2] dataset, the results on diverse scenes are shown in Fig. B.

D. More Implementation Details

Network Architectures. The Transformer feature dimension is 128, and the intermediate feed-forward network expands the dimension by $4\times$. We only use a single head in all the attention computations, since we observe that multi-head attention slows down the speed without bringing obvious performance gains. Our refinement architecture uses exactly the same Transformer for feature enhancement, except that the attentions are performed within smaller local windows. The self-attention layer in the flow propagation step is also shared for $1/8$ and $1/4$ resolutions, where we perform global attention at $1/8$ resolution and local 3×3 window attention at $1/4$ resolution.

Training Details. Our data augmentation strategy mostly follows RAFT [3] except that we didn't use occlusion augmentation, since no obvious improvement is observed in our experiments. During training, we perform random cropping following previous works. The crop size for FlyingChairs is 384×512, FlyingThings3D is 384×768, Sintel is 320×896 and KITTI is 320×1152. Our framework without refinement is trained on 4 V100 (16GB) GPUs. The full framework with refinement is trained on 4 A100 (40GB) GPUs. We are also able to reproduce the results on 4 V100 (16GB) GPUs by halving the batch size and doubling the training iterations.

References

[1] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *ICCV*, 2021. 1

[2] Federico Perazzi, Jordi Pont-Tuset, Brian McWilliams, Luc Van Gool, Markus Gross, and Alexander Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *CVPR*, pages 724–732, 2016. 2

[3] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *ECCV*, pages 402–419. Springer, 2020. 1, 2

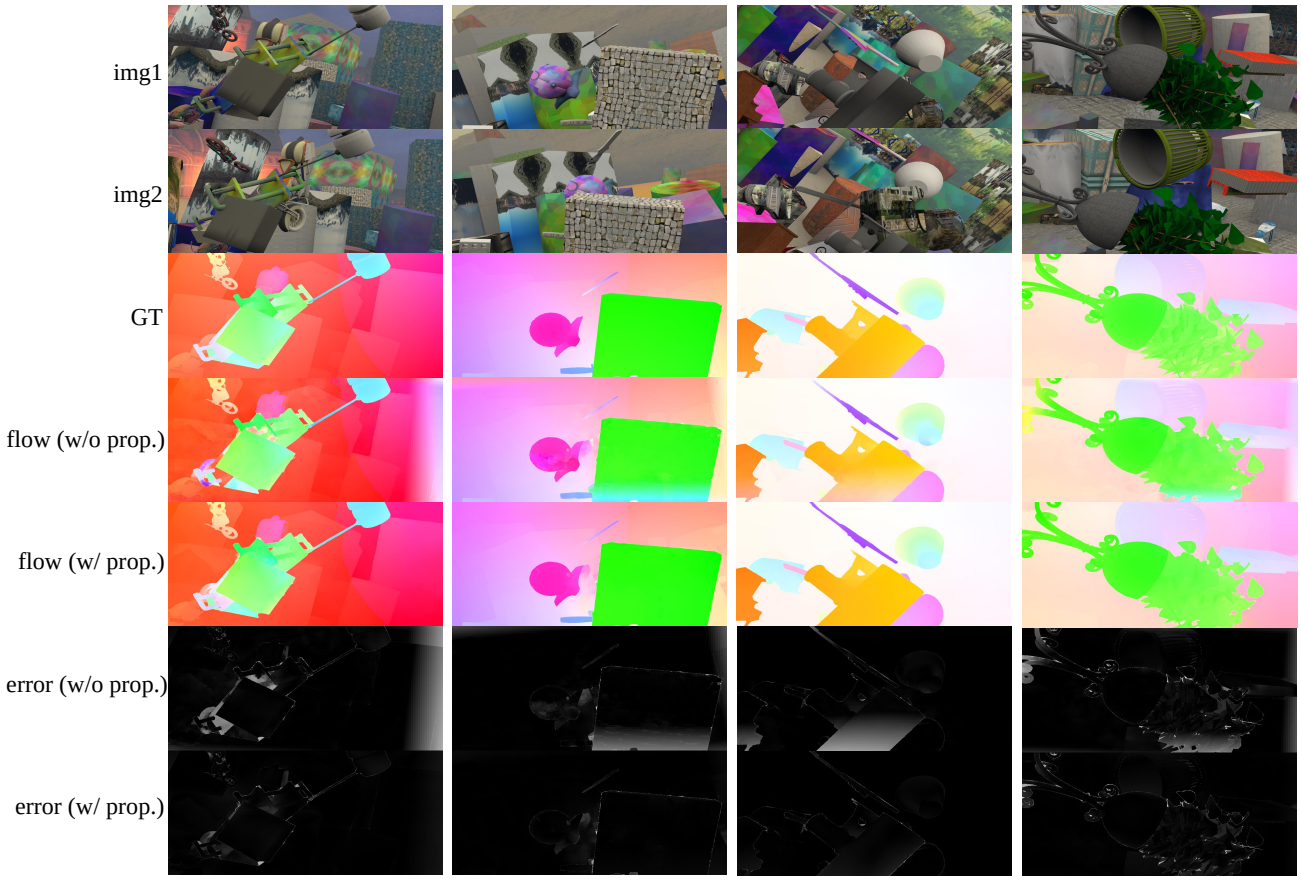

Figure A. Our **flow propagation** (prop.) scheme significantly improves the performance of occluded and out-of-boundary pixels.

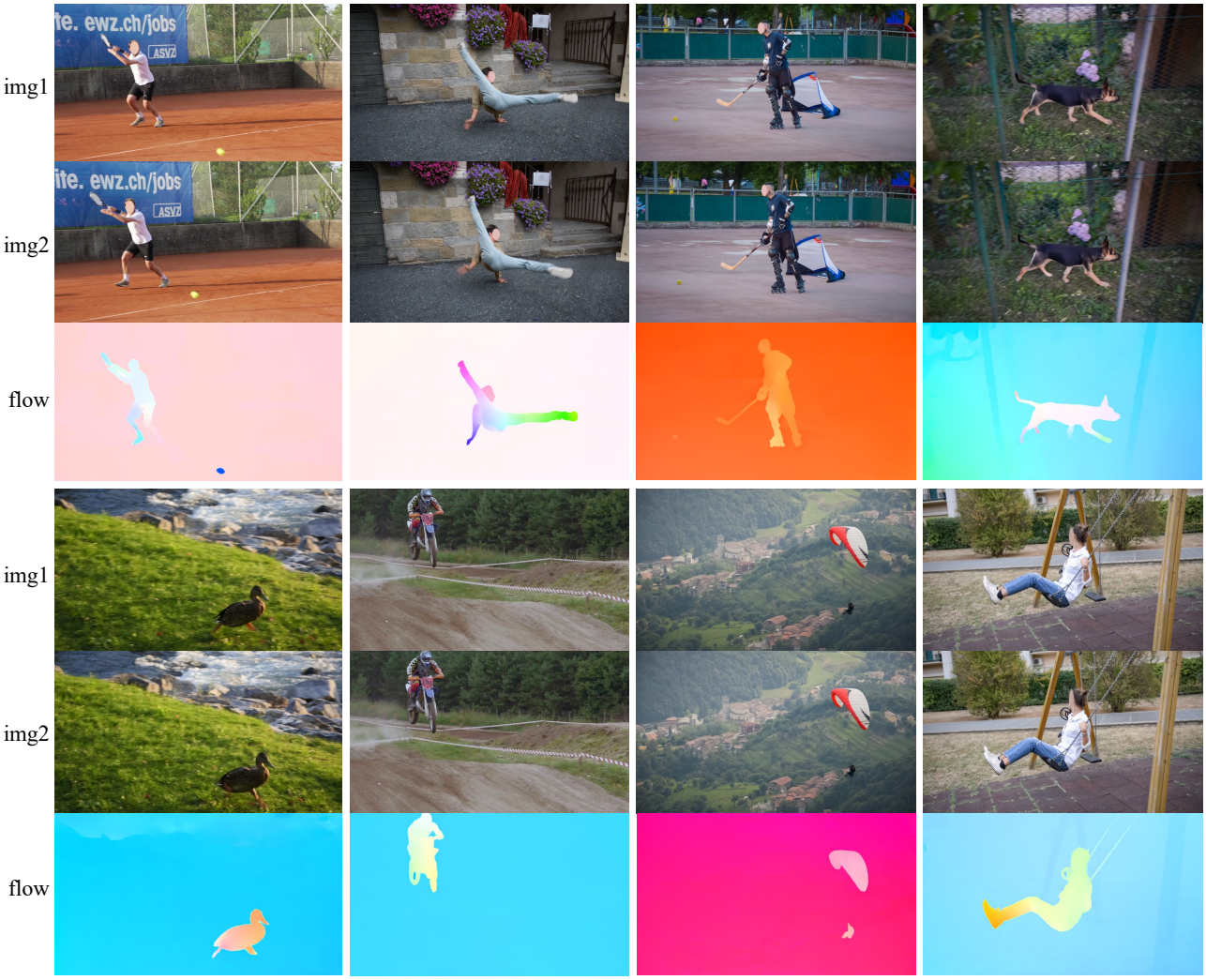

Figure B. **Visual results on DAVIS dataset**.